VersionStamper™

Version 6.5

by

Desaware, Inc.

Rev 6.5.2 (06/05)

Desaware, Inc.
Software License

Please read this agreement. If you do not agree to the terms of this license, promptly return the product and all accompanying items to the place from which you obtained them for a full refund.

This software is protected by United States copyright laws and international treaty provisions.

This program will be licensed to you for your use only. If you, personally, have more than one computer, you may install this program on all of your computers as long as there is no possibility of it being used concurrently at more than one location by separate individuals. You may (and should) make archival copies of the software for backup purposes.

You may transfer this software and license as long as you include this license, the software and all other materials and retain no copies, and the recipient agrees to the terms of this agreement.

You may not make copies of this software for other people. Companies or schools interested in multiple copy licenses or site licenses should contact Desaware, Inc. directly at (408) 404-4760.

Should your intent be to purchase this product for use in developing a compiled Visual Basic program that you will distribute as an executable (.exe) file, review the listing of which files (located below and in the File Description section of the product manual) can be distributed and or modified. If Desaware files are included in your executable program, you must include a valid copyright notice on all copies of the program. This can be either your own copyright notice, or "Copyright © 1994-2005 Desaware, Inc. All rights reserved." .

You may include with your program a copy of the files dwvstamp.vbx, dwvstp16.ocx, dwvstp32.ocx, dwvstp36.ocx, dwvsob36.dll, dwsockvs.dll, dwsockv6.dll, dwsock.dll, dwsock6.dll, dwvercls.dll, dwvercl5.dll, dwvslb4.dll, dwvslb5.dll, dwvslb6.dll, dwvscmp5.dll, and dwvscmp6.dll (the run-time libraries). You may **not** modify these files in any way.

You have a royalty-free right to incorporate any of the sample code provided into your own applications with the stipulation that you agree that Desaware, Inc. has no warranty, obligation or liability, real or implied, for its performance.

Please consult the online Help File under the topic File Descriptions for additional information.

Limited Warranty

Desaware warrants the physical CD and physical documentation enclosed herein to be free of defects in materials and workmanship for a period of sixty days from the purchase date.

The entire and exclusive liability and remedy for breach of this Limited Warranty shall be limited to replacement of defective CD or documentation and shall not include or extend to any claim for or right to recover any other damages, including but not limited to, loss of profit, data or use of the software, or special, incidental or consequential damages or other similar claims, even if Desaware has been specifically advised of the possibility of such damages. In no event will Desaware's liability for any damages to you or any other person ever exceed the suggested list price or actual price paid for the license to use the software, regardless of any form of the claim.

DESAWARE SPECIFICALLY DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Specifically, Desaware makes no representation or warranty that the software is fit for any particular purpose and any implied warranty of merchantability is limited to the sixty day duration of the Limited Warranty covering the physical medium and documentation only (not the software) and is otherwise expressly and specifically disclaimed.

This limited warranty gives you specific legal rights. You may have others, which vary from state to state.

This License and Limited Warranty shall be construed, interpreted and governed by the laws of the State of California, and any action hereunder shall be brought only in California. If any provision is found void, invalid or unenforceable it will not affect the validity of the balance of this License and Limited Warranty which shall remain valid and enforceable according to its terms.

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of Commercial Computer Software - Restricted Rights at 48 CFR 52.227-19, as applicable. Contractor/Manufacturer is Desaware, Inc., 3510 Charter Park Drive, Suite 48, San Jose, California 95136.

Table of Contents

About VersionStamper

This section introduces VersionStamper and the industry wide problems that led to its development. We strongly encourage everyone to read this section before proceeding to use VersionStamper. Since VersionStamper includes support for both VBX and ActiveX, 16 bit and 32 bit Windows, this manual will combine the descriptions of common features where applicable. In cases where they are not applicable, a **VBX ONLY**, **OCX ONLY**, **32 bit,** etc. heading will follow the section heading. Normally, VersionStamper will be used to identify the 32 bit ActiveX, but in instances where only the VBX or 16 bit apply, the appropriate dwvstamp.vbx or dwvstp??.ocx names will be substituted. If you are upgrading from VersionStamper version 1.0, please read the section *Migrating From Version 1.0 to Versions 4.0 and Greater* first.

The Trouble with Distributing Component Based Applications

Once upon a time.....

.... Every application was self-contained.

A program would consist of a single executable (EXE) file. Complex applications might consist of several executables that were chained to each other. One thing, however, was certain - the executables that accompanied a particular application could be used only by that application. For most programs, you could distribute all of the files used by that program without being concerned that other products might interfere with yours.

In the past few years the size of application files has grown dramatically. The Windows operating environment took advantage of the capability *dynamic linking* to allow code modules to be shared by applications. The most important example of this capability is Windows itself - the code modules that contain the functions that make Windows work (the Windows API), could be shared by all Windows applications. A code module that can be shared in this way is called a *dynamic link library* and has the extension .DLL.

Initially, this sharing of files was not a problem. Most applications only used the Windows DLLs, or private DLLs - dynamic link libraries were rarely shared between applications.

Want to Learn More?

VersionStamper is intended for both beginning and expert programmers. The manual is designed so that beginners will be able to use the product in a "boiler plate" manner by simply following the step by step instructions. If you wish to customize the VersionStamper components you will need to have some familiarity with Windows itself, specifically the Windows API. In order to help you quickly come up to speed if you do not already have this information, we will refer in the manual to pages and chapters in "PC Magazine Visual Basic Programmer's Guide to the Windows API" and its successor "Dan Appleman's Visual Basic Programmer's Guide to the Win32 API" which are published by Macmillan Press.

If you would like to learn more about how dynamic link libraries work and why they exist, refer to Chapter 1 in either of these books.

Software Installation of Shared Dynamic Link Libraries

As Windows evolved, Microsoft began to create additional dynamic link libraries that were designed to be shared among its own applications, and still others that were designed to be used by all Windows programmers. These dynamic link libraries contained groups of functions that provide a standard functionality, eliminating the need for each application to implement that functionality independently.

One example of these DLLs is commdlg.dll - the *common dialog library*. This DLL contains a group of common dialog boxes that can be used by any Windows application to perform standard operations such as obtaining a file name or choosing a color.

Each application that used common dialogs would need to distribute the file commdlg.dll, at first because it was not included in Windows, later because Microsoft would introduce updated versions of the DLL that fixed bugs or added new functionality.

What happened when a new version of commdlg.dll appeared? Obviously there is no magical way by which it would instantly be replaced for every application that needed it. Even if you replaced it in all new builds of a product, all of the previous distribution disks would have the older version of the DLL. It would not be unusual for an individual to own several programs that use this DLL, each of which had a different version of the DLL on its distribution disks.

It is not unusual for users to re-install software, either during a system upgrade or to change configurations. Frequently they would install software that included an older version of commdlg.dll on a system that already contained a program that previously installed a newer version. As soon the user attempted to run the program that required the newer version, problems would occur ranging from operational difficulties to general protection faults.

The appearance of Version Resources

Microsoft's answer to this problem was to create a mechanism to place a version description into a code module. This is accomplished by embedding a special resource called *Version Resource* into the code module. A resource is a block of data in a module that can be read from within the program when it is running, or by other programs. A version resource is a block of data that contains a special format that describes the version of a file and additional information about the file. The exact format of this information and how to access it is described in the *Technical Information* section of this manual.

The version resource makes it possible for installation programs to determine if a later version of a DLL is already present on a system. If so, the installation program can avoid overwriting it with an earlier version.

Want to Learn More?

If you would like to learn more about how resources work and why they exist, refer to Chapter 12 in the "Visual Basic Programmer's Guide to the Windows API" or Chapter 15 in the "Visual Basic Programmer's Guide to the Win32 API" and "Dan Appleman's "Visual Basic Programmer's Guide to the Win32 API".

The Trend Towards Component Based Solutions

Microsoft may have provided a version resource capability, but that alone did not solve all of the problems with distributing applications. Even now, many dynamic link libraries are created without version resources or the version resource is not updated on each release. Many applications use installation programs that do not check the version information of existing DLLs.

Still, as long as users had to deal with only a few shared dynamic link libraries, the problem was manageable. This all changed with the appearance of Microsoft's Visual Basic.

Visual Basic is the first product to take full advantage of a new software development philosophy called "Component-Solution" programming. Under this philosophy, programmers take advantage of "off the shelf" software components that implement specific functions. Visual Basic itself is the "glue" that binds these software components.

Under Visual Basic, software components consist of either dynamic link libraries or Visual Basic custom controls (VBXs) or ActiveX controls (OCXs).

This programming philosophy makes an enormous amount of sense. Why write your own communications function library when a single custom control can provide the same capability for a tiny fraction of the price? Visual Basic has literally hundreds of different VBXs and ActiveXs, which have assisted in making it a highly effective programming environment.

The Distribution Crisis

The component-solution framework for programming has had one serious side effect concerning the distribution of Visual Basic applications. Now instead of a few DLLs that are shared by several applications, there are hundreds of DLLs, VBXs and OCXs that may be shared by literally thousands of applications.

And all it takes is a single DLL, VBX or OCX to be missing, or present in an older version (or even an incompatible newer version), for an application to fail. A poorly designed installation program, user error, registration error or change in the user's PATH environment variable are a few of the things that can cause this to occur.

Worse yet, there is no reliable way to identify the failure, since the symptoms of the failure can vary from a minor error to a General Protection Fault or memory exception.

But the problems do not end there. Some applications place software components in their own directory, meaning that you can have several versions of the same OCX, VBX or DLL on your system at the same time. The one that is used may depend upon the sequence in which two applications are run or which component was last registered - leading to a whole list of "intermittent" problems that depend upon the interaction between unrelated applications.

And for those who are using Visual Basic 3.0 or a VBA application, add the final straw - programs created by these development tools do not have a version resource. This means that there has been no reliable way to prevent the overwriting of newer versions of your own programs.

It is not unheard of for technical support personnel to spend literally hours on the phone trying to track down elusive problems that turn out to be nothing more than the presence of an obsolete software component.

Which brings us to VersionStamper.

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 run-time environment.

Embedding Component Version Information into Your Application

VersionStamper makes it possible to embed into the executable or an ASCII text file a complete list of every DLL, OCX, VBX or other software component that is needed by your program. This embedded component information includes the required version information, date/time, size, and other 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 also includes the Conflict Wizard, a standalone utility that uses several different methods to scan your program allowing you to determine a list of the components used by your program, minimizing the chances that you will miss a critical component.

Detecting Component Incompatibilities at Run-Time

Since an executable stamped by VersionStamper "knows" exactly what components it requires at run-time, it is an easy matter for it to confirm that all of those components are available and current. The 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 our quick start solutions and add the standard vervrfy.frm or vervrfy2.frm form and associated modules to their application to provide a default verification check at load time or under user control. More advanced programmers may choose to modify the code, or use the VersionStamper classes, to create their own custom report. You might also provide the user with a step-by-step procedure as to 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 a sample "Rescue" program which is able to compare the component requirements of an executable with the run-time environment without actually loading that executable. The rescue program includes complete source code and can be fully customized to suit your own needs.

VersionStamper includes Internet features that allow your end-user to access an updated dependency list from a web site, report component conflicts via email and even automatically update their components via the world wide web or from an FTP site.

Additional Tools and Information

VersionStamper includes the program, 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 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.

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 a version resource, embedding component information and adding a run-time conflict report in a few easy steps.

Compatibility

VersionStamper is designed for use with Visual Basic 3.0 and higher, Microsoft Visual C++, Delphi, and other products that support the Visual Basic 1.0 VBX file format. The ActiveX (OCX) edition may be used with Visual Basic 4.0 and higher, and other products that support the full ActiveX control container specifications.

The ability to embed version resources is only provided for Visual Basic 3.0 itself. This ability is not required by Visual Basic 4.0 and later, Microsoft Visual C++ and Borland C++ and Delphi (those products come with other tools that allow the creation of version resources). VersionStamper can, however, be used to embed component information into applications created with these languages in order to support the run-time verification features of this product.

Customer Support

We at Desaware have one very simple company policy - we do our best to treat our customers as we would like to be treated (after all, we are programmers too).

If you have purchased this software directly from Desaware and you feel that VersionStamper is not for you or you are otherwise dissatisfied, please feel free to return it for a full refund (if you purchased it elsewhere you will need to contact your dealer for return or refund information - also, we reserve the right to limit this offer to 30 days from the invoice date). Your satisfaction is important to us, and we are well aware that this is a very unusual product and not appropriate for everyone.

For information on customer support and last minute changes, refer to the readme.wri file on the VersionStamper CD. This file is compatible with write.exe (included with each copy of Windows).

There is a saying in the software world that no non-trivial program is completely bug free. The corollary to that saying is that no program with more than 10 lines in it is non-trivial. VersionStamper is emphatically non-trivial....

VersionStamper has undergone extensive testing in all the existing Windows operating systems to make it as bug free as possible. Nevertheless, it is possible that some may have crept through. Please check the VersionStamper FAQ page at Desaware's web site (http://www.desaware.com/VersionStamperFAQL3.htm) for latest breaking news on bug fixes and other issues. Also, please run the VersionStamper Run Time Files Update utility (VsUpdate.exe) to ensure that you have the latest VersionStamper release files. Please contact us via email or fax if you encounter problems, and include all of the steps needed to reproduce the problem. Also, if there are any files needed to reproduce the error, include them in your email. Once we are able to duplicate a problem, we will provide you with a fix as quickly as possible.

Please address all correspondence to:

Desaware Inc.
3510 Charter Park Drive, Suite 48
San Jose, CA 95136

Phone (408) 404-4760, Fax (408) 404-4780

Internet: http://www.desaware.com, or support@desaware.com

Register! Register! Register!

We've found that the person who ultimately uses a software package is frequently not the person who purchased it. Therefore we really need your registration card. This will allow us to send you information about upgrades.

But we can't send this information to you without knowing who you are!

And please send us your suggestions for features that you would like to see in future editions of this product!

Installation

VersionStamper comes with a Windows installation program. **Refer to the readme file on the CD for the latest information regarding installation.**

- Place your VersionStamper CD in your CD ROM drive.

- The setup program should automatically start if your computer's autorun mode is enabled. Otherwise, use the Start Button's Run... command in Windows to run d:\setup.exe or e:\setup.exe (depending on the drive letter assigned to your CD ROM drive). You can also use Windows Explorer or the File Manager to initiate this program.
- Select the desired setup option to start the installation.
- The setup program will prompt you for a destination directory for the VersionStamper sample files and utilities. The VersionStamper ActiveX controls and components will be installed and registered in the appropriate System directory.
- Follow any further directions in the setup program. A summary of files installed will appear in the install.log file in your VersionStamper directory.

Installation programs are tricky - and we have found that occasionally a system is configured in such a way that the installation program fails. Please refer to the readme file for the latest information on these situations, and for instructions for manual installation.

The directory containing the VersionStamper sample files may contain files **readme.txt** or **readme.wri**. These files, if present, will contain recent information that could not be incorporated into the manual at time of printing. Use the Windows 'Notepad' program to view readme.txt, and the Windows 'Write' program to view readme.wri.

Which VersionStamper File Should I Use?

VersionStamper includes several different editions of the ActiveX control and component. The files or edition you should use depends on whether you are using 16 or 32 bit, and whether or not your programming environment is Visual Basic.

16 Bit

If you are developing in a 16 bit environment, you are limited to using either the dwvstamp.vbx or dwvstp16.ocx files. The dwvstamp.vbx file is the VBX edition of VersionStamper. This file does not include many of the new features offered in the OCX edition and should be used only in environments that support VBXs such as Visual Basic 3.0 or Visual Basic 4.0 – 16 bit. The dwvstp16.ocx file is the 16 bit OCX edition of VersionStamper. This file should be used only in environments that support 16 bit OCXs such as Visual Basic 4.0 – 16 bit. The 16 bit edition controls do not support many of the new VersionStamper features.

32 Bit

If you are developing in a 32 bit environment, you can use either the dwvsob36.dll, dwvstp36.ocx, or dwvstp32.ocx files.

File	Description
Dwvsob36.dll Desaware VerScan Component Type Library	This file is the 32 bit ATL ActiveX component edition of VersionStamper. This file can be used in any 32 bit development environment that supports ActiveX components.

Dwvstp36.ocx Desaware VersionStamper 6 Control	This file is the 32 bit ATL ActiveX control edition of VersionStamper. This file supercedes the dwvstp32.ocx file and includes additional features. This file can be used in any 32 bit development environment that supports ActiveX controls. The dwvstp36.ocx file requires distributing the dwvsob36.dll file.
dwvstp32.ocx Desaware VersionStamper Control	This file is the 32 bit MFC ActiveX control edition of VersionStamper. This file requires distributing the MFC40.dll and MSVCRT40.dll files. This file should be used only when you are planning to release both 16 and 32 bit versions of your application and are developing in a dual 16 and 32 bit environment such as Visual Basic 4.0. Otherwise, it is more advantageous to use either the dwvstp36.ocx or dwvsob36.dll files.

Additional 32 Bit Components

VersionStamper includes a component class library that wraps the VersionStamper control or component and provides additional support in areas such as conflict report resolution, file list parsing, and internet support. This component class library was written in Visual Basic and includes source code. You can add the required class files to your Visual Basic project, or reference the compiled component. This component class library is also compiled in several different editions.

File	Description
dwvscmp6.dll "Desaware VersionStamper ATL SW VB6 Components"	This is the latest VersionStamper Class Library file. This edition is compiled in Visual Basic 6.0. The dwvscmp6.dll file requires the dwvsob36.dll, dwsock6.dll and the Visual Basic 6.0 run time library files. You should use this file if you are developing in Visual Basic 6.0 or in NON-Visual Basic environments.

dwvscmp5.dll "Desaware VersionStamper ATL SW VB5 Components"	This file is similar to the dwvscmp5.dll file except it was compiled in Visual Basic 5.0 instead. The dwvscmp5.dll file requires the dwvsob36.dll, dwsock.dll and the Visual Basic 5.0 Service Pack #2 or later run time library files. You should use this file if you are developing in Visual Basic 5.0 or in NON-Visual Basic environments.
dwvslb6.dll "Desaware VersionStamperVB 6 ATL Components"	This file is similar to the dwvscmp6.dll file minus some of the newest changes (such as HTTP Proxy Support). The dwvslb6.dll file requires the dwvsob36.dll, dwsockv6.dll and the Visual Basic 6.0 run time library files. You should use this file only if you are updating an existing VB project that already uses this file. You can easily migrate to the dwvscmp6.dll file.
dwvslb5.dll "Desaware VersionStamperVB 5 ATL Components"	This file is similar to the dwvslb6.dll file except it was compiled in Visual Basic 5.0 (without any Service Packs). You should use this file only if you are updating an existing VB project that already uses this file or if you are developing new Visual Basic 5.0 projects without any Service Packs installed.
dwvslb4.dll "Desaware VersionStamperVB 4 ATL Components"	This file is similar to the dwvslb5.dll file except it was compiled in Visual Basic 4.0. You should use this file if you are developing new Visual Basic 4.0 projects.
dwvercl5.dll "Desaware VersionStamperVB 5 Components"	This file is the original component class library compiled in Visual Basic 5.0. You should use this only for existing Visual Basic 5.0 projects that already use this component (you can also migrate to the dwvscmp5.dll or dwvslb5.dll). The dwvercl5.dll file requires the dwvstp32.ocx, dwsockvs.dll and the Visual Basic 5.0 run time library.

dwvercls.dll "Desaware VersionStamper Components"	This file is the original component class library compiled in Visual Basic 4.0. You should use this only for existing Visual Basic 4.0 projects that already use this component (you can also migrate to the dwvslb4.dll). The dwvercls.dll file requires the dwvstp32.ocx, dwsockvs.dll and the Visual Basic 4.0 run time library.
dwsock6.dll "Desaware Winsock Library vb6"	This file includes HTTP and FTP support for retrieving files through the Internet. This file is licensed for use only with dwvscmp6.dll and is required by that file. This file also requires dwspyv6.dll, sockintf.dll, dwspy5.dll and the Visual Basic 6.0 run time library.
dwsock.dll "Desaware Winsock Library"	This file includes HTTP and FTP support for retrieving files through the Internet. This file is licensed for use only with dwvscmp5.dll and is required by that file. This file also requires dwspyvb.dll, sockintf.dll, dwspy5.dll and the Visual Basic 5.0 Service Pack #2 or later run time library.
dwsockv6.dll "Desaware Winsock Library (VS edition for VB 6)"	This file includes HTTP and FTP support for retrieving files through the Internet. This file is licensed for use only with dwvslb6.dll and is required by that file. This file also requires dwspyv6.dll, sockintf.dll, dwspy5.dll and the Visual Basic 6.0 run time library. This file is a subset of our SpyWorks Winsock ActiveX component.
dwsockvs.dll "Desaware Winsock Library (VS edition for VB 5)"	This file includes HTTP and FTP support for retrieving files through the Internet. This file is licensed for use only with the dwvercls.dll, dwvercl5.dll, dwvslb4.dll, and dwvslb5.dll files and is referenced by those files. This file also requires dwspyvb.dll, sockintf.dll, dwspy5.dll and the Visual Basic 5.0 run time library. This file is a subset of our SpyWorks Winsock ActiveX component.

File Descriptions

The following files and controls may be distributed with your compiled Visual Basic application without payment of license fees according to the terms on the License Agreement.

File	Description
dwvstp36.ocx, dwvstp32.ocx, dwvstp16.ocx	The primary VersionStamper (ActiveX edition) custom control. It should be registered in the registry, it is typically installed in the System directory. If you are creating a 32 bit only application, you should use the dwvstp36.ocx file.
dwvsob36.dll	The primary VersionStamper scanning engine COM component. This file is required if you are using dwvstp36.ocx. It can also be used by itself. This file is also required if you are using dwvslb5.dll, dwvslb6.dll, dwvscmp5.dll, or dwvscmp6.dll.
dwvscmp5.dll, dwvscmp6.dll dwvslb5.dll, dwvslb6.dll, dwvercl5.dll, dwvercls.dll,	VersionStamper classes run-time ActiveX DLL. It should be registered in the registry, it is typically installed in the System directory. These files are all similar, they are compiled in different VB editions. If you are using VB 6, you should be using the dwvscmp6.dll file. If you are using VB 5 (Service Pack 2 or later), you should be using the dwvscmp5.dll file. dwvercls.dll is the VB 4 compiled version, dwvercl5.dll and dwvslb5.dll are the earlier VB 5 compiled version, dwvslb6.dll is the earlier VB 6 compiled version.
dwsock.dll, dwsock6.dll, dwsockvs.dll, dwsockv6.dll	Desaware SpyWorks WinSock component (Version-Stamper edition). This WinSock component allows VersionStamper to retrieve files from a URL for file verification and perform FTP downloads for auto updates. This file can be distributed only with a VersionStamper application. dwsock.dll is used by dwvscmp5.dll, dwsock6.dll is used by dwvscmp6.dll, dwsockvs.dll is used by dwvslb5.dll and dwvercl5.dll, dwsockv6 is used by dwvslb6.dll. If you are starting a new project, use either the dwsock.dll or dwsock6.dll.

dwvstamp.vbx	The primary VersionStamper (VBX edition) custom control. It should be installed in a directory that is in your PATH environment setting (typically the System directory).
VsRescue.exe	This is a "last-resort" rescue application. This 32 bit rescue application can scan a file that has a VersionStamper embedded file list. It should be able to execute under most scenarios even when your primary application cannot run. You may distribute this file with your application solely for the purpose of providing file conflict information for your application.

The following files may NOT be distributed and are subject to the terms in your license agreement. However, you may incorporate the source code from these files (where applicable) into your own application.

File	Description
dwverd36.dll, dwverd32.dll, dwverd16.dll	Design time library and license file for VersionStamper (ActiveX edition).
dwverdes.dll	Design time library and license file for VersionStamper (VBX edition).
Verclass (dwVer*.vbp) projects and associated files.	These projects include helper classes used for a variety of functions related to reporting file conflicts. Commonly referred to as the VersionStamper Component Library.
vervrfy.frm vervrfy.bas	These modules demonstrate how to use the Version-Stamper control to scan for component conflicts at runtime. These modules can be incorporated directly into your own applications to provide this functionality. You may distribute these files as is only if you are also distributing your own source code under a valid copyright.

vervrfy2.frm vervrfy.bas	These modules demonstrate how to use the Version-Stamper control to scan for component conflicts at runtime and provide a typical verification report at the same time. These modules can be incorporated directly into your own applications to provide this functionality. You may distribute these files as is only if you are also distributing your own source code under a valid copyright.
verresq.vbp and associated files.	This project demonstrates a more robust scheme for reporting component file conflicts. It can also be used as a stand-alone utility for doing component verification on any file that contains embedded component information. This project includes full source code.
verinfo.vbp and associated files.	This project demonstrates how to read the version resource information from any executable. It is also a stand-alone utility for browsing the version resource and embedded component information for any executable. This project includes full source code, but requires that SpyWorks be installed on your system to load. verinfns.mak (listed next) is identical except that it does not require SpyWorks. (SpyWorks is used to implement the status bar in this project.)
verinfns.vbp and associated files.	This project is almost identical to verinfo.mak. It uses a different MDI form that does not include a status bar, and can be loaded without having SpyWorks on your system.
vsrtdemo.vbp and associated files.	This sample project demonstrates a new feature of VersionStamper - how to specify components to verify during run-time.

vrsqtest.mak and associated files.	This sample project demonstrates a minimal scheme for reporting version conflict information. It shows how to add the vervrfy.frm and vervrfy.bas modules to an application to detect conflicts.
vrsqtst2.mak and associated files.	This sample project demonstrates a more robust scheme for reporting version conflict information. It shows how to add the vervrfy.frm and vervrfy.bas modules to an application to detect conflicts.
versplsh.mak and associated files.	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.
VsReplace and associated files.	This sample project demonstrates how to use some of the more advanced features of the VerSmartReplaceFile class object.
VsWebLst and associated files.	This sample project demonstrates how to retrieve a file list to verify from a web address. It parses the file list and reports any component file conflicts.
VsFileLs and associated files.	This sample project demonstrates how to retrieve a file list to verify from a file on a local drive. It parses the file list and reports any component file conflicts.
VsMail and associated files.	This sample project demonstrates how to send an email to report component file conflicts.

VSAutoUpdate, AutoUpdate and associated files.	This sample project demonstrates how to create a self-updating application. It verifies the required files for an application by retrieving the file list from a web address, then it uploads newer files via an FTP location.
VsWizard.exe	The VersionStamper Conflict Wizard. This utility is used to perform an analysis of a particular Visual Basic project, EXE file, or other binary file and retrieve a list of dependent files required.
VerDepend.exe	This is a utility program used by VersionStamper to scan a binary file for direct dependencies.
dwvstamp.hlp	On-line help file for VersionStamper. This file contains context sensitive help for the VersionStamper control, the full manual text, and other supplemental information.

Files Required for Distribution

dwverd36.dll or dwverd32.dll	**You are NOT allowed to distribute these files.**
dwverdes.dll or dwverd16.dll	**You are NOT allowed to distribute these files.**
Other EXE files included with VersionStamper.	**You are NOT allowed to distribute these files.**

The following table summarizes the list of files required to be distributed with the VersionStamper components.

Files	Dependent Files
Dwvsob36.dll	Requires oleaut32.dll version 2.20.4054.1 or greater and stdole2.tlb version 2.20.4054.1 or greater. (These two files were not included with the original Windows 95 system but are included in Visual Basic 5.0 or later and Internet Explorer 3.1 or later.)
Dwvstp36.ocx	Requires dwvsob36.dll.
Dwvstp32.ocx	Requires msvcrt40.dll, mfc40.dll.
Dwvscmp6.dll	Requires dwvsob36.dll, dwsock6.dll, dwspyv6.dll, dwspy5.dll, sockintf.dll, msvbvm60.dll, and oleaut32.dll.
Dwvscmp5.dll	Requires dwvsob36.dll, dwsock.dll, dwspyvb.dll, dwspy5.dll, sockintf.dll, msvbvm50.dll (Service Pack #2 or later edition), and oleaut32.dll.

Dwvslb6.dll	Requires dwvsob36.dll, dwsockv6.dll, dwspyv6.dll, dwspy5.dll, sockintf.dll, msvbvm60.dll, and oleaut32.dll.
dwvslb5.dll	Requires dwvsob36.dll, dwsockvs.dll, dwspyvb.dll, dwspy5.dll, sockintf.dll, msvbvm50.dll, and oleaut32.dll.
dwvslb4.dll	Requires dwvstp36.ocx, dwsockvs.dll, dwspyvb.dll, dwspy5.dll, sockintf.dll, vb40032.dll, msvbvm50.dll, and oleaut32.dll.
dwvercl5.dll	Requires dwvstp32.ocx, dwsockvs.dll, dwspyvb.dll, dwspy5.dll, sockintf.dll, msvbvm50.dll, and oleaut32.dll.
dwvercls.dll	Requires dwvstp32.ocx, dwsockvs.dll, dwspyvb.dll, dwspy5.dll, sockintf.dll, vb40032.dll, msvbvm50.dll, and oleaut32.dll.
dwvstp16.ocx	Requires oc25.dll, wowglue.dll, and w32sys.dll.
dwvstamp.vbx	Requires wowglue.dll.
VsRescue.exe	Requires dwvsob36.dll.

Strategies for Using VersionStamper

VersionStamper contains tools that can be used in a number of different ways to assist you with distributing your Visual Basic applications reliably.

Quick Start - Our Standard "No Code" Approach

This chapter is intended to provide a quick-start approach that involves minimal effort on your part and a minimum amount of coding.

For Sophisticates - Customized Solutions

VersionStamper takes an unusual approach to detecting and reporting version conflicts. The detection capability is almost entirely built into the VersionStamper custom control. However we designed this control in such a way that the detection itself can be customized not only by the configuration of the control, but by adding your own Visual Basic code to events that are triggered during the verification process. The reporting capability is implemented using Visual Basic code in such a way that you can either plug in our standard reports, customize them, or create your own from scratch.

This approach makes it possible for VersionStamper to work as a simple "canned" solution for some, while providing enormous flexibility for those who wish a custom approach towards resolving version conflicts.

For those who wish to customize the verification process, we direct you to the following sources:

This manual: especially the chapter on the VersionStamper control where the verification process and events are described in detail.

The verresq, vrsqtest, vrsqtst, versplsh, vsrtdemo.vbp, autoupdt.vbp or vsupdate.vbp, vsreplace.vbp, vsfilels.vbp, vsweblst.vbp, and vsmail.vbp projects: These projects are included with complete source code and we encourage you to customize them to suit your own needs.

Tools

VsWizard - VersionStamper Component Conflict Wizard

VsWizard is a utility which can scan your Visual Basic project, EXE file, or other binary file to retrieve a list of the components required. You can use this utility to generate a list of files required, then specify warning conditions for the files you want VersionStamper to verify. The file list is output to an ASCII text file which can be read at run-time by VersionStamper, or embedded into an EXE file.

VsRescue - ATL Based Component Incompatibility Scan Program

The VsRescue program is a stand-alone utility that is similar to the other Visual Basic verification files and programs that are included with this package. The difference is that this is an ATL-based edition and requires minimal run-time files so it should be able to run even in cases where your primary application cannot run due to major file conflicts.

VerInfo - Version Information Spy Program

The VerInfo program can be used to view the version resource information for any file. It can also be used to display the embedded component information for each file. This project includes complete source code.

VerResQ - Component Incompatibility Scan Program

The VerResQ program is a stand-alone utility that is almost identical internally to the other verification files and programs that are included with this package. It adds the ability to select which files to verify, and to print the report or write it to a file. This project includes complete source code.

VerDepend - File Dependency Scanner Program

VerDepend is a utility which scans an executable or dynamic link library (DLL) file and extracts the dependency files for the scanned file. Each dependent file is in turn scanned for direct dependencies to ultimately produce a complete list of dependent files. Dependency relationships in addition to detailed version information for each file may be saved in a text file.

Quick Start Samples

There are several methods that can be used to add file verification to your application. This section outlines detailed step-by-step instructions on several of the common methods.

If you have used previous editions of VersionStamper, you are probably familiar with the "embedded" method. In this method, you use the VersionStamper control to interactively create a list of files to verify in addition to the warning conditions for each file. This is one of the simplest methods with which to begin. The advantage of this method is that the file list is embedded into your EXE file so you can be sure that the correct file list will be verified. Also, this method allows other VersionStamper utilities to verify your EXE file in case the conflict is so great that your EXE cannot even load.

Another method that can be used to verify a file list is the "dynamic run-time" method. This method allows you to specify the files to verify and warning conditions for each file dynamically at run-time. Generally, this involves reading the file information from another file during run-time. The advantage of this method is that you have more control over which files are verified and which warning conditions are used. For example, you can check the operating system or the user's license before deciding on which files to verify. Also, you can have your application retrieve the file information list from a central location such as a server or web site. This way, you can update the file list once, and all instances of your application will be updated to use the new verification criteria.

Embedded Verification

There are four basic steps in using this method to add file verification to your application.

1. Add the *Desaware VersionStamper 6 Control* (dwvstp36.ocx) to your application project and place it on a form. If you are planning to compile both 16 and 32 bit executables, you can use the dwvstp16.ocx or dwvstp32.ocx instead, these files can be used interchangeably for 16 and 32 bits where as the dwvstp36.ocx is 32 bit only and is not compatible with the other two files. Refer to the Files Description topic for more detailed explanation of these files.

If your development environment does not support ActiveX controls, does not allow placement of ActiveX controls or is formless, then you need to use the Dynamic Verification method in order to implement file verification.

2. Create the file list and warning conditions.

   You specify a list of files to verify by using the Select Files form. To open the Select Files form, click on VersionStamper's "SelectFiles" property to bring up VersionStamper's Property Page, then click on the "Select Files" button to bring up the Select Files form.

   If you had earlier used the VersionStamper Conflict Wizard to generate a VersionStamper Script File, you can import that file list by selecting the "Input File List" button and selecting the script file. If you did not use the Wizard to generate a VersionStamper Script File, you can manually add files to be verified by selecting one or more files from the *Available* list box then the *Add* button, or the *Select Additional Files* command button.

   If you are using Visual Basic, you can also use VersionStamper's *Scan Visual Basic Project* feature to create a list of dependent files referenced by your Visual Basic project.

   NOTE: If you are not certain which files are required by your application, we highly recommend that you use the VersionStamper Conflict Wizard to generate a list of files for you (even though you may not plan to verify all the files).

   The *Selected* list box contains the files that will be verified. Select any file in the *Selected* list box to review or edit verification attributes for that file. You can also change the file's reference information (such as version number, date/time, size, etc) by double-clicking on the entry in the *Selected* list box.

   For more detail information on the Select Files form, refer to the Select Files form.

3. Add code to perform the verification and detect file conflicts.

   To start a verification, you can use the VersionStamper control's VerifyMode or VerifyFile functions.

You can initialize the VerifyMode property to “1 – On Load” if you want the VersionStamper control to start the file verification as soon as the control is completely loaded. Or, set the VerifyMode property equal to “2” in your code to start the file verification at any time. The VersionStamper control will verify its own file list when the VerifyMode property is set. This method is used for verifying files within your own application.

VerStamp1.VerifyMode = 2

You can also start a file verification by calling VersionStamper’s VerifyFile method and passing it the path and file name of an EXE file that contains VersionStamper embedded file information. Usually, you would set the file name to your application’s file name, but you can set it to any file. This method allows you to create a separate “Rescue” application or “Splash Screen” start up application that can verify the files for your primary application. **NOTE**: When using this method, you need to re-compile your application each time you change any file list information in order for the changes to be written to the EXE file.

VerStamp1.VerifyFile = App.Path & "\MAINAPP.EXE”

For example, you can place a VersionStamper control on a form (which does not ever need to be loaded) in your project and use that control to embed the file list into your executable file. Then you can create a separate “Verification” application that displays a splash screen while it is verifying the files in your main executable. You would place the “Verification” application in the same directory as your primary application. You can organize it so that the verification application does a verification of your primary application first.

If no conflicts are found it will then “Shell” your main application with a command line parameter (to indicate that a verification was performed prior to running the primary application). If conflicts were found, you have the option of warning your user (giving them the option to proceed or stop), attempt an update (via the internet or intranet), or have them contact your technical support with the conflict report information. Refer to the VsSplash project for an example.

The file verification can be performed at any time, but depending on how many files you are verifying, the speed of the computer, and where the files are searched, this process may cause a noticeable delay in your application.

4. Resolve a conflict report.

   When VersionStamper determines a file conflict, it triggers the FileConflict event. Usually, when a FileConflict event is triggered, you would want to know which file caused the conflict. The ReferenceFile parameter in the FileConflict event contains the file name that you are verifying. The FoundFile parameter contains the path and file name of the file found on the current machine, this parameter string is empty if the file was not found. Also, you would probably be interested in what type of conflict was caused by this file. The Flags parameter is a Long value containing bit masks which describe the type of conflict encountered for this file. Also during this event, you can retrieve additional file conflict information by accessing VersionStamper's properties. VersionStamper includes several sample Visual Basic templates that retrieve the file conflict information and formats it to produce a simple conflict report. Refer to the VersionStamper sample projects for examples.

   When this event is triggered, you normally would just increment a conflict counter and log the results for later use. After all the files have been verified, VersionStamper triggers the EnumComplete event. At this time, you would check your conflict counter to see whether any conflicts were found. If so, you can compile the conflict information logged earlier and display some type of conflict report to your user. The information you decide to report (and resulting actions) is entirely up to you. Be sure to take into consideration the knowledge your users have of computers, shared files, file locations, etc.

   You can also immediately halt the verification process when a FileConflict event is triggered by setting the StopVerify parameter to True. You may want to do this if you are verifying a number of files and the conflict found would cause a major problem, or if you do not want your users running your application with any incorrect files.

So let's create a quick Visual Basic sample.

1. Start with a new EXE project and add the VersionStamper control to your project, then place the control on the main form.

2. Select the VersionStamper control and then select the "Select Files" property. This will open the Property Page. Select the "Select Files" button to open the Select Files form. A list of DLLs and OCXs should appear in the "Available" list box. Select the comdlg32.ocx and comdlg32.dll files (or some other random files) and "Add" them to the "Selected" list box. Usually, the default warning conditions assigned to the selected files would be sufficient. For this sample, we'll force an "older file version" condition. Highlight the cmndlg32.dll file and double click on it. The "Edit File Information" form should now be displayed. The purpose of this form is to allow you to override the file's reference information. Increment the file's "Version Number" and select the OK button. This makes the reference version number for the file greater than what actually exists on the hard drive which should produce a conflict error during the verification. Select the OK button in the "Select Files" form to close the form, then the OK button again to close the Property Page form.

3. First, we'll do a very simple verification. In the "Form_Load" event of the main form, add the following line of code:

   VerStamp1.VerifyMode = vstManualVerify (or 2).

4. In the "VerStamp1_FileConflict" event, add the following line of code:

   Debug.Print ReferenceFile & ": conflict found with " & FoundFile & ", conflict code = " & Hex$(Flags).

   You can also add a label to your form and assign the Label's Caption property the same string you assign to the Debug command line.

   Run this sample and look in your Debug or Immediate window. You should see the following message:

   COMDLG32.DLL: conflict found with c:\windows\system\ COMDLG32.DLL, conflict code = 4

This indicates that VersionStamper found an older version of the comdlg32.dll file. Refer to the FileConflict event for more detailed information on the event parameters.

Using the VersionStamper template files for reporting

We will do a similar example here, but in this case we will use the VersionStamper template files for reporting conflicts. Steps 1 and 2 are identical to the previous example.

Next, add the following files to your project: english.bas, vervrfy.bas, and vervrfy2.frm. These files are located in the VersionStamper "Template" sub directory.

3. In the "Form_Load" event of your main form, add the following lines of code:

```
FileToCheck = App.Path & "\VsSample.exe" ' Use the name of your executable here'
Load VerVrfy2

If ConflictFilesFound Then
   VerVrfy2.Show 1
Else
   Unload VerVrfy2
End If
```

Now compile the project and assign the executable file the same name that you assigned to the "FileToCheck" string in Step 3 above. Run the project. A form should appear listing the cmndlg32.dll file as a conflicting file in the top list box. When you highlight that file, additional detail information regarding the conflict will appear below.

This method allows you to use the template files for file verification and conflict reporting. The VersionStamper control you place on your main form contains the file list you want to verify. This list is embedded into the EXE file each time you compile your project. A VersionStamper control also exists on the VerVrfy2.frm file, but that control is used strictly for verification and conflict reporting.

Dynamic Verification (OCX Edition Only)

NOTE: This method is not supported in the VBX edition of VersionStamper.

There are four basic steps when using this method to add file verification to your application.

1. Add the *Desaware VersionStamper 6 Control* (dwvstp36.ocx) to your application project and place it on a form **OR** add the *Desaware VerScan Component Type Library* (dwvsob36.dll) reference to your application project and create a new instance of the VerScan object.

   If your development environment supports events triggered by ActiveX components, then you would probably want to choose the VersionStamper ActiveX component (dwvsob36.dll) rather than the ActiveX control (dwvstp36.ocx). The VersionStamper control actually requires the VerScan component so using just the VerScan component will result in one less file to distribute. After you add the VerScan reference to your project, you will declare the VerScan class as an object that triggers events. This declaration may vary depending on the development platform. In Visual Basic and other VBA platforms, the declaration is as follows:

   Dim WithEvents VersionStamperComp As VerScan

   You can then create a new instance of this object. The code to do this may vary depending on the development platform. In Visual Basic and other VBA platforms, the code is as follows:

   Set VersionStamperComp = New VerScan

2. Create the file list and warning conditions.

   There are several ways to dynamically create the file list and warning conditions. The most common method is to read the data from a file that contains the file information. The VersionStamper Conflict Wizard can generate a file list for your application that can be used by VersionStamper. VersionStamper also includes additional classes that support parsing from a text file that contains file list information. The file list information must be formatted in a command line format that VersionStamper recognizes.

3. Add code to perform the verification and detect file conflicts.

   You can use the VersionStamper control's or VerScan object's VerifyObjectFile methods to verify a particular file. This method call requires several parameters which consist of the file information for the file you want to verify. An example command line call to VerifyObjectFile2 for the comdlg32.dll file is specified below.

   ```
   VerStampComp1.VerifyObjectFile2 "comdlg32.dll", "", &H8009, 0,
   &H40000, &H5650005, "4.0.1381.5", "", 0, 185104, 1997, 4, 30, 21,
   0, 0
   ```

   This command line states that a warning be triggered if it found cmndlg32.dll with a *version number* earlier than 4.0.1381.5. The file's version number is actually specified by the "&H40000" and "&H5650005" numeric parameters, not the "4.0.1381.5" string parameter.

   When this method is called, VersionStamper immediately performs a verification on the specified file. Program flow does not continue until the verification process is completed, unless you have set the "Delay" property to True. In which case, it will return immediately and perform the verification at a later time. During the verification, if a conflict is found, the FileConflict event is triggered. When the verification is completed, the EnumComplete event is triggered before it returns from the VerifyObjectFile method call (unless the Delay property is set).

4. Resolve the conflict.

   When VersionStamper determines a file conflict, it triggers the FileConflict event. Usually, when a FileConflict event is triggered, you would want to know which file caused the conflict. The ReferenceFile parameter in the FileConflict event contains the file name that you are verifying. The FoundFile parameter contains the path and file name of the file found on the current machine, this parameter string is empty if the file was not found. Also, you would probably be interested in what type of conflict was caused by this file. The Flags parameter is a Long value containing bit masks which describes the type of conflict encountered for this file. Also during this event, you can retrieve additional file conflict information by accessing VersionStamper's properties. VersionStamper includes several sample Visual Basic templates that retrieves the file conflict information and formats them to produce a simple conflict report. Refer to the VsSample sample project for an example.

   When this event is triggered, you normally would just increment a conflict counter and log the results for later use. After you have verified all the required files, you would check your conflict counter to see whether any conflicts were found. If so, you can take all the conflict information logged earlier and display some type of conflict report to your user. The information you decide to report (and resulting actions) is entirely up to you. Be sure to take into consideration the knowledge your users have of computers, shared files, file locations, etc.

So let's create a quick sample.

1. Start with a new EXE project and add either the VersionStamper control or the VerScan reference to your project. Also add the VsRTCons.bas file to your project, this file contains several constants declarations and is located in the VersionStamper "Template" sub directory. Place the control on the main form OR make a global declaration of the VerScan object (WithEvents).

   Dim WithEvents VerStamp1 As VerScan

2. For this first sample, we will hard code the file's information into the VerifyObjectFile2 method so we don't need to do anything in this step.

3. For the sake of comparison with the embedded method, we will verify the comdlg32.dll and comdlg32.ocx files. You can attach the following code to the Form_Load event or to the event when you want to perform the verification.

```
Set VerStamp1 = New VerScan
VerStamp1.VerifyObjectFile2 "comdlg32.dll", "",
RTWARN_OLDERFILE Or RTWARN_VERSIONCOMPARE Or
RT_NOREGISTRYSEARCH, RTPATH_DEFAULT, &H40000,
&H5650005, "4.0.1381.5", "", 0, 185104, 1997, 4, 30, 21, 0, 0
VerStamp1.VerifyObjectFile2 "comdlg32.ocx", "{00020430-0000-
0000-C000-000000000046}#2.0#0", RTWARN_OLDERFILE Or
RTWARN_VERSIONCOMPARE Or
RTWARN_NOTINREGISTRY, RTPATH_DEFAULT, &H60000,
&H510045, "6.0.81.69", "", 0, 140096, 1998, 6, 23, 23, 0, 0
Set VerStamp1 = Nothing
```

The two "Set VerStamp1 =…" lines are not necessary if you are using the VersionStamper control since it will already exist on your form.

The parameters passed to the VerifyObjectFile2 methods are generally not hard coded, but read from a file or retrieved from some other source. For testing purposes, we will force a file conflict by specifying a newer version number for the comdlg32.dll file. At the time this manual was written, the shipping version of comdlg32.dll was 4.0.1381.5. We will change that to 5.0.1381.5 by incrementing the major version number in the vsMS parameter. The modification results in the following line:

```
VerStamp1.VerifyObjectFile2 "comdlg32.dll", "",
RTWARN_OLDERFILE Or RTWARN_VERSIONCOMPARE Or
RT_NOREGISTRYSEARCH, RTPATH_DEFAULT, &H50000,
&H5650005, "4.0.1381.5", "", 0, 185104, 1997, 4, 30, 21, 0, 0
```

Note that we did not have to change the "4.0.1381.5" version string because we are not checking that in this case.

4. In the "VerStamp1_FileConflict" event, add the following line of code:
Debug.Print ReferenceFile & ": conflict found with " & FoundFile & ", conflict code = " & Hex$(Flags).

Run this sample and look in your Debug or Immediately window. You should see a message similar to the following:

COMDLG32.DLL: conflict found with
c:\windows\system\COMDLG32.DLL, conflict code = 4

This indicates that VersionStamper found an older version of the comdlg32.dll file. Refer to the FileConflict event for more detailed information on the event parameters.

Using the VersionStamper Library Component

We will do a similar example here, but in this case we will read the file list information from a text file. We will also use the VersionStamper Library Component for parsing the file list information and reporting conflicts.

1. Add the "Desaware VersionStamper SW VB# ATL Components" reference to your project. **NOTE**: if you are using Visual Basic, you should select the "VB#" which corresponds to the version of Visual Basic in which you are developing. For more information please refer to the *Which VersionStamper file should I use* section of this manual.

2. For this step, you need to create a file list that VersionStamper can read. You can do this either with the VersionStamper Conflict Wizard, or the VersionStamper control. For our example, add just the comdlg32.dll and comdlg32.ocx files to the output file. Name the output file *test.vsf* and save it in your sample application's directory. Also, for our example, increment the reference *Version Number* of the comdlg32.dll file by 1 so VersionStamper will generate a conflict when verifying this file list.

Next, add the following files to your project: VsVfComp.bas and VsVfComp.frm. These files are located in the VersionStamper "Template" sub directory.

In the "Form_Load" event of your main form, add the following lines of code:

```
FileToCheck = App.Path & "\test.vsf"
Load VerVerifyCompForm

If ConflictFilesFound > 0 Then
   VerVerifyCompForm.Show vbModal
```

```
    Else
        Unload VerVerifyCompForm
    End If
```

Now run the project, a form should appear listing the cmndlg32.dll file as a conflicting file in the top list box. When you highlight that file, additional detail information regarding the conflict will appear in the text box below.

The previous example shows you how to use the VersionStamper Component Library and template files for file verification and conflict reporting. The VersionStamper Script File (.VSF) contains the information required to verify the specified files. The VersionStamper Component Library contains classes to parse the file and perform verification on the specified files. Conflict information is returned and can be used for reporting purposes. You can customize the template files to display the information you want your customers to see.

Emergency Rescue Program

What about situations where components are missing or so incompatible that your application will not even load? For these cases, the following solution is provided. VersionStamper makes it possible for one program to perform a file component verification on another executable. We call this type of program an "emergency rescue program" due to its ability to solve this type of problem.

You can turn both the vrsqtest and vrsqtst2 sample projects located in the Samples directory into stand-alone emergency rescue programs using the following steps:

1. In either project, open the main form for the program (vrsqtest.frm or vrsqtst2.frm).
2. Change the line that sets the **FileToCheck$** variable so that it is set to the name of your application (the one that you wish to "rescue").
3. Compile the project under the name of your choice. Feel free to modify any other parts of the program (form captions, report format, etc.).

That's all there is to it - you now have an emergency rescue program for your application. This application should be placed into the same directory as your primary application. For cases where even these simple Visual Basic applications cannot run, try using our VsRescue.exe utility. VsRescue is an ATL-based rescue program that should run in most situations.

Creating a Component List at Run-Time

You can create a list of components to check during run-time. VersionStamper exposes an ActiveX method that allows you to specify a file for which to search during run-time. This method makes it possible for VersionStamper to verify many combinations of file lists using different warning conditions. Refer to the **vsrtdemo.vbp** project for information on the parameters. The VersionStamper classes also use this method to verify a file list based on a text file. Refer to the VsFileLs or VsWebLst projects for information on the format.

Background Information

This section introduces version resources and provides basic information on how Windows works with components. It also outlines a number of distribution scenarios and strategies.

VersionStamper is designed primarily for Visual Basic developers who are distributing applications or components. If you are authoring your own components, you might also want to refer to chapter 25 of the book "Dan Appleman's Developing ActiveX Components for Visual Basic 6.0: A Guide to the Perplexed" (ISBN 1-56276-576-0) which discusses versioning from the component developer's perspective.

The Version Resource

The version resource is a block of data that can be placed into a Windows executable, DLL, custom control or other file that has an executable header. Starting with version 4.0, Visual Basic allows you to directly add version resource information to your application. VersionStamper includes the dwvstamp.vbx control which can be used to embed a version resource into a Visual Basic 3.0 executable.

A version resource is divided into three parts: the fixed file information, the translation table, and the string file information. The following figure illustrates the contents of a version resource:

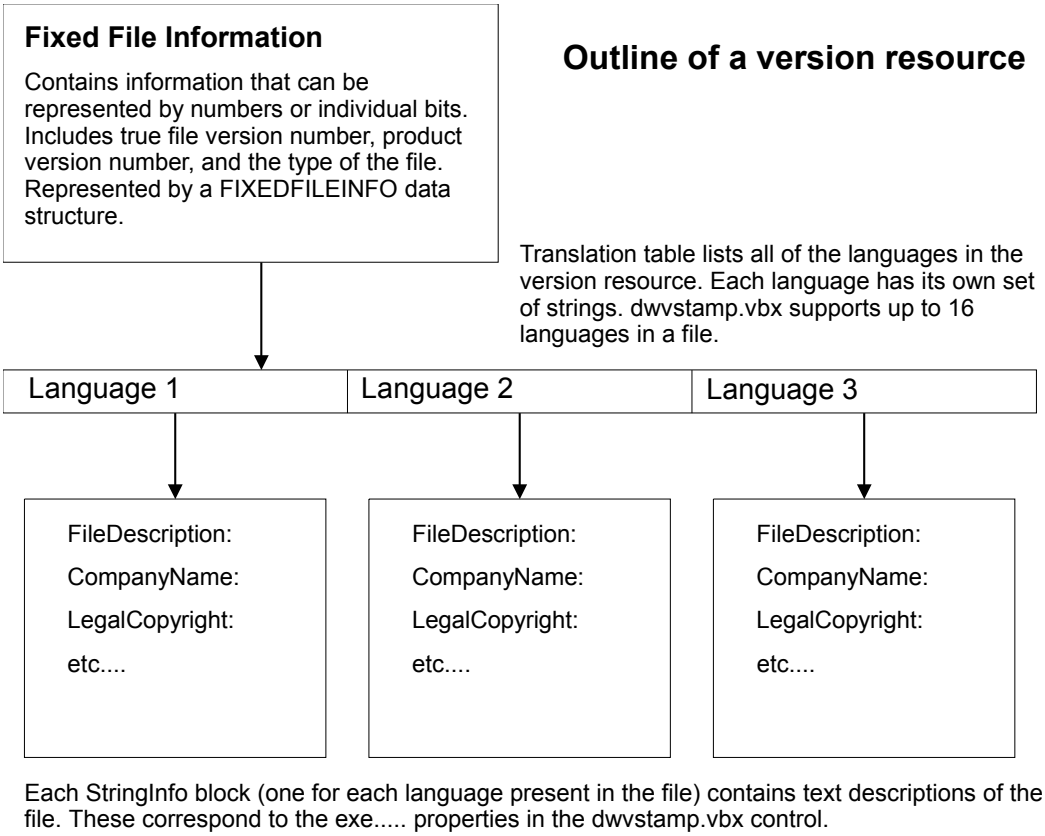

Figure 1
Outline of a Version Resource

Fixed File Information

The FIXEDFILEINFO data structure is present in every file that has a version stamp. It is defined below.

VB Declaration:

```
Type FIXEDFILEINFO ' 52 Bytes
    dwSignature As Long
    dwStrucVersion As Long
    dwFileVersionMS As Long
    dwFileVersionLS As Long
```

```
    dwProductVersionMS As Long
    dwProductVersionLS As Long
    dwFileFlagsMask As Long
    dwFileFlags As Long
    dwFileOS As Long
    dwFileType As Long
    dwFileSubtype As Long
    dwFileDateMS As Long
    dwFileDateLS As Long
End Type
```

The following table describes each of these fields:

Field	Type	Description
DwSignature	Long	Always contains &HFEEF04BD. Automatically set by dwvstamp.vbx.
DwStrucVersion	Long	The version of this structure. Will be greater than 29. Automatically set by dwvstamp.vbx.
DwFileVersionMS	Long	The high 32 bits of the file version number. Set by the dwvstamp.vbx **FileVersion** property.
DwFileVersionLS	Long	The low 32 bits of the file version number. Set by the dwvstamp.vbx **FileVersion** property.
DwProductVersionMS	Long	The high 32 bits of the product version number. Set by the dwvstamp.vbx **ProductVersion** property.
DwProductVersionLS	Long	The low 32 bits of the product version number. Set by the dwvstamp.vbx **ProductVersion** property.

DwFileFlagsMask	Long	Any combination of the constants described in the Version File Flags table that follows. The presence of a flag in this parameter indicates that the value of the **dwFileFlags** parameter for that bit is valid. Refer to the FileFlags table that follows for information on the dwvstamp.vbx properties that set these flags.
dwFileFlags	Long	Any combination of the constants shown in the Version File Flags table. Refer to the FileFlags table that follows for information on the dwvstamp.vbx properties that set these flags.
dwFileOS	Long	One of the constants that define operating system types. This field is set automatically by dwvstamp.vbx.
dwFileType	Long	One of the constants that define file types. dwvstamp.vbx currently marks flags as 16 bit Windows applications.
dwFileSubtype	Long	One of the constants defined in the verinfo.bas file that begin with the VFT2_ prefix. Set to zero by dwvstamp.vbx.
DwFileDateMS	Long	The high 32 bits that specify the date and time of the file's creation. Neither the Microsoft resource compiler nor dwvstamp.vbx set this value.
DwFileDateLS	Long	The low 32 bits that specify the date and time of the file's creation. Neither the Microsoft resource compiler nor dwvstamp.vbx sets this value.

All version comparisons should be based upon the information in the FIXEDFILEINFO structure. For more information, refer to the *Technical Information* section in this manual.

The dwFileFlags field contains flag bits that describe the type of this particular build allowing you to flag an executable as a debug version or indicate other characteristics. These flag bits are described by constants in the VerInfo.bas file and can be set using dwvstamp.vbx properties.

Constants	Description
VS_FF_DEBUG	This file contains debugging information. Set by the dwvstamp.vbx **FlagDebug** property.
VS_FF_INFOINFERRED	The version resource for this file is dynamically allocated and some of the blocks in the resource may be incorrect. Not used by dwvstamp.vbx.
VS_FF_PATCHED	This file has been patched. It may differ from the original file that has the same version number. Set by the dwvstamp.vbx **FlagPatched** property.
VS_FF_PRERELEASE	This is a pre-release version of the file. Set by the dwvstamp.vbx **FlagPre-Release** property.
VS_FF_PRIVATEBUILD	This version of the file is built specially as defined by the PrivateBuild StringFileInfo string. Set by the dwvstamp.vbx **FlagPrivate** property.
VS_FF_SPECIALBUILD	This version of the file is built specially as defined by the SpecialBuild StringFileInfo string. Set by the dwvstamp.vbx **FlagSpecial** property.

Support for Multiple Languages

The FIXEDFILEINFO section of a version resource contains numeric information which obviously does not vary according to the language in use. The final section of a version resource consists of text information that describes this file. This text information will naturally vary depending on the language in use. For example: you may wish to have a file description available in both English and Japanese.

Version resources include text strings describing the characteristics of the file such as the file description, copyright notice and so on. The resource can contain multiple sets of these strings - each in its own language.

The translation table in a version stamp defines the language and code page combinations that are included in the version stamp. It takes the form of an array of integer pairs. The first integer is the language code as listed in the Windows Languages table that follows. The second integer defines the character set or code page to use for that language as shown in the Windows Character Sets table that is located in the *Technical Information* section of this manual.

It is perhaps an indication of how new this feature is, that the language and code page definitions are not followed consistently by every application. For this reason, it is important to look at the translation table if one exists. You need accurate language and code page information to access the StringFileInfo strings that are defined in the next section.

If a translation table is not defined, the most common language/code combinations are &H040904E4, indicating U.S. English and the standard multilingual Windows character set, and &H04090000, which indicates U.S. English and the 7 bit ASCII character set.

For a list of all available character sets and languages, refer to the *Technical Information* section of this manual.

The VerInfo example program illustrates how the translation table for a version resource can be read, and shows how to locate the U.S. English entry in the table.

The dwvstamp.vbx control allows you to specify up to 16 languages character set pairs in an application. Each of these languages may have its own set of StringFileInfo strings.

String Information

The StringFileInfo entries in a version resource are strings that describe certain characteristics of the file. A file may contain unique strings for each language supported, thus the language/code page information is used to access this data as well.

The dwvstamp.vbx control allows you to set any of these strings. It supports up to 16 sets of strings at once, each with its own language.

How Windows and Visual Basic Load Components

In order to take full advantage of VersionStamper, it is helpful to understand a little bit about how Windows loads components at run-time. Once you understand this, you will not only be able to use the VersionStamper control effectively, but you will understand why these problems occur in the first place.

Searching for Components

Every Windows executable file (OCX, VBX, DLL, EXE or others) has a module name. This module name should be unique and will usually match the file name. This module name is also kept in the Windows internal task list to keep track of which programs are running.

For this reason, you should give all of your VB programs unique names. Under 16 bit Windows, attempting to run two programs with the same module name will often cause problems, even general protection faults.

16 Bit Applications

1. When Windows attempts to load a Visual Basic custom control or DLL, it will always confirm first that the module requested is already present in memory. There are NO exceptions to this. If a VBX or DLL of the same name has been already loaded by another application, that VBX or DLL will be used by the newly launched application as well.

    1a. When loading 16 bit OLE controls from your application, Windows always verifies that the control is registered in the Registry - even if the control already exists in memory. If the OLE control is not registered or not found in the specified registered directory, it will fail to load. Refer to the ***Registry is not fool proof*** comment at the end of this section for additional information.

2. The current directory is searched next. The current directory can be set during launch by the program manager, but there is no way to guarantee that it will be the same directory that contains your application.

3. The Windows directory is searched next. This directory can be determined using the **GetWindowsDirectory** API function.

4. The System directory is searched next. This directory can be determined using the **GetSystemDirectory** API function.

    4a. If running under Windows NT, the 32 bit System directory is searched next. Unfortunately, there is no API function that returns the 32 bit System directory when running a 16 bit application.

5. The project directory (the one that contains the executable) is searched next.

6. Finally, the directories specified in your PATH environment are searched.

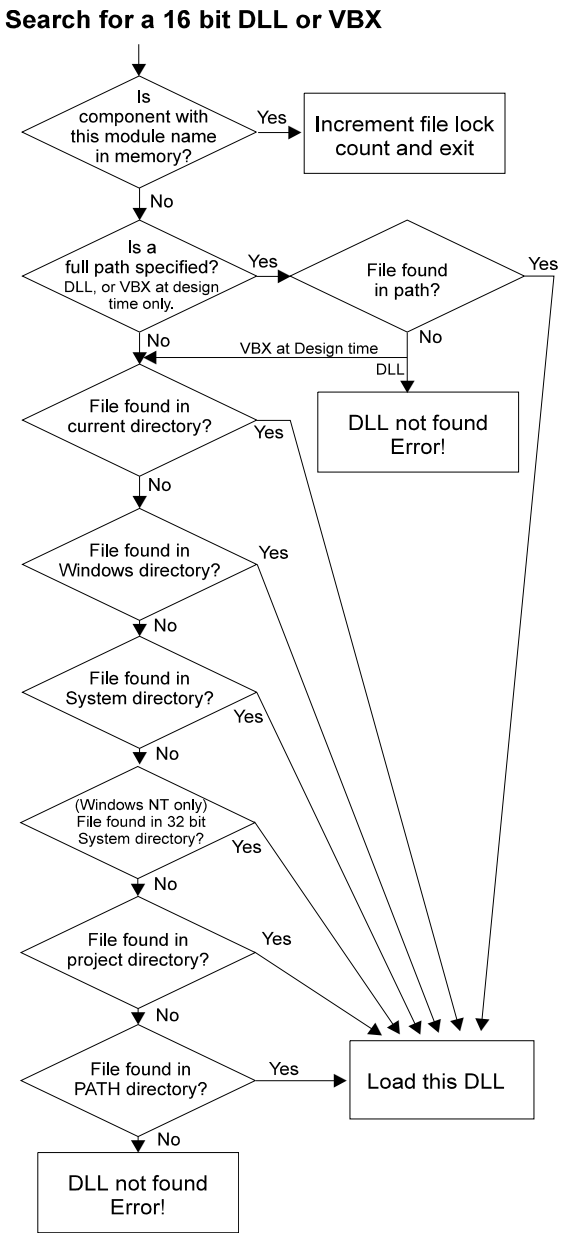

Figure 2
Search Scheme for a 16 bit DLL or VBX 1

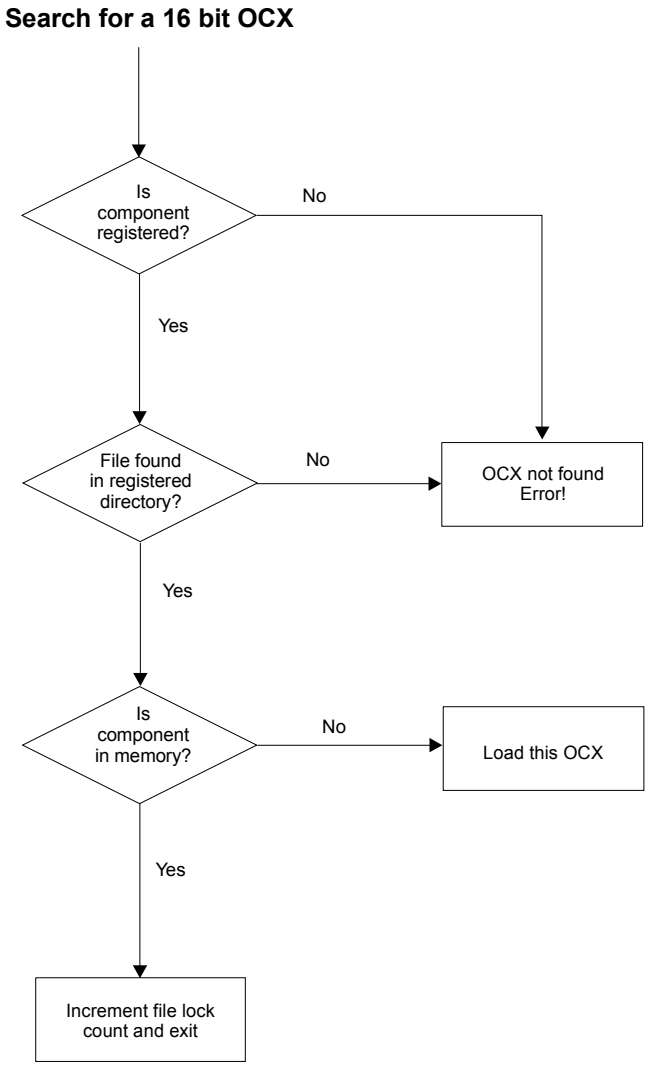

Figure 3
Search Scheme for 16 bit OCX

The project (MAK) file for a project can contain a full or relative path for a Visual Basic custom control. At design-time, this directory will be searched after Windows verifies that the VBX is already loaded into memory, and before any other directory. However, this full or relative path is ignored in the final executable. For this reason, you should always specify a normal search for the embedded file information for custom controls. Since OLE controls are registered, the project will normally contain the OLE control's GUID string. This string is used to search for the OLE control in the Registry. For this reason, you should always specify that the Registry be searched for OLE controls.

Dynamic link libraries follow a nearly identical process with one exception. If a full path is specified for a dynamic link library in a Declare statement, only that path will be checked for the DLL (after the obligatory test in memory). If there is a conflict between declare statements that use the same DLL (for example: one specifies a path and the other doesn't), the first one used when the program runs will take effect (since once the DLL is loaded successfully into memory, it will continue to be found in memory for future calls until it is unloaded). Refer to the *Registry is Not Foolproof* comment at the end of this section for information concerning calling exported functions from OLE controls.

32 Bit Applications

1. When a Windows application attempts to load a component or DLL, it will always confirm first that the module requested is already present in its own process space (refer to the *Beware of Paths* comments for a warning). The exception is when loading OLE controls. When loading OLE controls, Windows always confirms that the control is registered in the Registry - even if the control already exists in the application's process space. You see, OLE controls are actually a special type of DLL, they can export functions that can be called by other applications. Since calling a DLL will load it into the calling application's process space, the same occurs when you call an exported function from an OLE control. But, this method of loading an OLE control will not enable the OLE automation features of the OLE control. Refer to the *Registry is Not Foolproof* comment at the end of this section as to how this situation can occur.

2. If the OLE control is registered in the Registry, Windows will obtain the path of the file from the Registry and load it from there. If the file does not exist in the registered path or is not registered, Windows will display an error message and quit your application, it will NOT attempt to search elsewhere for that file.
3. The application's directory (the directory that contains the executable) is searched next.
4. The current directory is searched next. The current directory can be set during launching by the program manager, but there is no way to guarantee that it will be the same directory that contains your application.
5. The System directory is searched next. This directory can be determined using the **GetSystemDirectory** API function.

   5a. If running under Windows NT, the 16 bit system directory is searched next. Unfortunately, there is no API function that returns the 16 bit System directory when running a 32 bit application.
6. The Windows directory is searched next. This directory can be determined using the **GetWindowsDirectory** API function.
7. The App Paths registry key in the System Registry is searched next for the name of the EXE that is running. This registry key allows you to specify additional paths that Windows will search when loading DLLs and other files for that specific EXE file.
8. The directories specified in your PATH environment are searched.
9. Finally, if you are running in Windows 98, the System32 directory is searched.

Beware of Paths

There is one major difference regarding loading dynamic link libraries in 32 bit Windows. Under Win32, Windows not only compares the module name of DLLs but also the path from which the file was loaded. If you specify a path for a DLL in your declare statement, Windows will attempt to verify whether that particular file has been loaded into your application's process space. Even if the same file is already loaded into your application's process space, Windows will load the DLL again if their paths do not match. This is the case for Visual Basic 4.0, whether this behavior is true on other development platforms is yet to be seen.

In addition, if you specify a path for a DLL in Visual Basic 4.0, you will get an error message if the file is not located in that specific path even if it is located somewhere else in the normal search path.

Registry is not fool-proof

OLE controls are similar to DLLs and can export functions. If you are calling an exported function in an OLE control, then the search order is similar to that of a DLL. This brings up a very interesting situation where the OLE control used may be loaded from the normal search path instead of from the registered directory. If you call an exported API function in the OLE control file before loading the OLE control in your application, then you may be loading the control from the normal search path instead of the Registry.

In 16 bit, when you later load your OLE control, it will still check the Registry, but since it cannot load the same module more than once, it will use the OLE control loaded via the API function call. If your search path happens to find an older version of the OLE control before the current version, then you will be using an older version of the OLE control with your application. Not only that, but you will affect every subsequent application which uses that OLE control since all subsequent applications will just use what's in memory.

The impact is minor in 32 bit environment. The worse that can happen is that you will end up with two different versions of the OLE control mapped into your process space.

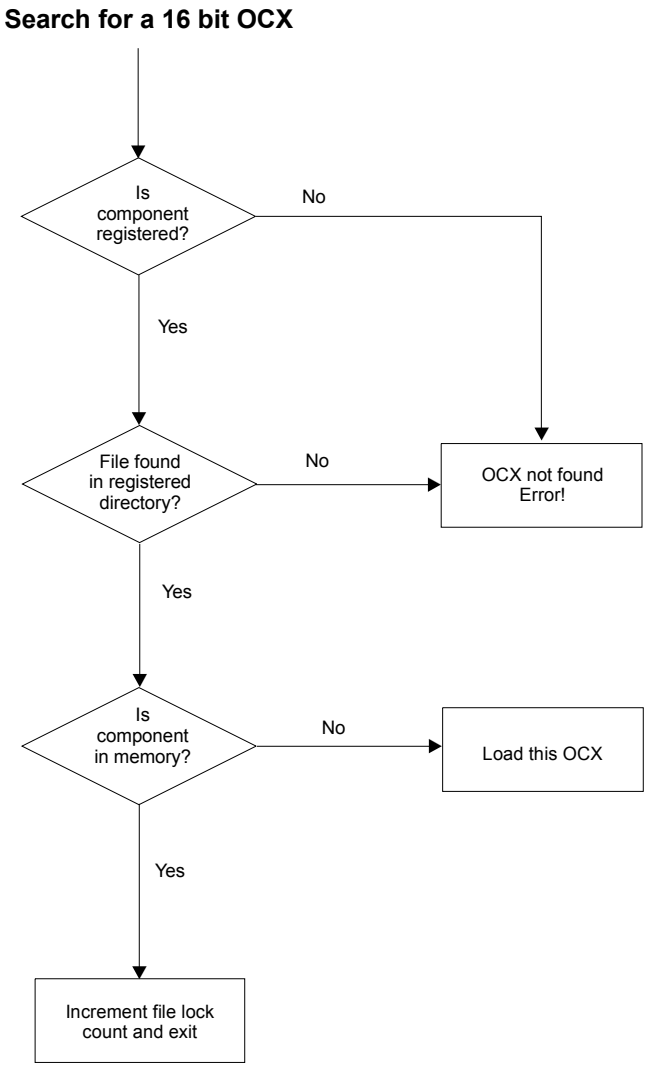

Figure 4
Search Scheme for 32 Bit DLL

Search for a 32 bit OCX

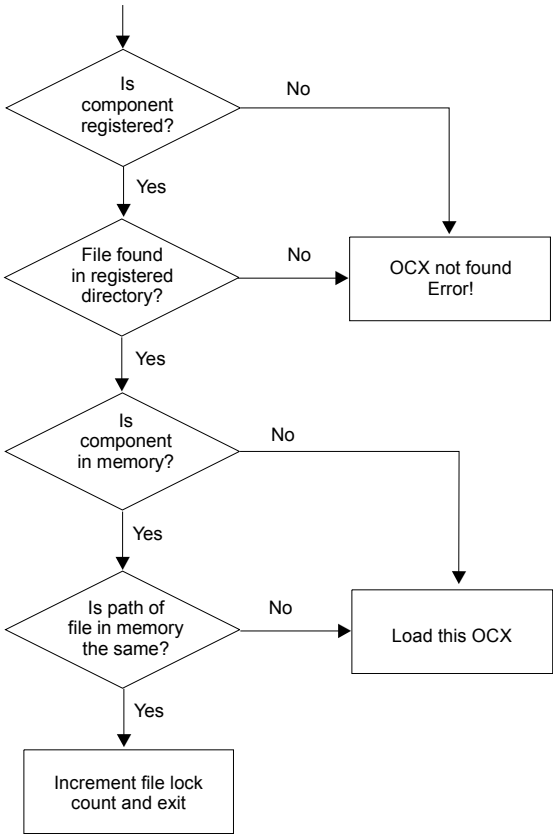

Figure 5
Search Scheme for 32 Bit OCX

Problems with Updating Components

16 bit Windows allows code modules to be shared - in fact, it requires it. This applies to all code modules including executables, dynamic link libraries and custom controls. The advantages of this approach are obvious in terms of memory saving and the saving of disk space. The key disadvantage is part of the reason that VersionStamper exists - once another application loads an incompatible version of a component, any attempt to load that component will use the version that is currently in memory. Under Win32, code modules are mapped into the calling application's process space.

Each code module under 16 bit Windows has a reference count that keeps track of how many applications are using it. When an application closes, it frees the code modules that it is using, decrementing their reference count. When the reference count for a module reaches zero, Windows unloads it from memory. This mechanism poses a serious complication to the problem of updating code modules. Windows keeps track of the location of each code module on disk so that it can load segments from the module as needed. It also locks the code files so that you cannot overwrite them (assuming share is loaded). If you were able to overwrite a code segment that was in use, you would receive a certain General Protection Fault as soon as Windows attempted to load a new segment. Not only that, but overwriting a module would not update the segments of the module that are already in memory.

You may ask: what about the possibility of freeing the module that is currently in use by force, then overwriting the module file with an updated file? This also will not work. Each application maintains pointers to the code modules that it is using, so any attempt to free a module that is in use will lead to a General Protection Fault as well. It is necessary to close all files that are using a component before you update that component.

For this reason, VersionStamper focuses on detection of problems rather than repair. By identifying exactly which files are incompatible, and whether they were detected in memory or on disk, you or your customers will be able to determine the best course for solving the problem based upon the environment in which the program is running.

Under Win32 the nature of the update problem is different, but still exists. Shared components are mapped into the individual process spaces, making it impossible to free a component from another process space. In many cases the operating system prevents you from overwriting a file that is currently in use by another application. In other cases you can overwrite the file, but risk raising an exception in the other application. Fortunately, there are a number of well defined techniques for safely updating components, though in many cases they require you to reload Windows or reboot the system. VersionStamper includes a component update object that can help you implement automatic component updating if you choose to do so.

Typical Distribution Problems: Why They Occur and How to Fix Them

Now that you know how Windows loads components, here are some of the common scenarios that occur that relate to component incompatibility and are often very difficult to track down.

A Component was Incorrectly Registered, or an Incompatible Component was Registered (ActiveX Controls)

ActiveX controls must be registered in the Registry. When loading ActiveX components in Windows NT, Windows 95, and Windows 98, the Registry is searched first. If the component is not found in the Registry, then the application is normally terminated. If the incorrect component is registered, Windows use will that file.

VersionStamper Report:

If the component can be registered, VersionStamper can search the Registry for that component. Additionally, it can also flag a warning if the component is not registered. VersionStamper will report any conflicts based upon the warning conditions that you specified.

Solution:

Review the "other files" and register the latest version of the component in conflict (use Visual Basic's Project – Components...Browse... command to specify the path of the control to register).

A Newer Version of a Component Breaks Existing Code

In the ideal world this would never happen, but in practice it does. What happens at this point is that an updated version of a component is somehow incompatible with the prior version.

VersionStamper Report:

The default VersionStamper test is for older components only. If you are concerned about this type of problem, you can have file conflict warnings triggered for newer components as well - in fact, you can always trigger a warning and have your own verification code decide whether there is a problem or not. The report will once again detail any conflicts based on the warning conditions that you specified.

Solution:

Contact the vendor of the control and report the problem as soon as possible. In the mean time you may wish to re-install the prior version of the control. If a second application requires the newer version of the control, you may have a situation where that application cannot run at the same time as yours (unless your application is 32 bit, or unless you are running in Windows NT and assign your application to run in a separate memory space) until your receive an updated control that supports both.

A subtle, but very effective strategy!

You probably can't anticipate every possible incompatibility problem that may occur on your target system - so how do you set the warning conditions for the file?

Well, there is no reason to limit yourself to a single rescue program. Set the warning conditions for your file and your normal rescue program to detect older versions, but include also a second rescue program that is embedded with the exact list of components used by your primary application - except that the warning conditions are set to always warn for each component. This will produce a complete report of the actual components found on the target system - an invaluable tool for your technical support personnel.

Ability to update warning conditions.

This is a good example on how retrieving a file list to verify from a web site can be very useful. Once you are aware of this problem, you can edit the text file to change the warning conditions for the affected file such that VersionStamper will report a conflict for this particular version.

An Older Version of a Component is in the System Directory (32 bit Windows)

Similar to the problem above. Under Windows NT, Windows 95, and Windows 98, the System directory is searched before the Windows directory. This means that if an older component is present in the System directory, it will be loaded even if the later component is correctly in the Windows directory.

VersionStamper Report:

A file verification with VersionStamper will report a file conflict in this situation. The desired (reference) file and version will be listed as will the file that was actually found. The report will show that the file was found first in the System directory, and the other file's report will indicate that a later version of the component was found in the Windows directory.

Solution:

Move the later version of the control from the Windows directory to the System directory.

Another Application Overwrote the Latest Version of a Component

This is also an extremely common problem. It is caused when an application does not employ an installation program that uses the version resources to determine whether it should update a file (or when a component does not have an embedded version resource). Any application that is installed with a batch file should be automatically suspect.

VersionStamper Report:

A file verification with VersionStamper will detect the older version of the control and report where it was found. The other file's report will indicate that no later version of the control were found.

Solution:

Re-install your application. If you can locate the offending program, complain to the developer and avoid re-installing it. This problem is especially common with lower quality shareware or freeware products.

A VB Executable was Overwritten During a Re-install

This problem occurs naturally with programs that are updated periodically. In an ideal universe, a user would throw out old distribution disks any time an update arrives. In practice, this does not always occur. When the user re-installs the program, it is possible that any Visual Basic executables will be overwritten if they do not contain any version resources which will allow the installation program to check for an existing newer version of the program.

Solution:

For Visual Basic 3.0 executables, use VersionStamper to embed a version resource into your application. Be sure to update the **FileVersion** property with each release (including internal releases).

A Component was Accidentally Deleted

or more likely - an application was distributed without all of its components.

This is one situation that Windows itself can detect. Visual Basic reports on any custom controls that are not found. The error report is not as precise for DLL function calls.

VersionStamper Report:

VersionStamper will report on any components that are not found in its search path.

Solution:

Use VersionStamper's Conflict Wizard or Visual Basic's Distribution utilities to ensure that you include all required components.

The PATH has Changed

This problem is unusual. It only occurs when applications depend upon the PATH to locate components - something that is not common. The majority of applications use the System directory or their own project directory for components. The problem is similar to that of the Windows and Windows System directory - the wrong component can be found.

VersionStamper Report:

A file verification with VersionStamper will report a file conflict in this situation. The desired (reference) file and version will be listed as will the file that was actually found. The report will indicate which files were found first and any other files with the same name that were found in the path.

Solution:

Copy the latest version of the component into your System directory.

An Older Version of a "Dependent" File was Found

For example, some of the ActiveX controls in your application may require a later version of MFC40.DLL (the run-time ActiveX control library), or other DLLs other than those provided with Visual Basic or Windows. Usually, your component's manual will contain information regarding additional support files and the version number of those files which are required by the control. Some of these files may not be detected by the "Scan Project" or Dependency Scan feature of VersionStamper. For best results, use the Process Scan feature of the VersionStamper Conflict Wizard.

VersionStamper Report:

Will not identify this conflict unless the files are included in the file list to verify.

Solution:

Use the VersionStamper Conflict Wizard to retrieve a list of the files required by your application. Be sure to check the "file distribution list" of any components used in your application to make sure all required files are accounted for. Manually add any additional "dependent" files to the file list to be verified.

An Older Version of a Component is in the Windows Directory (16 bit Windows)

In prior versions of 16 bit Windows, the Windows directory was searched before the Windows System directory. This means that if an older component is present in the Windows directory, it will be loaded even if the later component is placed in the correct location in the System directory.

This problem is all too common as some programs continue to install components into the Windows directory (which until VB 2.0 was, in fact, the recommended practice).

VersionStamper Report:

A file verification with VersionStamper will report a file conflict in this situation. The desired (reference) file and version will be listed as will the file that was actually found. The report will indicate that the file was found first in the Windows directory, and the other file's report will indicate that a later version of the component was found in the System directory.

Solution:

Delete the file in the Windows directory. **Exception:** network systems with shared system directories often load components in the Windows directory. In this case, copy the latest version of the component into your Windows directory.

An Older Version of a Component is Present in Memory (16 bit Applications)

This common scenario usually occurs when another application has a private copy of a component that it loads from its project directory. When you run an application that requires the same component, Windows will locate the other application's version of the component already in memory and your application will begin to use it. If it is an older incompatible version, your application may fail to work correctly or even cause a General Protection Fault.

VersionStamper Report:

Assuming your executable was stamped with embedded component information, a file verification with VersionStamper will report a file conflict in this situation. The desired (reference) file and version will be listed as will the file that was actually found. The report will note if the file was found in memory. A list of all other components of the same name found anywhere in the path or Windows and System directory will be produced as well.

Solution:

Review the "other" files list and locate the latest version of the control. Place this latest version in the System directory and delete all other versions of this control on your disk. You will probably have to close other applications using that component in order to perform the copy, and certainly in order for the solution to take effect. The safest approach is to exit Windows, make the changes, and then re-enter Windows.

Strategies for Distributing Component Based Programs

Expensive Strategy: Don't Use Custom Controls

An obvious approach to avoiding component incompatibilities when distributing Visual Basic applications is to avoid using custom controls in general. This approach does, however, involve giving up a large portion of the power and flexibility that Visual Basic derives from the vast array of components that are currently available. It also involves an added cost, as even the most expensive custom controls are far less expensive than developing the same functionality on your own.

Writing your own DLLs and custom controls is also expensive and only minimizes the problem - the distribution problems listed earlier can still occur.

Unreliable Strategy: Place Controls in the Project Directory

Some vendors have taken the approach of placing copies of the components used by their application in their own project directory. This approach has a number of problems:

(16 bit Windows)

1. Should the current directory not match the project directory, these controls will not be loaded if another component of the same name exists in the current directory, Windows directory, System directory or anywhere else in your PATH.
2. This approach does not resolve conflicts which occur when another application loads (from a different directory) one of the components used by your application. If that component is incompatible, problems will occur should the other application run first. These type of incompatibilities are difficult to detect since they depend upon the other application and the sequence in which the programs are loaded.
3. This approach wastes disk space.

(32 bit Windows)

1. If the control can be registered, other installation programs may "re-register" the component in another directory.
2. This approach wastes disk space.

Best Strategy: Use System Directory and VersionStamper

True, we are biased here - but the whole idea of VersionStamper is to take advantage of the component-solution based programming model that is so ideal for Visual Basic. Our recommendation is as follows:

1. Place any components that are unique to your application in your own project directory - no need to clutter up the System directory with components that will never be shared.
2. Place all third party controls, DLL's or other shared components in your system or Windows directory (depending upon whether you are on a stand-alone or networked Windows system).
3. Be sure to rigorously follow correct installation procedures and never overwrite a later version of the control that your customer may have obtained elsewhere.

4. Use only supported controls from reputable companies. This is not just a sales pitch (though we admittedly consider ourselves a reputable company and do support our controls). Components are very complex (and becoming more so as the technology advances - OLE controls, for example, are considerably more sophisticated than VBXs). Despite the extensive testing that component vendors apply to their controls, some bugs will slip through. When a bug due to a new release breaks existing code, you will want to deal with a vendor that can fix the problem and provide you an update quickly. Saving a few bucks in the near term can cost you dearly through the life of your program if the company that sold you the control can't make enough money to survive.

5. Use the VersionStamper control to embed a list of the components used by your program into your executable. Provide an emergency rescue program so that your technical support people will be able to walk the customer through a file verification process to determine if any incompatibilities exist on their target system. Even though we provide the ability to perform a verification on load, frankly we prefer to use a stand-alone program (refer to the VerSpash project for an example) since the verification process does slightly lengthen the loading process.

Using VersionStamper

The following details the primary functionality of the VersionStamper control, each has its own set of properties.

OCX and VBX

Embedded File information: This is the list of components (DLLs, OCXs, VBXs, etc.) that are embedded into the executable with the desired warning conditions. The VersionStamper control can use this information at run-time to locate incompatibilities between the desired component versions and those actually present on the system. Properties exist to scan through an executable and read this embedded information as well.

Conflict Resolution properties: These properties are used to read information about the component incompatibilities that exist at run-time. The actual conflict detection is performed by the VersionStamper control and is triggered under program control.

VBX Edition Only

Version Resource properties: These properties are used to define the version resource for a Visual Basic 3.0 executable. The dwvstamp.vbx control automatically embeds the information from these properties into a standard Windows version resource. This information is available in the VersionStamper online help only.

Embedded File Information

One of the most important features of the VersionStamper custom control is its ability to embed component information into your executable. This information consists of a list of files. For each file you specify a version or file date that typically represents the version used at the time of release. You also specify a set of warning conditions. These warning conditions determine the situations in which the VersionStamper control will trigger run-time incompatibility warnings.

SelectFiles Property

Click on the  button in the property window to bring up the **Select Files** dialog box that allows you to embed the version information for custom controls, DLLs, EXE's and other components into the executable of your Visual Basic application. For the OLE control edition, this brings up VersionStamper's Property Page, select the **Select Files...** button to display the **Select Files** dialog box.

Using the Select Files Dialog

The **Select Files** dialog has four major sections. (**Note**: the **Select Files** dialog box in the VBX and MFC OCX edition is a subset of what is shown.)

The **Available** list box creates a default list of all of the components on your system. These include dynamic link libraries, Visual Basic custom controls, and ActiveX controls (.OCX extension). You can also use the **Directory & File Filter** (*OCX edition only*) button to select additional files to display in the **Available** list box.

Any time you click on an entry in the **Available** or **Selected** list box, the version information for that file will appear (if available) in the **File Details** list box. When selecting an entry in the **Available** list box, the version information will be displayed for the specified file. When selecting an entry in the **Selected** list box, the version information will be displayed for the first file that is actually found on the current system for the specified component name (using a normal Windows search).

Use the **Add** or **Remove** buttons to add controls from the **Available** list to the **Selected** list. You can also double click on entries in the **Available** list to add them to the **Selected** list. You can select multiple entries in the **Available** list, all of which will be added to the **Selected** list. Finally, you can use the **Select Additional Files** button to add files not displayed in the **Available** list box to the **Selected** list box.

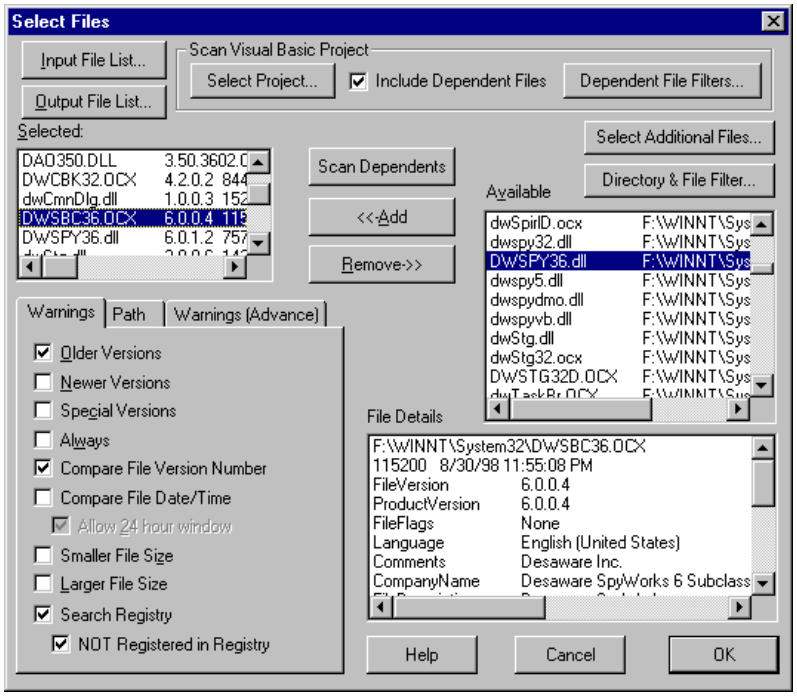

Figure 6
Select Files Dialog Box

The **Selected** list box contains a list of those components that you wish verified for your application at run-time. VersionStamper allows you to embed two lists of components to check (*MFC OCX edition only*) depending upon whether your compiled program will be a 16 or 32 bit executable. Select the list that you want to create (the default select list will be in the same environment as the VersionStamper control).

The target version number, file date, and size are also displayed in the **Selected** list box. You can press the Enter key when an entry is selected, or double click on an entry to display the **Edit File Information** dialog box for the selected entry.

The **Edit File Information** dialog box allows you to change some of the file's information. If a file does not have version information, you can still add a version number in anticipation that it may have version information on the target system. (**Note**: the **Edit File Information** dialog box may not include all of the fields shown below depending on which edition of the VersionStamper control you are using.)

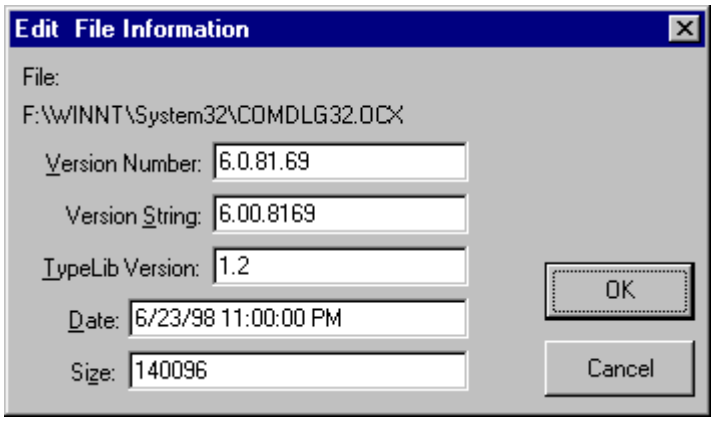

Figure 7
Edit File Information Dialog Box

For the MFC ActiveX edition, the **Display Selected List** frame determines whether the 16 bit or 32 bit file list is displayed in the **Selected** list box. Files added to or removed from the **Selected** list box modifies the 16 bit or 32 bit file list based upon whether the **16 bit** or **32 bit** option button is selected. The **Scan Project** scans for 16 bit or 32 bit DLL files based on whether the **16 bit** or **32 bit** option is selected.

The **Scan Project** command can only scan Visual Basic projects. Be sure your project is saved before using this feature to scan your project. The **Scan Project** button displays a file dialog box which lists Visual Basic project files. Selecting a file in this dialog box will cause that project to be scanned for Visual Basic custom controls and dynamic link libraries referenced in "Declare" statements. These components will be added automatically to the **Selected** list box. For the OCX edition, this command can parse the "#ifdef Win32 Then" directive and add the appropriate DLLs depending upon whether the **16 bit** or **32 bit** option button is selected. The parsing of the #ifdef directive is limited to a single #ifdef level and it cannot recognize constants other than "Win32" or "Win16".

Note on Project files: although Visual Basic project files contain the actual file names of the components used in the "Object=" statement, it actually searches the Registry using just the component's TypeLib identification number and version. This allows the 16 bit version of Visual Basic to open a 32 bit project file, all references to 32 bit components will automatically load their 16 bit equivalent. The same applies to VB 32 opening a VB 16 project. VersionStamper's **Scan Project** feature mimics Visual Basic's behavior, allowing you to create both lists regardless of whether your project was saved as a VB 16 or 32 project. The **Scan Project** feature can also include the DLL, VBX, and OCX dependency files if the **Include Dependents** check box is selected. You can also apply a filter list of dependent files to exclude when scanning a project. This filter list may be edited by selecting the **Dependent File Filters...** command button. Some components may require additional support files that VersionStamper may not detect. These files are usually documented in the component's user manuals and must be manually added to the **Selected** list.

The **Scan Dependents** command button adds a list of the dependency files of a selected file to the **Selected** list box. This is generally used for files that are manually added to the file list since the **Scan Project** feature will automatically perform a recursive scan on all dependent files.

The **Warnings** option button displays a set of warning conditions for the component in the **Selected** list box that is currently highlighted. The default is to warn if an older version of the component is found. This warning can be based upon the version resource (if one exists) or the file date, time, and size - or all of these. You also have the option of triggering a warning if a newer version is present. Checking both newer and older causes a warning to be triggered if the version does not exactly match the one requested.

Checking the **Special** check box causes a warning to be triggered if any of the flags in the FileFlags field of the FIXEDFILEINFO portion of the version resource is set. This allows you to detect Debug, PreRelease, PrivateBuild, Patched and SpecialBuild files.

The **Always** check box causes a warning to always be triggered for the selected file. This is useful if you wish to create your own custom warning conditions. As you will see, the VersionStamper control makes it possible for you to add your own Visual Basic code to the verification process and to customize it to suit yourself.

The ATL edition includes additional warnings such as warn if the found file's **TypeLib** version string is older or newer than that specified. The **TypeLib** version only supports major and minor version numbers, for example 2.1. Also supported in the ATL edition is warn if the found file's **FileVersion string** is older or newer than that specified. Note that the **FileVersion string** will be converted into a number before comparing, thus it will be a numerical comparison rather than a string comparison. Also note that some DLLs **FileVersion string** resource includes additional text that are not related to version information.

When a file is added to the **Selected** list box, it is scanned to determine whether it contains a type library. If so, the **Search Registry** and warn if **Not Registered in Registry** check boxes are automatically selected. Some files may be registered in the Registry even though they do not have a type library, but VersionStamper will not find them in the Registry because it only searches the TypeLib branch so it is best to leave the "warn if **Not Registered in Registry"** check box unchecked.

The **Path** option button allows you to select where the VersionStamper control will search for the control at runtime. The default is a normal Windows search.

The MFC edition of VersionStamper supports two additional path overrides. A full path search causes the VersionStamper control to only search in the exact directory in which it was originally found. A relative path search causes the VersionStamper control to only search in the directory in which it was originally found relative to the current directory. In practice you will almost always want to perform a normal search. The only exception is where you have specified a full path in the Declare statement in your VB project (a practice which is discouraged), or a particular DLL has to be located in a specific directory.

The ATL edition of VersionStamper supports additional path overrides, most of which are self-explanatory. The *Relative to UserDefined directory* mode will search for a file relative to a user defined directory as specified in the VersionStamper control's **UserDirectory** property. This is used for situations where files are installed in a private directory or a directory that the end user can specify during installation. This allows your primary application to retrieve the directory before running the verification. The *Relative to Reference File's directory* mode will search for a file relative to the specified reference file's directory. This will be mainly used for

dependents of registered files that are installed in private directories. The specified "Reference File" must also be in the list of files searched. An example of this are the ADO DLLs which are installed in the \Program Files\Common Files\System\ADO directory. Some of the ADO files are self registering, thus VersionStamper would find them. But dependents of these registered files are not registered and VersionStamper would not find them under the default search.

The **OK** button updates the VersionStamper control according to the changes you've made. The **Cancel** button discards all changes.

NOTE: File lists embedded using the MFC edition (dwvstp32.ocx) control cannot be read by the ATL edition (dwvstp36.ocx) control and vice versa.

The Art of Enumeration

With VersionStamper we faced the challenging task of trying to find a solution to the problem of version conflicts that was easy enough for anyone to use, yet flexible enough to satisfy even expert programmers. The approach we decided upon takes advantage of the event driven nature of Visual Basic and uses a technique called enumeration.

When you trigger the verification operation using either the VerifyMode or VerifyFile operation, the VersionStamper control first searches for the list of embedded files and warning conditions that are embedded into the program. The **VerifyMode** property is used to verify the component list for a particular VersionStamper control. The **VerifyFile** property is used to verify the component list for an executable file on disk.

The VersionStamper control examines each file in the component list that it is given. The control then checks for the presence of that file on disk and checks for the warning conditions specified in the list. If any of the warning conditions occur, a **FileConfict** event is triggered. During this event, there are many properties that can be accessed to obtain detailed information concerning the incompatibility that was detected. Your Visual Basic code can be written in such a way that it can determine how to respond to each file conflict. It can ignore the incompatibility, report it, add it to a list for later reporting, or even write code that attempts to fix the problem.

The following chart illustrates this enumeration process:

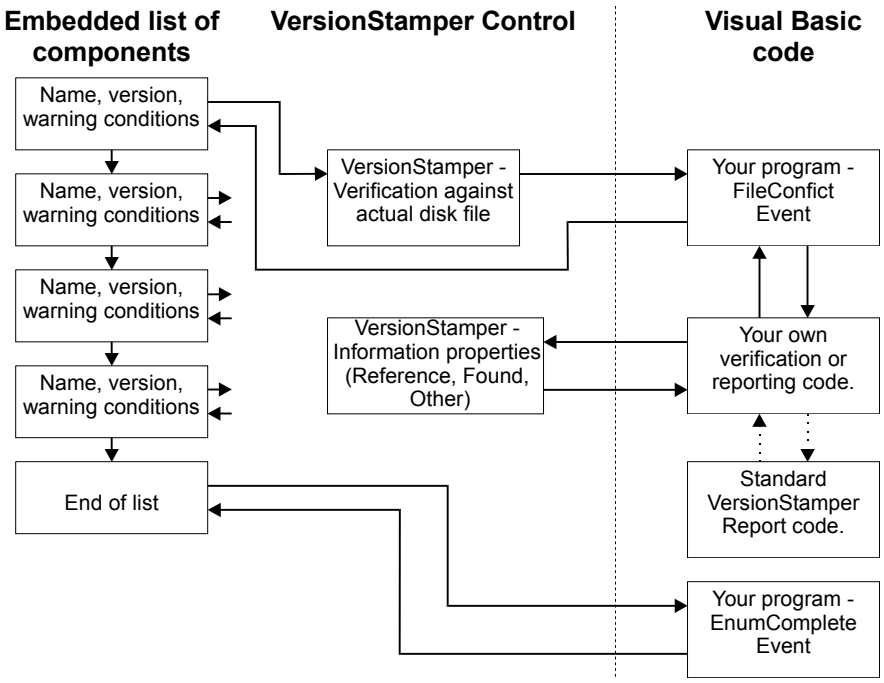

Figure 8
VersionStamper Enumeration Process

This enumeration technique is also employed to obtain a list of the components embedded into the file. In this case the **FileScan** event is triggered for each component. During this event it is possible to obtain information about the file, version and warning conditions for the component.

The **EnumComplete** event is triggered after all of the embedded files have been scanned or verified. It is very possible for this to be the only event triggered during a verify if no conflict situations exist, or in the case of a scan of a file that contains no embedded component information.

Scanning and Verification Events

The following events are triggered during the course of a verification or scan of the embedded component list of a file.

FileConflict Event

FileConflict(*ReferenceFile* As String, *FoundFile* As String, *Flags* As Long, *StopVerify* As Boolean)

VersionStamper allows you to embed version information and warning conditions for components into your Visual Basic executable. During file verification, the VersionStamper control scans this embedded information and searches the system for the first file it encounters that matches the component name. It compares the version and date information found against the warning conditions included in the file. If any of the warning conditions are detected, a **FileConflict** event is triggered with the following parameters:

Parameter	Description
ReferenceFile	The embedded file name. This will include a full or relative path only if specified during the embedding process.
FoundFile	The full path of the file that was found that matches the embedded file. The empty string if no file was found.
Flags	A Long that determines which warnings were triggered. It can be a combination of the following: Bit 0 Older file was found (Date compare). Bit 1 Newer file was found (Date compare). Bit 2 Older version found (Version compare). Bit 3 Newer version found (Version compare). Bit 4 Special version found. Bit 5 Always warn was set. Bit 6 File was found in memory. Bit 7 No matching file was found. **NOTE**: If a path override was specified, then this flag will be set if the file was not found in the specified override directory even if the file was found elsewhere.

	Bit 8 A version compare was requested but no version resource was found in file FoundFile. Bit 9 A larger file was found. Bit 10 A smaller file was found. Bit 11 File was not registered in the system registry. Bit 12 Older file was found (TypeLib version compare). Bit 13 Newer file was found (TypeLib version compare). Bit 14 No typelib resource was found in the file. Bit 15 Older file was found (File Version String compare). Bit 16 Newer file was found (File Version String compare).
StopVerify	Set this parameter to True to stop the verification process.

File verification is initiated using the **VerifyMode** or **VerifyFile** properties, and the **VerifyObjectFile** method.

You can also use the CF_ constants as masks to determine the value of a particular bit. These constants are defined in the file vervrfy.bas.

FileScan Event

FileScan(*ReferenceFile* As String, *VerifyFlags* As Long, *PathFlags* As Long, *StopScan* As Boolean)

VersionStamper allows you to embed version information and warning conditions for components into your Visual Basic executable. During file scanning, the VersionStamper control scans this embedded information and triggers a **FileScan** event for each file found using the following parameters:

Parameter	Description
ReferenceFile	The embedded file name. This will include a full or relative path only if specified during the embedding process.

VerifyFlags	A Long that determines the warning conditions for this file. It can be a combination of the following: Bit 0 Date & Time is specified. Bit 1 Version number is specified. Bit 2 String Version is specified. Bit 3 Use relative path (no search). Bit 4 File Size is specified. Bit 5 TypeLib is specified. Bit 6 Search in the registry for this file. Bit 16 Trigger warning on older file. Bit 17 Trigger warning on newer file. Bit 18 Trigger warning if special flags set. Bit 19 Base warning on version comparison (bits 16 and 17). Bit 20 Base warning on date comparison (bits 16 and 17). Bit 21 Always trigger warning for this file. Bit 22 Trigger warning on larger file size. Bit 23 Trigger warning on smaller file size. Bit 24 Trigger warning if file is not registered in system registry. Bit 25 Trigger warning on older file version string. Bit 26 Trigger warning on newer file version string. Bit 27 Trigger warning on older file TypeLib version. Bit 28 Trigger warning on newer file TypeLib version. Bit 30 No 24-hour leeway on date comparison.
PathFlags	A Long value that specifies special path overrides used to search for this file. It can be the following bits: Bit 1 Search in the same directory as a specified “Reference File”. The Reference File name is specified in the ReferenceFileName property. Bit 2 Search in the specified full path.

	Bit 3 Path is searched relative to one of the following directories as specified by Bits 4-11: Bit 4 Current Directory Bit 5 Windows Directory Bit 6 System Directory Bit 7 User Defined Directory Bit 8 Application Directory Bit 10 Program Files Directory Bit 11 Common Files Directory
StopScan	Set this Boolean parameter to True to stop scanning the rest of the file.

File verification is started using the **VerifyMode** or the **ScanFile** properties.

You can also use the VF_ constants as masks to determine the value of a particular bit. These constants are defined in the file verenum.bas.

EnumComplete Event

This event is triggered after file scanning or verification to indicate that the operation is complete. This event has no parameters.

Starting a Verify or Scan Operation

Three properties are used to initiate a file verify or scan operation.

VerifyMode Property

This property is used to control verification of the component information embedded into your Visual Basic application. Each of the embedded components is verified against the components specified in the VersionStamper control for which the property is set.

There are four possible values for this property:

Value	Description
0	None - Verification is not triggered.
1	On Load - Verification is triggered as soon as the form containing the VersionStamper custom control is completely loaded and events are "unfrozened".
2	Manual Verify - Setting the **VerifyMode** property to this value causes an immediate component verification

	operation to take place. Any files on the system that trigger a warning condition according to the embedded component list will cause a **FileConflict** event to be triggered.
3	Manual Scan - Setting the **VerifyMode** property to this value causes the list of embedded components to be enumerated by the **FileScan** event for this control.

The only values that are valid at design-time are 0 and 1. Values 2 and 3 can be set at run-time under program control to trigger the corresponding operation. The property's value is automatically set back to zero upon completion, so this operation can be repeated as often as desired. You cannot perform another verify or scan with the same VersionStamper control if one is already in progress. **NOTE**: Do not attempt to set this property to 1 (vstOnLoad) at run time. Doing so may trigger verification at an unpredictable time.

A subtle technique: Unlike the **ScanFile** and **VerifyFile** properties that work directly upon an executable file, the **VerifyMode** property works with the list that is internal to the control. If you have multiple VersionStamper controls in a project (refer to the **Slave** property description later in this chapter), you can have multiple component lists in your project and have each VersionStamper control verify against a different set of files or warning conditions.

VerifyFile Property

The **VerifyMode** property can be used to control verification of the application containing the VersionStamper control. During this verification process, a **FileConflict** event is triggered for each conflict that is detected between the embedded component information and the actual components located on the target system.

The **VerifyFile** property allows you to perform this verification process for any file that contains embedded information. This is frequently used by rescue programs such as verresq.exe to allow you to detect conflicts for files that cannot even load.

To start a verification, simply set this property to the name of the file. You cannot perform another verify or scan with the same VersionStamper control if one is already in progress.

ScanFile Property

The **VerifyMode** property can be used to scan the current application and obtain information about the components and warning conditions that are embedded in the file. During this scanning process, a **FileScan** event is triggered for each file that contains embedded component information.

The **ScanFile** property allows you to perform this scanning process for any file that contains embedded information. This is used by the VerInfo program to obtain information on other files.

To initiate a scan, simply set this property to the name of the file. You cannot perform another verify or scan with the same VersionStamper control if one is already in progress.

FileCurrentDir Property

This property can be used to override the current directory during file verification triggered by the **VerifyFile** or **VerifyMode** properties. Normally, during file verification, the current directory is searched before the Windows directory (after searching for modules resident in memory or registry). This property allows you to specify a different current directory to search.

This property is a string, and is accessible only at run-time.

Delaying a Verify or Scan Operation

You can delay a verification or scan by setting VersionStamper's **Delay** property to True. Normally, the verification or scan is performed immediately when VersionStamper is invoked (at load, or when setting one of the properties or methods) and control is not returned until it is completed (after the **EnumComplete** event is fired). However, there are some ActiveX containers (e.g. Visual FoxPro) which prevent events from firing while loading or invoking a property or method of an ActiveX control. In this case, set the **Delay** property to True. When a verify is requested, VersionStamper will return right away and post a message to itself telling itself to perform the verification as soon as possible. In this case, watch for the **EnumComplete** event to occur (set a global flag there), this will tell you that the verification or scan is completed. See the example in VSRTDemo for a warning on using this with the **VerifyObjectFile** method.

Properties Available During Scanning and Verification

Properties with the prefixes “Found”, “Other” and “Ref” are accessible only during the **FileConflict** and **FileScan** events. These properties are generally used to obtain information about the files and conditions that triggered the event. Refer to the Help file for detailed information regarding each property.

Using Multiple VersionStamper Controls in an Application

Slave Property

This property controls the behavior of VersionStamper controls when there is more than one instance of a VersionStamper control in an application.

The nature of the VersionStamper control is such that you will rarely need to have more than one of these controls in your application. After all, each application can contain two embedded file lists.

However, there are cases where you may wish to use the capability provided by the control without saving a version resource or embedded information for that control. This is demonstrated in the VerInfo project, where each MDI child form contains a VersionStamper control which is used to scan for embedded version information for a file. This also occurs when you want to keep multiple component lists in a program. (You may also refer to the VerifyObjectFile method for another method to use multiple component lists for a single program.)

In order to prevent conflicts due to multiple instances of the VersionStamper control in an application, all controls created or loaded (after the first one) in the Visual Basic design environment are automatically marked as slave controls (and the value of the **Slave** property for that control is set to True).

Version information for slave controls is not stamped into the executable (applies to VBX edition only), and while you may specify an embedded file list for a slave control, that list will not be detected by an external scan of the file. This list can only be verified internally by an application using the **VerifyMode** property.

Warning! If you create two VersionStamper controls in an application, and delete the one that is not a slave, the other control will continue to be a slave, thus version information from that control will not be stamped into the file. To solve this problem, simply set the **Slave** property to False.

VersionStamper Methods (ActiveX Edition Only)

VerifyObjectFile Method

VerifyObjectFile *(*filename$, guid$, warningflags&, reserved%, verMS&, verLS&, filesize&, year%, month%, day%, hour%, minute%, second%)

VerifyObjectFile2 (filename$, guid$, warningflags&, pathflags&, verMS&, verLS&, versionstring$, referencefile$, typelibver&, filesize&, year%, month%, day%, hour%, minute%, second%)

The VerifyObjectFile methods allow you to specify a reference file to verify during run-time. This method is useful for applications that may enable/disable certain portions of the reference file depending on the user's system configuration or license. You can create a list of the required files from a database or another file type based on a particular configuration of your application dynamically rather than try to embed all the possible combinations into many VersionStamper controls. Invoking this method will trigger the normal VersionStamper events. Refer to the vsrtdemo.vbp for an example.

Another use is to create the required file list on your web site, and have VersionStamper dynamically retrieve the list each time and perform a scan on the target computer. You can use the VersionStamper Conflict Wizard to create a list of required files for your application and output the file list information into a text file that can be parsed by VersionStamper. This allows you to update the file list and warning flags when necessary (such as if a newer version of a particular file breaks backwards compatibility or if a newer version of a particular file requires a newer version of a dependent file). This also opens the possibility of using VersionStamper on the internet or intranet.

The VersionStamper components include many classes to support integrating VersionStamper with the internet or intranet. Other examples are: sending an email message to you (or the System Administrator) whenever VersionStamper detects a file conflict on a particular computer or automatically updating a computer with an older version of a particular file. Refer to the VersionStamper help file for details on using these methods.

VersionStamper - API Functions

VersionStamper exports several API functions that can be called directly from your Visual Basic application. Declarations for these functions can be found in file verapi.bas and VsObAPI.bas. In order to save space, the functions have a generic declaration except in cases where the parameters are different. The library name from the declarations is omitted. The following library name should be used as required: VBX - dwvstamp.vbx, 16 bit OCX - dwvstp16.ocx, 32 bit OCX - dwvstp32.ocx, dwvstp36.ocx, or dwvsob36.dll. Refer to the VersionStamper Help file for more information on these functions.

VersionStamper Conflict Wizard

Wizard Functionality Overview

The VersionStamper Conflict Wizard is used to generate a list of files used by your application. This list can then be exported to create a VersionStamper Script File (VSF) for use in detecting file conflicts. The VersionStamper ActiveX controls can also import a VSF file list, but some of the more advanced features such as custom messages and file update will be ignored. The first step in using the Wizard is choosing whether you want to generate a new VSF file or open an existing one to edit. When creating a new VSF file, you may choose the quick manual method of selecting just the files you want, or use one of the Wizard's comprehensive scanning methods which does some additional tests to retrieve dependent files.

The Wizard can scan any EXE, DLL, OCX or other compatible binary files for direct dependents used by these files. The Wizard obtains this information from the import table of the file to be scanned. This provides quick and reliable information as to what other libraries this particular file may require. However, the Wizard cannot find dependents for any file that does not have an import table, and thus cannot scan databases and such for dependents. The only exception to the above rule is the ability to scan Visual Basic Project files (*.vbp), Wise Installation Script files and InstallShield Installation Script files. The Wizard scans the Visual Basic project file itself and all the code of the project's forms and modules and determines, if at all possible, which components and libraries are needed by this project. The Wizard can also scan the Wise Installation and InstallShield script files to retrieve the files that are distributed with your application.

Also, the Wizard is able to generate a list of currently loaded modules for any process currently running on your system. The Wizard (upon your selection) attaches to the given process (or starts a new application) and monitors any modules that the application loads into memory. It then builds a list of these modules which is also used to create a VersionStamper Script File.

The Wizard also provides a rather comprehensive editing mechanism for VersionStamper Scripts – it has the ability to modify scripts by the process outlined above, or it can load existing scripts from a disk. The system allows you to modify all information that is stored in the script, including almost all version and date information, as well as setting various flags which VersionStamper uses in client programs to fix version conflicts. This stage of the process also allows you to add non-executable files (such as README.TXT and data files) which do not have version information or import tables but which may nonetheless be useful to your application.

Finally, the Wizard supports a command line batch mode. That is, you can run the Wizard as part of a batch process to either create a list of files for your application, or update an existing file list using the files on the build machine as reference.

The purpose of this Wizard is to make creation of VersionStamper Script Files (*.vsf) easy and quick. The Wizard attempts to incorporate almost every possibility for VersionStamper Script File options, but it also does a good deal amount of error checking; this may cause the Wizard to disallow some strange and unusual combinations of options, even though this combination may not be completely erroneous. In these cases, it is always possible to edit the Script File directly using any standard text editor (i.e. Notepad).

Selecting Scanning Type

Although there is no magic solution for determining all the required files that a particular application may use, VersionStamper offers several scanning methods that should give a fairly accurate picture of which files are required by a particular application. Each of these methods uses a different technique to determine the required files, a combination of these methods will give the best results.

Visual Basic Projects

This method scans a Visual Basic project to retrieve files directly referenced by the project. All referenced files are then recursively scanned to compile a list of all files used by the project. This method may not detect all dependent files, especially those that are NOT directly referenced or dynamically loaded which is the case with Visual Basic ActiveX components and controls.

Standalone files

This method scans a single binary file (DLL, EXE, OCX, etc) and retrieves the direct file references of the file. All direct file references are then recursively scanned to compile a list of all files. This method will not detect files that are NOT directly referenced or dynamically loaded which is the case with Visual Basic compiled files. However, it should work on almost all other Windows applications, including a majority of those written in C++.

Currently Running Process

This method scans a process and retrieves the files currently mapped into the process space. The Wizard can monitor a running process for an indefinite period. When using this method, keep in mind that not all required files are mapped into an application right away. ActiveX components and controls and other files are loaded when required. To obtain the most accurate file list, this method requires that you run your application and select as many options as possible, and open as many forms as possible to try to get all required files to be loaded by your process.

Installation Script File

This method scans an Installation Script File to retrieve files directly installed by the script. Currently, this feature supports Wise Installation, InstallShield Pro and InstallShield Express script files. This method also allows you to apply filters to omit certain types of files or specific files.

Selecting Script Source

The VersionStamper Conflict Wizard provides several options for selecting for source of the file list. The first step provides you with three choices: Scan New Script, Quick Script which assists you in manually adding files and lastly the option to load and edit an existing VersionStamper script.

The two latter choices are available under the Other Script Options selection.

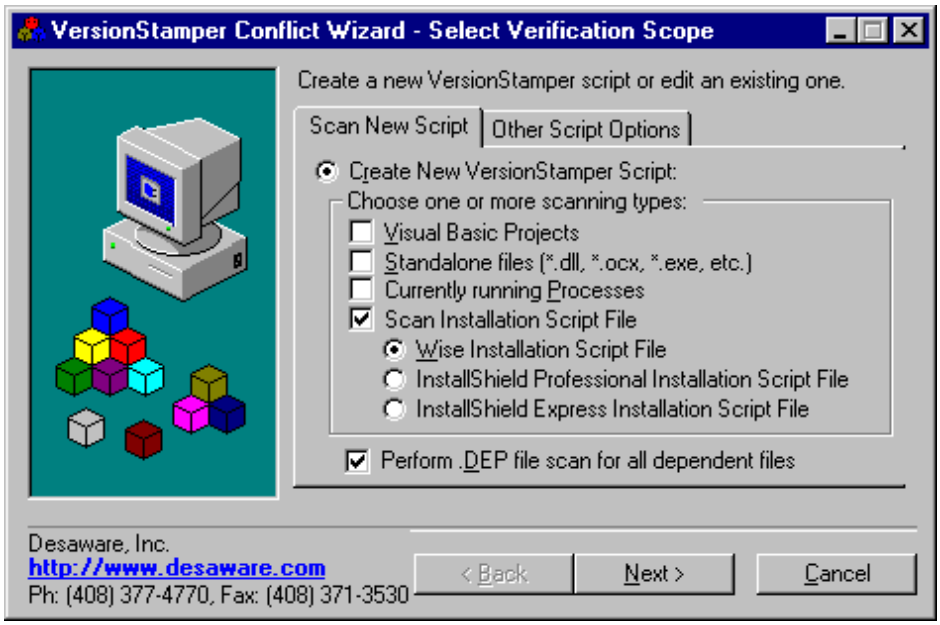

Figure 8
Creating A New VersionStamper Script

Create new VersionStamper Script via Scanning

This is the most commonly used and most complex option for the Wizard. If this option is selected, the Wizard guides you through several steps of selecting executables, system components, Visual Basic project files, installation script files, and processes. After these items are selected, the Wizard scans and then determines (if possible) all of their dependencies. See *Selecting Wizard Scanning Types and Scanning Processes* for more information. In this step, the Script is created from scratch (no previous information is used); however, you are given the chance to edit any and all script information after the scan is completed.

The ***Other Script Options*** *Pane* displays the following:

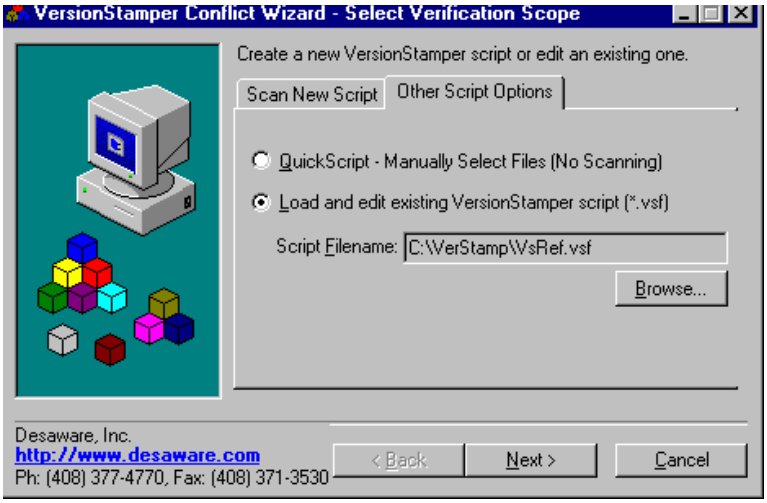

Figure 9
Other Script Options Pane

QuickScript - Manually Add Files to List

This option allows you to start from scratch (empty initial file list). Using the Editing File List Window you can create a very small and a quick Script File -- probably only a couple entries. This option does not constrain you to a limited set of verification and version options -- you may use all the advanced settings. However, this option does no automated dependency scans and only adds files to the list as you manually select them.

Load Existing VersionStamper Script File

This option allows you to edit an existing VersionStamper Script to add new files, delete files, or change version/verification options. Like the QuickScript option, this option does no automated dependency scans; at this point, you cannot use automated scans to augment an existing script. After the initial file information is loaded from the script file, this option behaves identically to the option above.

Scanning Installation Script File

This step of the VersionStamper Conflict Wizard attempts to build a dependency file list by scanning an installation script. It is important to note that the installation script scanning algorithm was based on our own testing of script samples created in-house and that we do not have access to or knowledge of the actual installation script file formats for the installation programs we support. We will make every effort to create as complete as possible an installation scanning implementation.

Limitations on Scanning InstallShield Express installation Scripts

The VersionStamper Conflict Wizard requires that you first successfully build the InstallShield Express setup before it can scan it. Furthermore, the VersionStamper Conflict Wizard does not scan the actual installation script file (the format of the script file is not public knowledge), but the installation's "Report" build file. The Report file is in HTML format and contains a summary of the files included with the installation in addition to the source paths. However, each installation script can be distributed on more than one type of media. As a result, each media type has its own corresponding Report file. Finally, Merge Modules included with the installation are not included in the generated file list.

We currently support Wise Installation script files 6.0 through 8.1, InstallShield Pro script files and InstallShield Express script files.

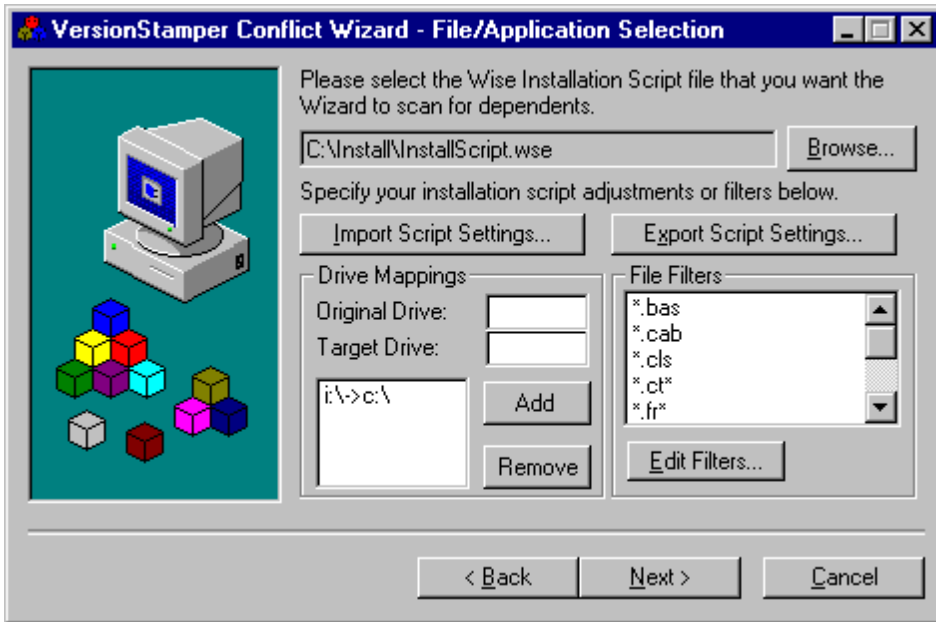

Figure 10
File Application Selection

To scan an installation script file, select the installation script file by using the Browse button to select the file. When using the VersionStamper Conflict Wizard to scan an InstallShield Express installation script, select the script file's Report file rather than the script file itself. The InstallShield Express Report build file can be found in your InstallShield Express script file's \Express\<Media Type>\Reports subdirectory where <Media Type> is the media type you selected for the build (e.g. SingleImage, CD_ROM, etc).

The Drive Mappings section allows you to specify a alternative drive letter while scanning the installation script file. This is useful for cases where the script file is located on a different system than where you are building the dependency file list.

For example, if the script file is located on another system and references files on the C drive of that system the script scanning algorithm may not find those files on the current computer's C drive. You can map a drive to the reference files' C drive, let's say F, then add a drive mapping to the script where C is the original drive and F is the target drive. After selecting the Add button, you should see c:\->f:\ in the list box.

This tells the script file scanning algorithm to replace all references of the C drive to your F drive (which is mapped to the reference files C drive). You may assign more than one drive mapping for each installation script file.

The File Filters section allows you to exclude certain files that the script scanning algorithm returned. The Edit Filters button displays a form that allows you to add or remove filters from the list. You may use the standard Visual Basic string wildcards in your filters. For example, to exclude Visual Basic Class Module files, add "*.cls" to the filter list, to filter out files that starts with "readme", add "readme*.*" to the list.

You may also import or export the drive mappings and file filter settings from and to an external file by selecting either the Import Script Settings or Export Script Settings buttons.

Scanning Currently Running Processes

This step of the VersionStamper Conflict Wizard attempts to build a dependency file list for an application by monitoring the modules (ActiveX Controls, DLL's, etc.) that it may load into memory. It is important to note that the application may not necessarily pre-load all of its controls; it may delay bringing them into memory until they are necessary. Thus, it is the user's responsibility to attempt to coerce the application into loading as many modules as possible. Usually, to accomplish this, one must open/close files, attempt to access as many different features of the application as possible, and (perhaps) modify application settings.

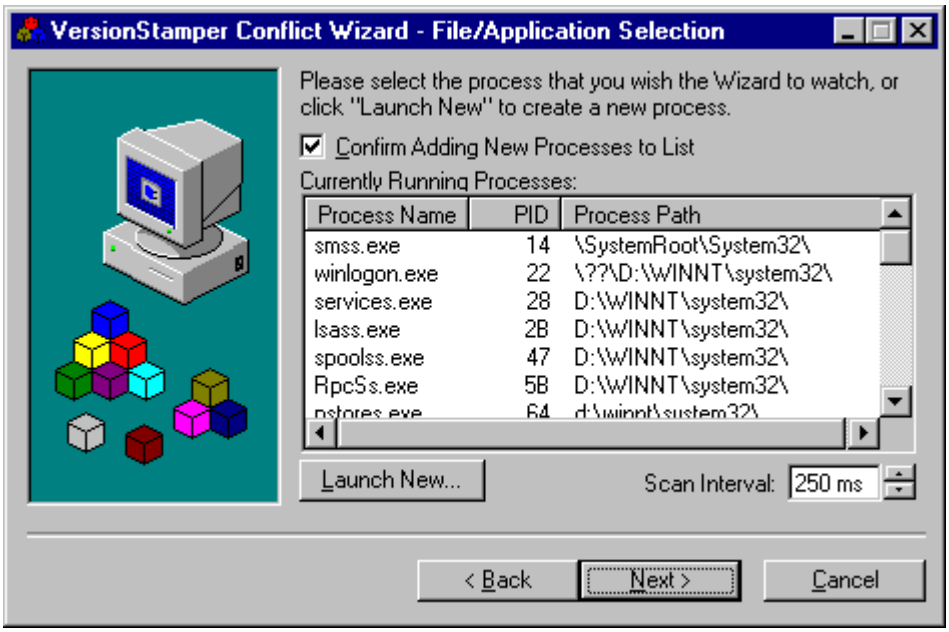

Figure 11
Scanning Processes

In order to scan an application, all the current processes on the system are loaded into the Wizard's Process List box. The user can then choose a process from the list (which is updated continuously), or the user can choose to launch a new process directly by clicking the "Launch New…" button and selecting an executable program. The "Launch" method is suggested for most applications because it allows the Wizard to monitor the application's progress from the very beginning, and may encounter modules that are loaded at startup and then freed.

If a new process is launched, the Wizard automatically moves on to the "Scanning" step; otherwise, the user must click the "Next >" button to begin the process scan.

Once the Wizard is in scan mode, it continuously updates its list of modules loaded by the target application; this may cause a certain reduction in the speed of your system. The user may increase the time interval between scans and thereby decrease the reduction in performance by editing the numerical box below the Process List. The default setting is 250 ms as shown in the Scan Interval box in the Figure 11.

The scan ends when the user exits the Wizard step by clicking either of the "< Back" or "Next >" button, and, at that point, the file list is saved for future use and dependency scanning.

If a new process appears on your system while a scan is in progress, the Wizard produces a message box to query the user if this new process should be added to the scan. This is in case the application being scanned starts an ActiveX EXE server or another executable program on your system. If the process that appears is not related to the application in question, the user should click "No" and the Wizard will remove that process from the scanning list.

Unlike the other scanning options (Visual Basic Project, Executable, and System files), this option does not perform any other dependency scans on the files found by monitoring a process (i.e. no import tables or function references are checked inside the files found). If you want a dependency scan to be performed, you may run the Scan Process feature, note the name of an EXE or DLL the Wizard found, and then manually add it to the Scan Executable step of the Wizard.

Editing and Filtering Files

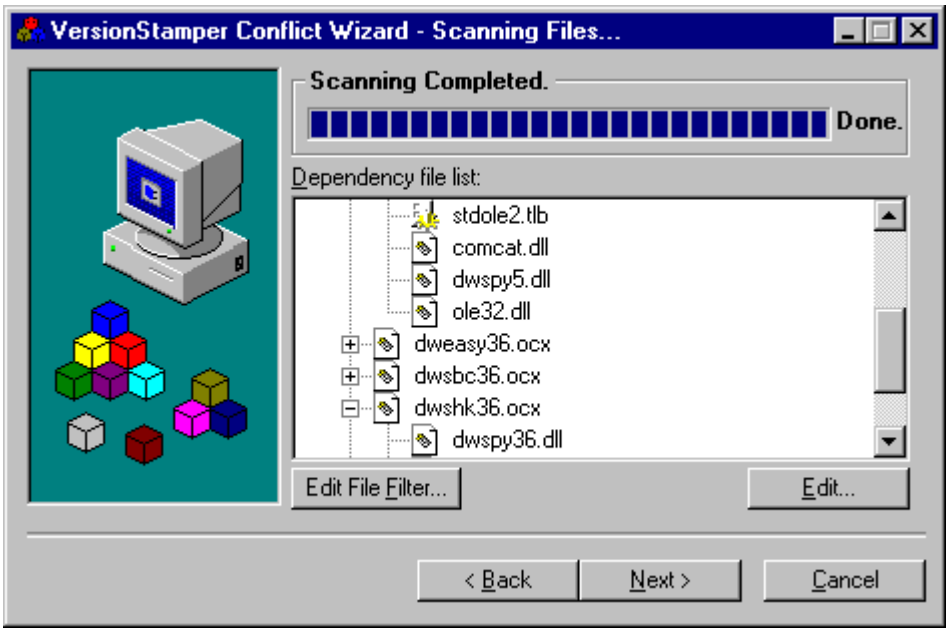

Figure 12
Edit/File Filter

After scanning the files to create a file list, the Conflict Wizard presents you the option to exclude certain files, modify the warning conditions to check for, include autoupdate information, override default search order, or attach custom messages to files. These options are accessible through the System File Filter form and File Edit form.

System File Filter

Figure 13
File Filter

An important feature that makes the editing easier is the VersionStamper System File Filter mechanism. VersionStamper maintains a list of files (which can be modified by clicking the "Edit File Filter…" button in the scanning step) which are deemed to be System Files. In other words, these files are deemed to be necessary for every Windows system and thus should already be found on any client's computer (files such as "kernel32.dll", "user32.dll", etc). Almost every Windows application and component uses at least one of these files, and it would be a waste to store the information for every occurrence of these common files. VersionStamper filters all of these files from the File List generated by scanning -- the files are simply hidden from the list and not output into the VersionStamper Script File.

Editing the File List

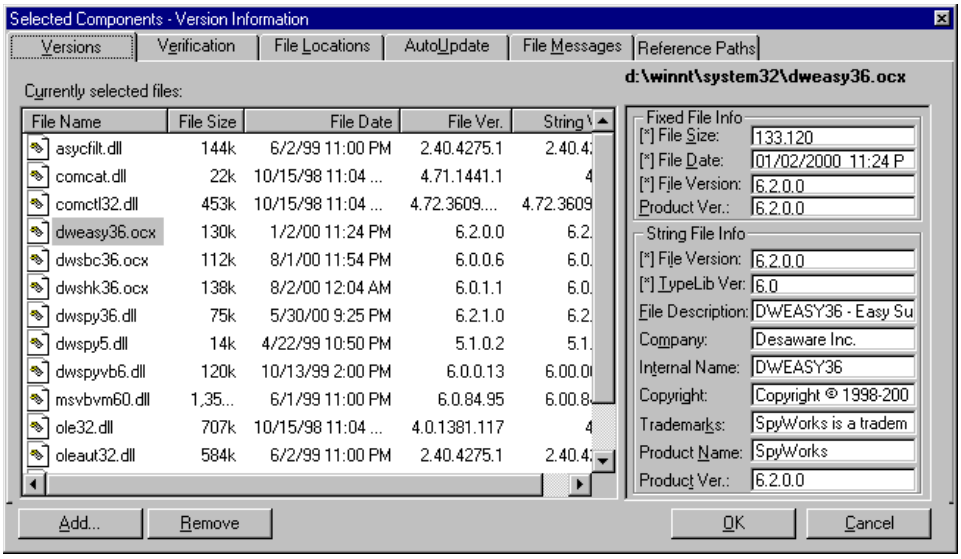

Figure 14
Edit File Window

This step is the most complex and the most powerful step of the Wizard. It allows the user to set all possible options for a VersionStamper Script File. The Edit Window operates on a list of files obtained through the previous steps. Depending on your choices, the initial information in the Edit Window may be a pre-existing loaded VersionStamper Script, a list of files generated through file and process scanning, or a new blank list if the user selected the QuickScript option in the first step of the Wizard.

The Edit Window is accessed by clicking the "Edit…" button at the bottom right of the Wizard "Scanning for Dependents" pane after the dependency scans (if any) for all the selected files are completed.

The Edit Window is organized into six tabs or panes, all of which allow the user to view and edit a certain subset of the information that can be stored in a VersionStamper script.

On the left side of each tab window is a large table containing the names of all the files in the list and the information on the files appropriate to the tab. The table is read-only; its purpose is to provide an at-a-glance view of the settings of all the files in the list.

On the right side of each tab is a data window that allows access to ALL the information for each file (not just the subset of information that is associated with the table). This is also the window in which all the editing of information will be performed. The editing is done by selecting settings with checkboxes and option buttons, as well as entering data into Text Fields.

Each Text Field can be modified in a manner similar to the way Windows Explorer allows you to rename files. The visible textboxes themselves cannot be modified directly — you can't type directly into them. Rather, the user can bring up a small "Edit Box" by either pressing the "Return" or "Enter" key in any Text Field, or by double-clicking on one.

On the first tab, where not all the Text Fields can be edited, the fields which can be modified, are marked with an "[*]"; all the other fields are read-only. When you are done typing the new value into the Edit Box, you can press "Enter" or "Return" to confirm the change (or you can click the mouse anywhere other than the box). To abort the Text Field edit and to return to the previous value, the "Escape" key may be pressed at any time.

Keep in mind that not all values are acceptable for every field; the Wizard will not allow you to enter an invalid value and will prompt you to re-enter. If you decide to cancel the edit, press Escape. Some controls may be disabled until another control is selected, these usually apply to options that depend on another option being selected.

Refer to the headings on the specific tabs listed in the following pages for more detailed explanation of the panes.

Editing Selected Files

The "Edit File List" window consists of two major panes – the "File List" pane and the "File Information" pane.

The "File List" pane is on the left of the screen and provides the list of files currently selected for addition to the script. It also provides a great deal of read-only information for each of the files, most often an easier-to read synopsis of the data displayed in the "File Information" pane. Adding or removing files from the "File List" pane determines whether those files will be added to the script that the VersionStamper Wizard generates.

The “File Information” pane is a small frame on the right side of the screen. This window provides more detailed information on the *currently selected* file (the file highlighted in the “File List” pane). This window is where all the editing of file information is done – all the information in the Edit File List window (including that in the “File List” pane) is read-only. The majority of the input controls in the “File Information” pane, however, can be edited by the user, and are the only way to edit information for specific files in the VersionStamper script. Currently, you must edit one file at a time using the File Information pane.

Many of the text boxes in the “File Information” pane cannot be modified directly. They require quite complex error-checking and data verification which would cause problems if run every time the user typed a character in the box. Thus, many text boxes in this pane use a system similar to the one which is used by Windows to allow users to rename files in Explorer.

In order to start an edit, you must either double-click on a text box, or press the “Return” key while the text box has the focus. After you have typed in the new value, you can either press “Return” again to save the value or simply click the mouse anywhere outside the text box. If the Wizard finds that the value you have entered is invalid, it will inform you of the error and restart the edit.

If you want to cancel the edit of a text box AT ANY TIME, you may push the “Escape” key on your keyboard and the Wizard will terminate the edit, restoring the old value to the text box.

The buttons at the bottom of the Edit File List window are used for managing the file list. The “Add” and “Remove” buttons add and remove files from the “File List” pane (and thus from the VersionStamper Script). The “Add” button brings up a standard “Open File” box and allows multiple files to be added to the list at the same time. The “OK” and “Cancel” buttons determine whether the changes you make (either to the list of the files or to the options of a specific file) are saved to the VersionStamper Script. Once “OK” is clicked, all previous information is lost; there is no way to recover the original list of files unless it has been saved on disk.

The Edit File List window is also organized into six sections. The purpose of this window is to allow access to the various types of information contained in the files selected in the "File List" pane. The file list is static across all of these panes. Removing a file (or adding a file) from/to one tab will do so for ALL panes, not just the one currently selected. The specifics of each pane are described next.

Version Information Tab

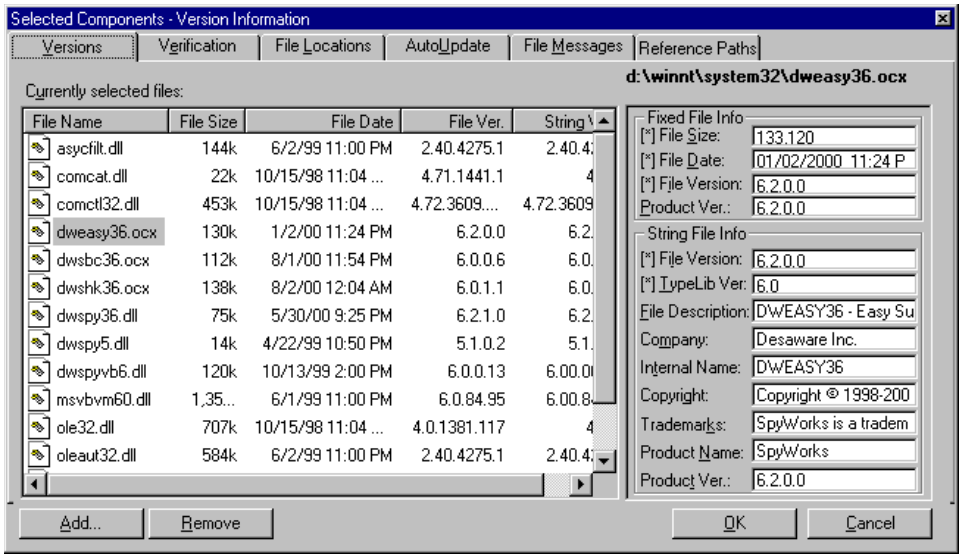

Figure 15
Version Information Tab

This window provides access to the version information stored within the VersionStamper script file. Within this window, the File Size, File Date, File Version (from the fixed version resource information of a file), and String File Version (from the string version resource information) are displayed for each file in the "File List" pane.

The “File Information” pane provides access to almost all of the fixed version resource and string version resource entries for a file. However, not all of these entries are stored in the VersionStamper script (for example, the “Copyright” string entry has nothing to do with the functionality of the VersionStamper Wizard and thus is ignored). Because this “extra” information is not stored in the script but only in the actual files themselves, these data entries will only be available for viewing when the selected file has been added in the *current Wizard session* (either through a dependency scan or manually via the “Add File” button).

These “extra” data entries **cannot** be modified. The VersionStamper Wizard marks those text boxes that can be modified and saved in the script with an “[*]” at the beginning of the caption. Thus, the “[*] File Version:” text box is editable and essential to VersionStamper operations (and saved in the script), while the “Product Version:” text box is not always available and never editable.

If you make any changes to the default values in these boxes (which are read from the file’s version information itself), the corresponding caption for the box will turn red to indicate that you have overridden the default value.

Verification Settings Tab

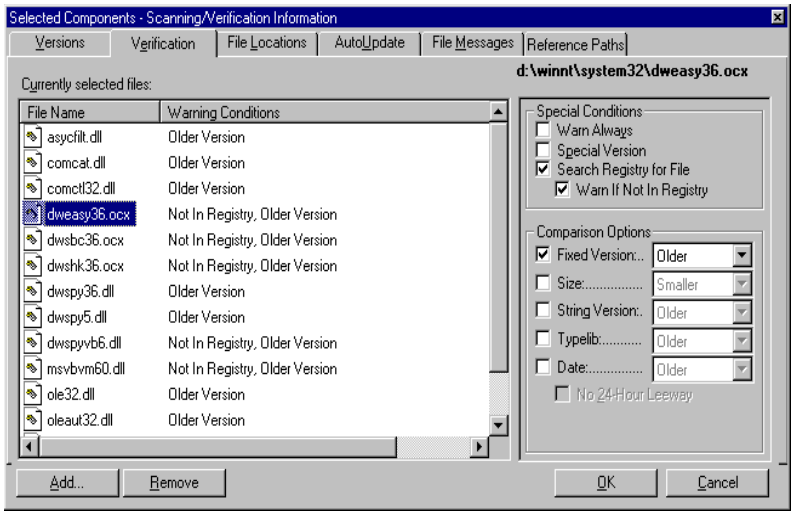

Figure 16
Verification Settings

This tab allows you to set the error conditions for your script. When a VersionStamper client reads your script on a user's machine and attempts to locate which versions of the listed files the user has, it will only produce an error (or warning) if the conditions specified in the script are met.

The VersionStamper Wizard allows practically all combinations of settings, and provides default error flag settings for various types of files that should be adequate in most cases. The default error settings are:

Comparison Options	Description of Comparison
Older Date	Files with no version information at all.
Older Version	Files with version information.
Older Version, or Not In Registry	Files with version information and a GUID (that should be registered).

The information displayed in the Scanning/Verification Information pane for this tab is a human-readable synopsis of the error conditions selected in the "File Information" pane for each file. Keep in mind that none of these error conditions are checked by the Wizard; they are simply stored in the VersionStamper script file as flags, to be read later and compared with the status of a user's machine.

The following is a description of each input control in the Scanning/Verification Information pane:

Checkbox or Combo Box	**Description**
Warn Always	Specifies that the client (the program which is reporting the conflict) should always report an error when encountering this file.
Special Version	The client will flag an error if the Special Version flag is set on the file found on the user's machine (rare).
Search Registry For File	The client will attempt to scan the registry to determine the location of the file on the user's machine.
Warn if not in Registry	Produces an error if the file was not found in the user's registry.
Fixed Version, Size, String Version, Typelib, and Date	The checkbox for each one of these conditions indicates whether any type of verification will be performed at any time on this characteristic of the file. The item selected in the combo boxes determines whether an error will be flagged on a Newer (Bigger), Older (Smaller), or Different (Either Older *or* Newer) setting for that characteristic.
Fixed Version	This is the normal method used for verification. The file's version number is compared. ***Note:*** The Date and File Version combo boxes are linked together due to an old standard in VersionStamper that must be

maintained for backwards compatibility. Thus, you cannot select both Newer Version and Older Date, etc.

Size	This will compare the file's size.
String Version	This will compare the file's Version String instead of Version Number. Not normally used, but there are many cases where developers update the file's string version but not the file's version number when a new release is made. The file's version string is first converted into a numeric value, then a numeric comparison is made (rather than a string comparison).
TypeLib	This will compare the file's TypeLib version. Not normally used, this applies only to self-registering files that exposes a Type Library. A file's Type Library version is changed whenever the file changes the interface. In Visual Basic, the Type Library appears in the project file after the GUID string.
Date	This is generally used when the file does not have version information. The file's date and time is compared. Also refer to the Allow 24-Hour Leeway checkbox.
Allow 24-Hour Leeway	Determines whether date comparisons will find two dates that differ by less than 24 hours to be identical. This should be enabled as files are now generally time-stamped relative to GMT, thus moving the files between time zones will shift the file's display time's hour by the number of time zones moved.

File Locations Tab

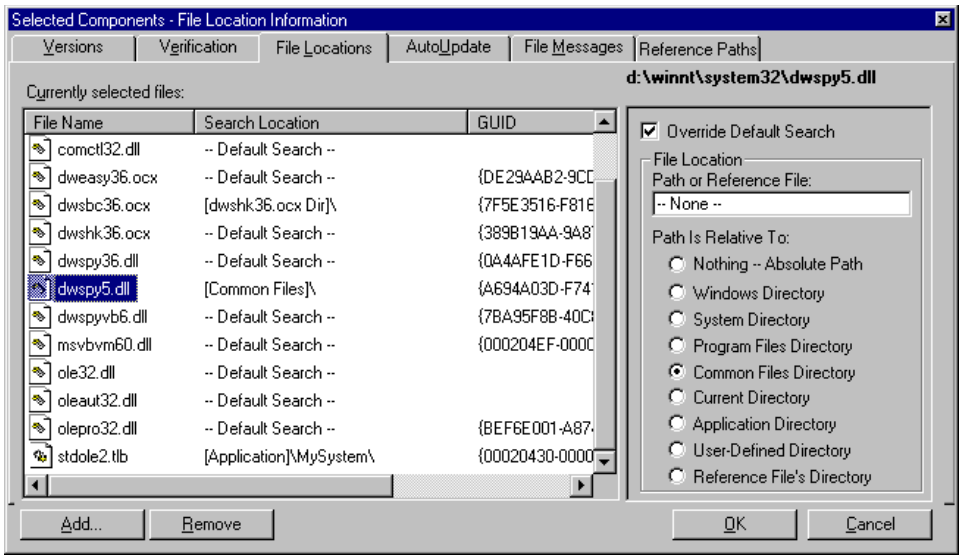

Figure 17
File Location Information

When VersionStamper searches for files listed in the VersionStamper script on the client's machine, it follows a standard procedure of searching the directory specified in the file's registry entry (if "Search In Registry" is selected in the Verification Options tab), the current directory, the Windows system directory, the Windows directory, and the directories listed in the PATH environment variable. If the file is located anywhere else on the user's machine, VersionStamper will not be able to find it.

The information recorded in this pane allows you to override this default searching path and specify another location for the file. The "File List" pane of this tab displays the current overridden location for the file (or "Default Search" if the location has not been modified), as well as the GUID of the file if one exists.

The "Override Default Search" checkbox enables the rest of the "File Information" pane and determines whether VersionStamper will use a default search or a modified search for the selected file.

The “Path or Reference File” textbox allows you to enter a path for the location of the file (either relative or absolute), or the name of a reference file. Its meaning depends on the option buttons selected below. This textbox is cannot be modified directly. You must double-click the text box or press the “Enter” key to start the edit. See the “Edit File List Overview” for more information.

The options detailed in the *Path is Relative To* section allow VersionStamper to interpret the meaning of the value in the “Path or Reference File” box. They generally indicate to which value the box is relative:

Option Button	**Description**
Nothing – Absolute Path	Indicates that the specified path is absolute, and should be used directly.
Windows Directory	Indicates that the location of the Windows Directory on the user’s system will be prefixed to the path value in order to find the location of the file. For example, if the “Path or Reference File” textbox contained “Crystal”, and this option button was selected, the client would search for this file in “C:\Windows\Crystal on the user’s machine (assuming that the Windows directory is C:\Windows”).
System Directory	Same as above, but the system directory on the user’s machine will be added to the path.
Program Files Directory	Same as above, but the “Program Files” directory on the user’s machine will be added to the path.
Common Files Directory	Same as above, but the “Common Files” directory on the user’s machine will be added to the path. The Common Files directory is usually a sub directory of the “Program Files” directory.

Application Directory	Same as above, but the installation directory on the user's machine will be added to the path (wherever the application using the client is stored).
User-Defined Directory	A user-defined directory (specified not in the script but rather at the time the script is executed, by the user) is added to the path. The User-Defined directory is specified in the VersionStamper control or component's "User-Directory" property.
Reference File	The value in the "Path or Reference File" textbox *must* be the name (no path) of a file already added to the "File List" pane. VersionStamper will search for this file in which ever directory it located the Reference File. This is normally used for dependent files of registered files where these files are installed to a private directory not found in the path. An example would be some of the Microsoft DAO or ADO files which are installed to the "\Program Files\Common Files\Microsoft Shared\" directory.

AutoUpdate Tab

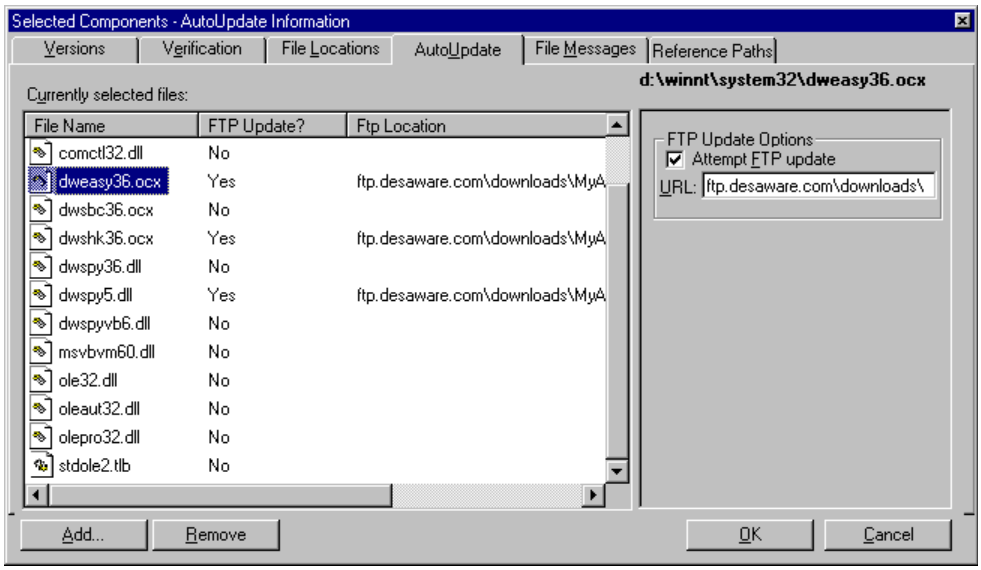

Figure 18
AutoUpdate Information

This tab allows you to specify the download locations for the selected file (and to view the locations for all files in the "File List" pane).

The text boxes in the "File Information" pane can be modified directly, and no error checking is performed. The value is assumed to be a valid FTP URL that will allow the client to retrieve the appropriate version of the file if a conflict is detected on the user's machine.

The "Attempt FTP Update" checkbox enables and disables the corresponding textbox and determines whether automatic update information will be stored for the selected file.

File Messages Tab

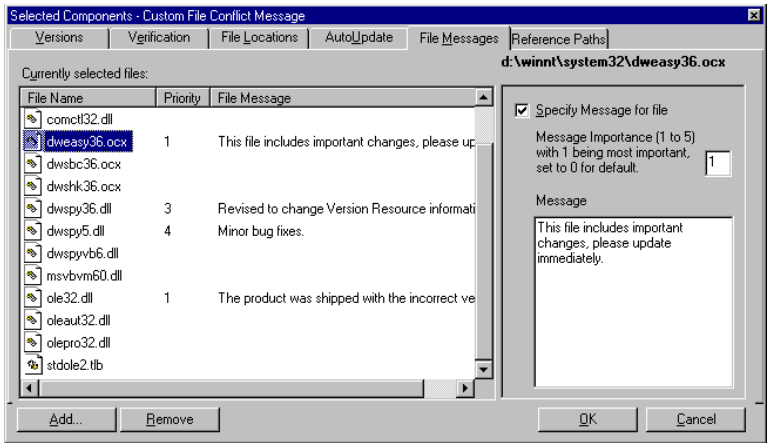

Figure 19
Customing File Conflict Message

This tab allows you to specify a custom message to be displayed for a particular file if a conflict was triggered for that file. This allows you to notify your users as to the severity of the conflict and the importance of updating to the latest file.

Select the "Specify Message for file" check box to specify a priority and message string for the selected file. Enter a priority number in the Message Priority text box to indicate the severity of the conflict. Currently, priorities 0 to 100 are reserved for Desaware use, you may specify your own custom priority numbers. Enter your custom message in the Message text box.

Reference File Paths Tab

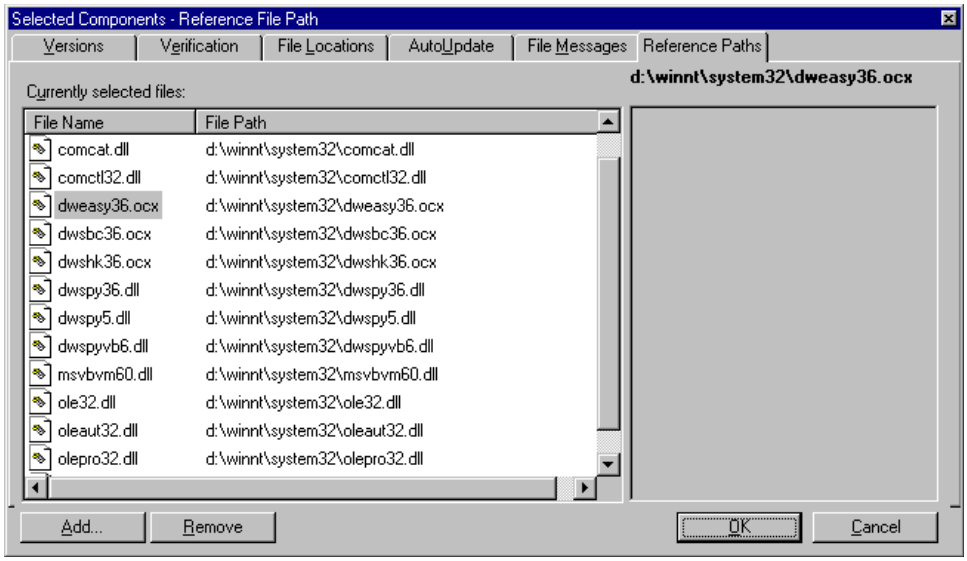

Figure 20
Reference File Paths

This tab displays the full paths for all of the reference files in the file list. The purpose of this window is to allow you to quickly verify the reference files, This is important should you have multiple copies of the reference files on your system and the scanning method selected a file different from the one that you had intended.

Saving VersionStamper Script Files

After you have performed your scan and dependency checking, and have edited the list manually via the Edit File List window (optional), you must save the script you have created to a file on your hard disk.

This step provides a textbox that displays the currently selected file name. You cannot modify this information directly. To choose a suitable location, you must click the “Browse” button which displays a standard “Save File” dialog box with which you may choose the file name and location or the new script file.

There are two file output formats from which to choose. In most cases, you would select the VerStampFileInfo2 (New Format) as it supports all of the latest VersionStamper features. You would only select the VerStampFileInfo1 format if you are creating a file list for an older application using controls from version 5.0 of VersionStamper.

When you click "Next >", the file is saved and the status of the Wizard is displayed in the next step.

Conflict Wizard Command Line Commands

With VersionStamper 6.5, you can run the VersionStamper Conflict Wizard in batch mode. Running the Wizard in this mode allows you to generate or update a VSF file. This feature allows you to add the VSF file list generation to your automatic builds. Command line arguments are not case sensitive. The following describes the command line commands.

/batch — Required for command line batch, must be specified.

/update — Updates existing VSF file, preserve warning flags, path flags, ftp location, and custom messages. Update other file information, including ftp file's date/time. Note that files within the file list are updated even if the existing files in the list are newer.

/new — Output to new VSF file, if /update is also specified will update existing file but save the results to a new file.

/outfile=c:\program files\filelist.vsf — Used with /new, specify the file to create.

/infiletype=# (# corresponds to one of the options described for infile)	/infile=(input file – depends on infiletype)
1. c:\vsffile\filename.vsf.	Used with the /update flag, updates the specified file.
2. c:\vbfile\project.vbp	Default is to scan VBP project using default settings and filter system files using verstamp.ini.
3. c:\windows\system\mydll.dll	Default is to scan DLL, EXE, OCX or other binary file using default settings and filter system files using verstamp.ini.

4. c:\setup\sysfiles	Infile specifies a directory to add all files from this directory to the script using default settings. If the /depfile is not specified, then would be similar to the Quick Script – Manually Select files (no scanning). If the /depfile is specified, then would be similar to the scan EXE, DLL, OCX option – full binary and .DEP scan.
5. C:\install\wise\setup.wse	If /filter is specified, it refers to a wise filter file, also /depfile is ignored.
6. C:\install\installshield\prosetup.ipr	If /filter is specified, it refers to an Installshield filter file, also /depfile is ignored.
7. c:\install\installshield\expsetup.htm	If /filter is specified, it refers to an Installshield filter file, also /depfile is ignored.

/depfile Perform DEP file scan for all files, not applicable when updating existing VSF file.

/refdir=<directory> If infiletype is 1, then this directory contains the directory to use to update the existing VSF file. Normally, VersionStamper would use the default Windows search to update an existing VSF file, but in this case, it will only look in the specified directory for newer files.

/filter=<file> The functionality of this switch depends on the infiletype.

- If infiletype is 2 or 3, then this file is similar to the verstamp.ini file format, and the file contains a list of files to NOT include when infile is scanning a VBP, EXE, DLL or OCX file).

- If infiletype is 5, 6, or 7 (parsing a Wise or InstallShield script file), then this file contains the drive mappings or file filters for the script file. Drive mapping is defined as a Drive from the script file mapped to the drive on the current system, it is basically used to retrieve the actual files used in the installation script. File filters are defined as file names or extensions retrieved from the installation script file that should NOT be added to the VersionStamper script file.

Batch samples

This sample demonstrates how to scan a VB Group project file to create a new VSF file list.

VsWizard.exe /batch /new /infile=c:\MyApp\TestVSGroup.vbg /infiletype=2 /outfile=c:\MyApp\TestVSGroup.vsf

This sample demonstrates how to create a new VSF file list from the reference directory.

VsWizard.exe /batch /new /infile=c:\program files\referencedirectory /infiletype=4 /outfile=c:\MyApp\FileList\RefList.vsf

This sample demonstrates how to update an existing VSF file list from the reference directory.

VsWizard.exe /batch /update /infile=c:\MyApp\FileList\RefList.vsf /infiletype=1 /refdir=f:\ToDistribute\Common Files\System32

This sample demonstrates how to update an existing VSF file list using the default file search.

VsWizard.exe /batch /update /infile=c:\MyApp\FileList\RefList.vsf /infiletype=1

Installation Script File Filter Format

You can redirect drive mappings and apply file filters when scanning an installation script file to retrieve files that are distributed with your installation. The VersionStamper Conflict Wizard offers both an interactive method to assign drive mappings or file filters and also a batch method.

The interactive method is illustrated in Figure 8 in the *Selecting Script Source* section of this manual.

The batch method reads from a file when the /infiletype=# command line switch is specified where # can be either 5, 6, or 7. The /filter=filename command line switch must also be specified where filename refers to a valid path and file name of the installation script file filter file.

The installation script file is a plain ASCII text file. The file format includes a section for specifying drive mappings, and a section for specifying file filters. The file format is as follows:

[Drive Mapping]

C=I

[File Filter]

*.txt

readme*.*

*.fr?

The sample above specifies mapping the “C” drive to an “I” drive. It also specifies excluding all files with the “txt” file extension, all files that begins with “readme”, and all files with the “fr?” file extensions.

Technical Notes

Using VersionStamper with Visual C++ and Other Environments that Support ActiveX Controls

The problems relating to distributing component based applications are not limited to Visual Basic. Today, there are many development platforms that supports ActiveX custom controls (OCXs) and ActiveX components (DLLs) in their environments. Some examples are Microsoft Visual C++, Microsoft Visual FoxPro, and Microsoft Office applications. In addition, there are many more products in development that will support ActiveX components, these will be available in the near future. Needless to say, we cannot verify that VersionStamper will work in every environment, but any development platform that fully supports ActiveX controls or components should work with VersionStamper.

Migrating From Version 1.0 to Versions 4.0 and Greater

This section summarizes the differences between VersionStamper version 1.0 and later versions. It also lists several of the key differences in researching conflicts between 16 bit and 32 bit operating systems. The VBX edition (version 1.0) has not changed in any way, you may still use it without any problems in Visual Basic version 3.0. We recommend that you use the 16 bit OCX edition if you are planning to use the 16 bit edition of Visual Basic 4.0 (or any 16 bit development platform that supports both VBXs and OCXs).

Features not Supported in Version 4.0

Since Visual Basic version 4.0 allows you to embed a standard version resource into the executable, the version embedding capabilities of VersionStamper are not implemented in the OCX editions. True, Visual Basic 4.0 only allows you to embed a subset of the standard version resource, but it supports the version number which we feel is the most important part of the version resource. One other item of note is that Visual Basic 4.0 only supports three of the four sets of version numbers that can be used. The sets are identified as *Major*, *Minor*, and *Revision* in Visual Basic's *EXE Options* dialog box. Using the numbers entered in these fields, the executable's version number translates to the following format: Major.Minor.X.Revision where X will always be set to zero.

New Features for Version 4.0 (OCX Only)

Select Files Dialog Box

VersionStamper's **Select Files** dialog box has been improved to allow better filtering of the **Available** files. You can specify the file types to be displayed and from which directories to include files. Files selected from the **Select Additional Files** command are now added to the **Selected** list instead of the **Available** list. Verifying the file size is implemented and a warning may be triggered upon finding a larger or smaller file size. An option is available to search the Registry for each file. Files that are registered during the creation of the **Selected** list will have this option enabled.

We recommend searching the Registry only for files that have a TypeLib (or GUID) identification number. These include ActiveX controls. If a selected file has a TypeLib, it is displayed as a string in the **Selected** list after the file's date and time information.

For files located during the Registry search, you may specify that VersionStamper trigger a warning if the file is not registered in the Registry. Since searching the Registry can be a very slow process, the entire registry is not searched. Instead, only the TypeLib branch of the HKEY_CLASSES_ROOT key is searched (this is the root where all components are to be registered).

Each VersionStamper control may contain two file lists, a 16 bit and 32 bit file list depending upon the environment in which your executable will run. This allows you to create a file list for 16 and 32 bit environments without having to use two separate controls. You can use either the dwvstp16.ocx or dwvstp32.ocx file to embed both file lists. During verification, the dwvstp16.ocx control checks the 16 bit file list while the dwvstp32.ocx control checks the 32 bit file list. The **Scan Project** command button can differentiate between the DLLs for the 16 bit or 32 bit file list based upon the "#ifdef Win32" or "#ifdef Win16" statements. Refer to the *Using the Select Files dialog* description in the *Embedded File Information* section for more information.

File and Directory Filtering Properties

As mentioned above, the **Select Files** dialog box allows file and directory filtering. The **FileFilter**, **FileFilterExt**, **PathFilter**, and **OtherPaths** properties allow you to specify many combinations of file and directory filters. Refer to their descriptions in other sections of this manual.

File Size Properties

You may choose to trigger a conflict if the file sizes do not match. There are three new properties that are available during verification to support this new feature. The **RefSize** property contains the size of the reference file, the **FoundSize** property contains the size of the file found, the **OtherSize**() property contains the sizes of additional files found. Refer to the verenum.bas and vervrfy.bas files for the new constants associated with file size.

VerifyObjectFile and VerifyObjectFiles Methods

These new ActiveX methods allow you to perform verification for individual components during run-time. These methods are useful for applications that may enable/disable certain portions of the reference depending upon the user's system configuration or license. You can dynamically create a list of the required files from a database or an alternative file type based upon a particular configuration of your application rather than try to embed all the possible combinations into several VersionStamper controls. Refer to the ***VerifyObjectFile Method*** information earlier in the manual for a more detail explanation.

Delay Property

Normally, when you invoke a verify or scan, it takes place immediately. However, some OLE containers disable events during the modification of an object's property or during the invoking of an object's method. This will prevent VersionStamper's events from firing during the scan or verification. If the **Delay** property is set to True, invoking the verify or scan will return immediately and the actual implementation will take place at a later time. Since the **EnumComplete** event is fired when the operation is complete, you can set a global variable in that event to notify you that the delayed operation is completed. Refer to the VSRTDemo project for an example.

Upgrading Visual Basic 3.0 Projects

After installing VersionStamper 4.0, you should be able to open any Visual Basic version 3.0 project and upgrade your dwvstamp.vbx control to the dwvstp16.ocx or dwvstp32.ocx control. All relevant properties of your VBX control will be transferred. The previous version resource properties are not saved when upgrading the control. The file list from the **Select Files** property is saved as the 16 bit file list group regardless of whether you're upgrading to the 16 or 32 bit OCX.

Search Order

The sequence in which 16 bit Windows applications search for a file has changed somewhat and the search order for 32 bit Windows applications differs from the 16 bit application. Refer to the *How Windows and Visual Basic Load Components* section for a detailed explanation of the search order.

Using Components and DLLs

For 16 bit applications, each component used is shared among all running applications. This is not the case for 32 bit applications. Each application can now load a "private" copy of the component into its own memory space. 16 bit applications running in Windows NT have the option of running in its own memory space, but 16 bit applications running in Windows 95 must share all components with other 16 bit applications.

Under Visual Basic version 3.0, when a Visual Basic application is loaded, all of the VBXs referenced by that application are loaded immediately even though the forms containing the VBXs have not been loaded. ActiveX controls (OCXs) are not loaded until they are actually referenced by the application. Also, in 16 bit Visual Basic version 4.0, VBXs are not loaded until they are actually referenced by the application. This has several implications. First, if you install any controls in a private directory which becomes the current directory during run-time, the control might pass verification by VersionStamper when your application initially loads because the current directory is searched before the Windows and System directories. But, since the VBX is no longer loaded immediately, the current directory may be changed before the form containing the VBX is actually loaded. This will change the search order and may change the path in which the VBX is found. The reverse can also occur if you

have older VBX files located in private directories. If any of the private directories are somehow designated as the current directory, then the VBX will be loaded from the private directory rather than the System directory.

Our recommendation is to be extremely careful if your application changes the current working directory and keep these considerations in mind. You may want to add a menu command to your application that will perform the verification on demand. In this way, you'll be able to verify that the files used are the correct files at certain points in your program.

In VB 3.0, specifying a path for a DLL library in a Function or Sub declaration will search the specified path after searching for the file in memory. If the file is not in the specified path, the standard search order is followed.

In VB 4.0, and later, an error will occur if the file is not found in the specified path. ActiveX controls must exist in their registered directory, otherwise an error will occur - even if they exist in another location in the search path.

If a particular version of a registered ActiveX control is not found in the Registry, then the ActiveX control with the same GUID and "largest" version number greater than the reference version number in the Registry will be used. If none is found, an error occurs. This implies that your ActiveX control will not use an older version of the same control.

VersionStamper API Functions 1.0 to 4.0 and Later

Some of VersionStamper's API functions also changed due to the change in Visual Basic version 4.0 and later. The most significant of these changes involves the handling of Visual Basic strings. It is now more difficult to obtain an address of a Visual Basic string. We recommend that you use the Visual Basic Byte array to allocate buffers. You can call the vsGetAddressForObject function to get the address of your Byte array variable. We have also included several new functions for use with Byte arrays. Refer to the *VersionStamper API Function* section for a description of vsLeftB and vsInstrB.

New Features for Version 5.0

Select Files Dialog Box for VersionStamper 5.0

VersionStamper's **Select Files** dialog box has been improved to provide better scanning of your Visual Basic project. This version supports scanning Visual Basic 4.0 and 5.0 project files. The **Scan Project** feature can also include the DLL, VBX, and OCX dependency files if the **Include Dependents** check box is selected. You can also apply a filter list of dependent files to exclude when scanning a project. This filter list may be edited by selecting the **Dependent File Filters...** command button. Some components may require additional support files that the VersionStamper scan may not detect. These files are usually documented in the component's user manuals and must be manually added to the **Selected** list.

The **Scan Dependents** command button adds a list of the dependency files of a selected file to the **Selected** list box. This is generally used on files that are manually added to the file list since the **Scan Project** feature will automatically perform a recursive scan on all dependent files.

New Utilities and Samples

VersionStamper includes some new utility programs. The following briefly describes each application.

VerDepend

VerDepend is a utility which scans an executable or dynamic link library (DLL) file and extracts the dependency files for the scanned file. Each dependent file is in turn scanned for direct dependencies to ultimately produce a complete list of dependent files. Dependency relationships in addition to detailed version information for each file may be saved to a text file.

AutoUpdate

AutoUpdate is a sample application that can update outdated files on the target computer. Updated files are retrieved from an FTP site and installed on the target computer.

VsWebLst

VsWebLst is a sample application that retrieves a text file from a host web site. This file contains file version and other conflict resolution information required to perform a file verification. Each file in the list is verified against the currently installed file on the target computer. All incompatible (older version) files are logged and a report for each file is displayed.

VsMail

VsMail is a sample application that retrieves a text file from a host web site. This file contains file version and other conflict resolution information required to perform a file verification. Each file in the list is verified against the currently installed file on the target computer. All incompatible (older version) files are logged and a report for the verification file is emailed to a specified email address.

New Features for Version 6.0

New ATL Control and Component

VersionStamper includes a new ATL based ActiveX control and ATL based ActiveX component. The dwvstp36.ocx file is similar to its predecessor (dwvstp32.ocx) except it no longer requires the Microsoft C run-time files (mfc40.dll and msvcrt40.dll). The dwvsob36.dll file is a new ActiveX component that exposes a VerScan class which is similar to the VersionStamper ActiveX control. Since it is an ActiveX component, it does not need to be placed on a form. It is used for verification only, and cannot hold a list of files to verify.

New VersionStamper Library Class Component

VersionStamper includes a new Component Library class compiled in Visual Basic 6.0 using the new ATL ActiveX component. It also includes a new Visual Basic 4.0 and 5.0 edition of the Component Library class using the ATL component instead of the MFC control.

Select Files Dialog Box for the ATL VersionStamper Control (dwvstp36.ocx)

VersionStamper's **Select Files** dialog box has been improved to include additional warning conditions and path options. The **Path** tab allows you to specify additional special path searches for specific files. The **Warnings (Advanced)** tab allows you to check a file's File Version String resource or TypeLib version number. The **Input File List** button allows you to input a list of files to verify from a VersionStamper Script file. The **Output File List** button allows you to output the currently selected file list to a VersionStamper Script File. This version supports scanning Visual Basic 4.0, 5.0 and 6.0 project files.

New Utilities and Samples

VersionStamper includes some new utility programs. The following briefly describes each application.

VsWizard

VsWizard is a utility which allows you to compile a list of files required by your application or another binary file; it has the capability of finding dependents for any "executable" file on your system (such as EXE's, DLL's, OCX's, etc.). The VersionStamper Conflict Wizard can then generate a VersionStamper Script File used by VersionStamper to verify the required components on a client machine.

VsRescue

VsRescue is a stand-alone utility that is similar to the other Visual Basic verification files and programs that are included. The difference is that this is an ATL-based edition and requires minimal run-time files so it should be able to run even in situations where your primary application cannot due to major file conflicts.

VsFileLs

VsFileLs is a sample application that retrieves a VersionStamper Script file from a local or network path. This file contains file version and other conflict resolution information required to perform a file verification. Each file in the list is verified against the currently installed file on the target computer. All incompatible (older version) files are logged and a report for each file is displayed.

Migrating from 4.0 or 5.0 to the ATL Control

If you are using the EXE embedded verification method, you can migrate the dwvstp32.ocx control to the new dwvstp36.ocx control by adding the "VersionStamper 6" control to your existing project. Then "output" your current embedded file list from the "Select Files" form of the dwvstp32 control into a "VersionStamper Script File" (VSF). Finally, go into the "Select Files" form of the new dwvstp36 control and "input" the VSF file. Replace any additional VersionStamper controls marked as "slave" with the new VersionStamper control in your project.

Code Changes

VB code changes are minimal. If you are using our template files, new templates are included to help with the migration. If you do not plan to use any of the new "warnings" or "path override" conditions, you will probably not need to change any code at all. Here are some of the more important things to note:

FileScan event now includes a new PathFlags parameter that indicates whether or not to override the default search. The StopScan parameter in this event has been changed from an Integer to a Boolean.

The StopVerify parameter in the FileConflict event has been changed from an Integer to a Boolean.

The new "Ref*", "Found*", "Other*", and "PathRefFileName" properties are accessible only during a file verification or scan.

If you are using the VersionStamper Component Class Library, you should replace the old reference with one of the new "ATL Component" references. The old VersionStamper Component Class Library uses the MFC based VersionStamper ActiveX control while the new "ATL Component" Libraries use the ATL based VersionStamper ActiveX component.

New Features for Version 6.5

New VersionStamper Script Command

New VerStampMessages command to support the display of custom messages for file conflicts.

New Features in the VersionStamper Conflict Wizard

VersionStamper 6.5 includes batch command line support for the Conflict Wizard that allows you to create a VersionStamper Script File in batch mode. Also includes support for scanning Wise Installation script files or InstallShield Installation script files to generate a list of files for verification.

New VersionStamper Library Class Component

VersionStamper includes new dwVsCmp6.dll and dwVsCmp5.dll components which uses the SpyWorks WinSock component to handle internet functions instead of the limited VersionStamper edition.

Introduction to Internet Support

VersionStamper is fundamentally a distribution toolkit for programmers who wish to safely distribute Visual Basic applications. This differs from an installation program whose primary task is to correctly install an application. VersionStamper is concerned with what happens to your application after it has been installed in order to make sure that your application will continue to run correctly as the target system undergoes changes due to configuration changes, updates, and installation of other applications that may use different versions of the same components that your application uses.

There are two approaches to using VersionStamper. One is to use the predefined objects and components that we provide in order to quickly add component verification capabilities to any application. The other is to modify the source code for these objects in order to add customized component verification to your application.

Starting with version 5.0, the VersionStamper philosophy and approaches were applied to the realm of the internet. The product includes a number of components that can be used as-is, or modified to suit your own needs. These components provide the following features:

Flexible file list support: Prior editions of VersionStamper were designed primarily to work with lists of files that were embedded into the executable itself. This maintained an air-tight correspondence between the required version information and the executable. Now VersionStamper allows you to load file lists dynamically from a disk file or from a web site that you maintain.

File Download support: Classes are provided that allow you to download updated files from an FTP site or a web site. File download using the FTP or HTTP protocols are supported.

Choose your Internet Support: The VersionStamper internet objects are designed to work with any FTP or HTTP control. Samples are included that use the Microsoft MSInet control, Crescent Internet Toolkit control, and Desaware's SpyWorks Winsock component. You can easily adapt the package to use any TCP/IP internet component package. The compiled VersionStamper components use the Desaware SpyWorks Winsock component.

EMail notification support: The product allows you to email conflict information, allowing your technical support department to automatically receive the detailed information about the target system that is needed to resolve component conflicts. In some cases, if you perform scans regularly, it may notify you of a developing problem before it becomes apparent to the customer.

Automatic file updates: VersionStamper includes objects that can download and install updated components automatically.

Custom messages for file conflicts: This edition of VersionStamper allows you to assign custom conflict messages for a file conflict. Custom messages may be displayed in addition to the normal file conflict information. Use custom messages to specify additional instructions your user must perform when updating a file, to notify them of the changes with a new file, etc.

VersionStamper also includes a number of sample applications that demonstrate how to effectively use the VersionStamper objects to take full advantage of these internet based features.

The Desaware Winsock Library

The addition of internet features to VersionStamper posed an interesting dilemma to us. Exactly how do we provide access to internet features from Visual Basic? We considered licensing one of the multitude of internet controls available. We considered standardizing on using the Microsoft internet controls. But ultimately we decided to design the package with enough flexibility that you could use whichever internet control you wished.

Then, to be sure that you would be able to use VersionStamper's new 32 bit internet features without having to purchase another software package, we've included our own Desaware SpyWorks Winsock component (dwsock?.dll).

For most cases, you do not need to access the SpyWorks WinSock component directly as the VersionStamper Library objects exposes the necessary properties and functions that you would use. In case you do need to access the FTP or HTTP objects directly, their functions are described in the VersionStamper Help file.

Using a Different Internet Control

VersionStamper includes Desaware's SpyWorks WinSock component to handle internet transactions. But you can use another internet control by replacing several files in the dwVsCmp? project. The files to replace are verFtp.cls, verWebPg.cls, verDWINet.cls, and any BAS files that are specific to a particular internet control. The following outlines the steps necessary to use another internet component.

- Add the new internet component to this project.
- Replace the VerDWINet class with a class that specifically wraps the internet component. This class can expose the necessary internet properties for the higher level classes. At a minimum, it should expose the **RequestTimeout** property and the RetrieveBinaryFile and RetrieveURLPage functions. This class name should have a similar naming convention as Ver??Net. This class should be a private class.
- Replace the VerFTP class with a class that specifically wraps the Ver??Net class. This class must expose the FTPFile function and the **RequestTimeout** property which are accessed by the other VersionStamper classes.
- Replace the VerWebPage class with a class that specifically wraps the Ver??Net class. This class must expose the RetrieveURLPage function and the **RequestTimeout** property which are accessed by the other VersionStamper classes.

VersionStamper Components

The VersionStamper components are written in Visual Basic and include full Visual Basic source code. We encourage you to use the actual component in your projects - it is quite efficient and has a great deal of functionality.

Future VersionStamper releases will probably extend several of these classes and include new ones. The classes exposed by the VersionStamper Component Library are summarized in the next section. Refer to the VersionStamper Help file for additional details on each class. Also, please review the VersionStamper readme file for the latest information.

Before you try to **use or ship** these components, please read the following sections: *Before You Begin* and *Distribution and Licensing*.

Before You Begin

There are a few other issues that you should be aware of with regards to this component.

- Not all of the functionality is available in both 16 and 32 bit. The 32 bit edition has additional capabilities.
- When using 32 bit, we recommend using the latest VersionStamper Library ("Desaware VersionStamper ATL SW VB? Component"), other 32 bit editions may not have the same functionality.
- The different editions of the 32 bit VersionStamper components are similar, the difference being which edition of Visual Basic they were compiled and which VersionStamper control or component was used. Refer to the ***Which VersionStamper file Should I use?*** section for details on the different editions.
- If you recompile this project for any reason (such as to use different internet controls), you need to rename your project and rename the VersionStamper component file name to something else. Otherwise, you may cause incompatibilities with other applications that are using this particular file. For this reason, all the project information for the VersionStamper component project is cleared.

Distribution and Licensing

This component is provided with the intent that it be used in applications and components that offer significant and primary functionality beyond that of this component. In other words - our license does not permit you to use the code provided here or the component itself to market another component that provides the same or slightly modified functionality - like another version conflict detection tool or utility.

You may not distribute the source code provided with the component to anyone who has not purchased VersionStamper version 6 or later.

You may not, under any circumstances, distribute the files dwverd36.dll, dwverd32.dll, dwverd16.dll, or dwverdes.dll.

You may distribute this component on a royalty free basis with your compiled applications. You may distribute modified versions of this component with your compiled applications as long as you include either our licensing code or other code that will prevent the component from being programmed without a valid VersionStamper license. If you have any questions regarding distribution, please do not hesitate to contact us directly.

The VersionStamper Component Library must be registered in order for it to work. Refer to the **Files Required for Distribution** section for details on required files to distribute for each component.

VersionStamper Classes Summary

All of the VersionStamper classes including internet support are compiled into several different permutations of the dwvercls.dll file. All of the VersionStamper classes are written in Visual Basic, and will run in Visual Basic 4.0 or later. All of the VersionStamper classes are exposed from this DLL in 32 bit. Since you cannot compile an OLE DLL in 16 bit Visual Basic, you need to add the appropriate classes to your 16 bit projects. Furthermore, when running in 16 bit, you must have a license for an internet control in order to use any of VersionStamper's internet features. The compiled DLL file must be registered before it can be used. To use dwVerCls, select the References menu command and add the "Desaware VersionStamper Components" (or one of its permutations, "Desaware VersionStamper* Components") reference to your project. Please refer to the *Which VersionStamper File Should I Use* section of this manual should you need clarification on which component to select.

The following briefly describes each of VersionStamper's classes.

VerConflict

The VerConflict class holds conflict information for each file that triggers a conflict situation during the verification process. The VerConflict class's LogFileConflict method is normally called whenever a **FileConflict** event is triggered by the VersionStamper ActiveX control. Calling this method logs all of the file conflict information into the private data members of the class. All of the conflict information can then be retrieved through properties exposed by the VerConflict class. This class also includes additional exposed methods that can fill a list box with a list of all conflicting files, or retrieve the conflict information for a particular file in a report format.

VerControl

The VerControl object is used to set or retrieve global variables exposed by the VersionStamper ActiveX Control or Component that are not contained within any of the other VersionStamper Library classes.

VerDWInet

The VerDWInet object is a wrapper class that uses the Desaware WinSock ActiveX component to handle FTP and HTTP file retrieval. This class exposes the generic RetrieveURLPage and Retrieve-BinaryFile functions. This class can be used by the VerFTP or VerWebPage classes.

VerFileDateTime

The VerFileDateTime object is used to set and retrieve file times for files. This class supports file creation, last access, and last write time values that are supported by the FAT, NTFS, and HPFS file systems. The file's last write time value is normally referred to as the file's date/time value.

VerFTP

The VerFTP class is a generic class that higher level classes call to handle FTP. This class should expose the generic FTPFile function which retrieves a file from a specified location. This class should call a wrapper class which uses a specific ActiveX control or component that supports FTP.

VerParseCommandLine

The VerParseCommandLine class is used to parse a command line containing valid VersionStamper formatted command data. After the ParseLine method of this class is called, the class will contain the type of VersionStamper data this line holds and the rest of the command line. Depending on the command type, another class will be used to interpret the rest of the command line.

VerParseFile1

The VerParseFile1 object is a high level class used to parse a file containing valid VersionStamper command lines and carry out the specified commands. This object is typically used to perform file verification based on a local text file.

VerParseFormat

The VerParseFormat class supercedes the VerParseFormat1 class and is used to parse a command line containing VersionStamper formatted data in the VerStampFileInfo1 or VerStampFileInfo2 format. The VerifyObjectFile method is called to verify a file using the data presented in the current command line. This class requires the use of a VerConflict object and the VersionStamper ActiveX control or component.

VerParseFormat1

The VerParseFormat1 class is used to parse a command line containing VersionStamper formatted data in the VerStampFileInfo1 format. The VerifyObjectFile method is called to verify a file using the data presented in the current command line. This class requires the use of a VerConflict object and the VersionStamper ActiveX control or component.

VerParseWebPage1

The VerParseWebPage1 class is a high level class used to retrieve a text file from a URL address, parse the file, and return conflict information in a VerConflict object or string. This class references many of the other VersionStamper classes.

VerSecurity

The VerSecurity class deals with Windows NT security. Currently, this edition sets the privilege for the current process to allow it to reboot the system.

VerSmartReplaceFile

The VerSmartReplaceFile class can be used to install files retrieved via FTP or other methods. This class supports replacing files that are in use, as well as support for long file names. This class references the VerSecurity class.

VerWarningFlags

The VerWarningFlags class converts warning flags into string descriptions.

VerVersionInfo

The VerVersionInfo class is used to retrieve version resource information from files that contain a version resource. This class is not exposed by the VersionStamper Component Library, but you may add the class module file directly into your project.

VerWebPage

The VerWebPage class is a generic class that higher level classes call to retrieve a text file from a URL address. This class should expose the generic RetrieveURLPage function which retrieves a text file from a specified URL location. This class should call a wrapper class which uses a specific ActiveX control or component that supports HTTP.

VerMSINET

The VerMSINET class is a generic wrapper class that uses the MSINET ActiveX control to handle FTP and HTTP file retrieval. This class should expose the generic RetrieveURLPage and RetrieveBinaryFile functions. This class can be used by the VerFTP or VerWebPage classes. This class requires the form frmInet which contains the actual ActiveX control. Not necessarily used in all editions of the VersionStamper component.

VerCINET

The VerCINET class is a generic wrapper class that uses the Crescent Internet Toolpak ActiveX controls to handle FTP and HTTP file retrieval. This class should expose the generic RetrieveURLPage and RetrieveBinaryFile functions. This class can be used by the VerFTP or VerWebPage classes. This class requires the form frmInet which contains the actual ActiveX control. Not necessarily used in all editions of the VersionStamper component.

frmInet.frm

This form is used to hold the actual internet ActiveX controls used to perform the internet operations. This form may be used by higher level classes. This form is used only in combination with an Internet ActiveX control.

VsMain.frm

This form is used to hold the actual VersionStamper ActiveX control used to perform the file verification. This form may be used by higher level classes. This form is used only in combination with an Internet ActiveX control.

Technical Information

This part of the manual contains technical information that will help you to work directly with the version API, or use VersionStamper under other environments.

The Windows Version API

Version resource support is provided by a set of functions in the VERSION.DLL (or VER.DLL in 16 bit Windows). This version API can be called directly from Visual Basic. We have provided a set of functions in the verinfo.bas file that provides an easy to use interface to the version API, in addition to a great deal of sample code that demonstrates how it is used. We also included a VB class VerVersionInfo (vsverinf.cls) that wraps the version functions.

The information in this chapter will help you to understand the code in the VerInfo project, and provide the information that you need to access the version API directly should you choose to do so.

A word of thanks. Portions of this section are reprinted from the "Visual Basic Programmer's Guide to the Windows API" and "Dan Appleman's Visual Basic Programmer's Guide To The Win32 API" with the kind permission of Macmillan Press.

All of the version functions described here require that the dynamic link library VERSION.DLL (or VER.DLL in 16 bit) be present on your system. This dynamic link library is provided with Windows 3.1, Windows 95/98, Windows NT, Windows 2000 and many other applications. It is not included with VersionStamper.

The functions listed in the following table are used to support version resources and to aid in the installation of files based on their version stamp. Both 16 and 32 bit declarations are included.

Version Control API Functions

Function	Description
GetFileResource	May be used to load a version resource. GetFileVersionInfo is preferred however. This function is not supported in Win32.

GetFileResourceSize	May be used to determine the size of a version resource. GetFileVersionInfoSize is preferred however. This function is not supported in Win32.
GetFileVersionInfo	Loads a version information resource block.
GetFileVersionInfoSize	Determines whether version information is available. Returns the size of a version information resource block.
VerFindFile	Determines the recommended destination directory in which to install a file.
VerInstallFile	A powerful function for installing a file onto a system. Supports expansion of compressed files and version checking.
VerLanguageName	Determines the text name of a language based on a standard language code.
VerQueryValue	Determines the value of a version attribute for a file.

The Version Data Structures

The version stamp for a file can have many different components depending on the file. Three of these are commonly used by installation programs and are used in virtually all programs that use version stamping. The first component is a data structure known as the FIXEDFILEINFO structure. It contains numeric version information and flags defining the type of the file. The second component defines the language and code page translations that exist in the version resource. The third component consists of one or more strings called StringFileInfo attributes.

The FIXEDFILEINFO Structure

The FIXEDFILEINFO data structure is present in every file that has a version stamp. It is defined below.

VB Declaration:

```
Type FIXEDFILEINFO ' 52 Bytes
dwSignature As Long
dwStrucVersion As Long
```

```
dwFileVersionMS As Long
dwFileVersionLS As Long
dwProductVersionMS As Long
dwProductVersionLS As Long
dwFileFlagsMask As Long
dwFileFlags As Long
dwFileOS As Long
dwFileType As Long
dwFileSubtype As Long
dwFileDateMS As Long
dwFileDateLS As Long
End Type
```

The following table describes each of these fields:

Field	Type	Description
dwSignature	Long	Always contains &HFEEF04BD.
dwStrucVersion	Long	The version of this structure. Will be greater than 29.
dwFileVersionMS	Long	The high 32 bits of the file version number.
dwFileVersionLS	Long	The low 32 bits of the file version number.
dwProductVersionMS	Long	The high 32 bits of the product version number.
dwProductVersionLS	Long	The low 32 bits of the product version number.

dwFileFlagsMask	Long	Any combination of the constants described in the Version File Flags table that follows. The presence of a flag in this parameter indicates that the value of the dwFileFlags parameter for that bit is valid.
dwFileFlags	Long	Any combination of the constants shown in the Version File Flags table.
dwFileOS	Long	One of the constants defined later in this chapter in the Version File Operating System Types table.
dwFileType	Long	One of the constants defined later in this chapter in the Version File Types table.
dwFileSubtype	Long	One of the constants that begins with the VFT2_ prefix. Defined later in this chapter and in the verinfo.bas file.
dwFileDateMS	Long	The high 32 bits that specify the date and time of the file's creation. The Microsoft resource compiler does not set this value.
dwFileDateLS	Long	The low 32 bits that specify the date and time of the file's creation. The Microsoft resource compiler does not set this value.

Version numbers are typically 64 bits long to allow for numeric comparisons of versions. However, the internal structure of these numbers deserves further clarification.

The most significant 32 bits, the high 16 bits comprise the major revision number, and the low 16 bits comprises the minor revision number. Thus Windows 3.10 will have &H0003000A as its major version number. The 3 in the high order word indicates version 3, the hexadecimal A represents the number 10 for a minor revision number of .10.

This technique is used on the lower 32 bits to allow an even finer resolution of versions, but these numbers are typically used only in a development environment and are rarely used by either application programmers or users. Some programs that can read version information do not even bother displaying the minor revision numbers.

When using dwvstamp.vbx, the control embeds a version resource, that sets most of the fields in this structure automatically. The **FileVersion** and **ProductVersion** properties set the dwFileVersionMS, dwFileVersionLS, dwProductVersionMS and dwProductVersionLS fields. The **FlagDebug, FlagPatched, FlagPrerelease, FlagPrivate and FlagSpecial** properties set bits in the dwFileFlags and dwFileFlagsMask fields.

Version File Flags

The following table lists the flags defined in the verinfo.bas file that are used in the **dwFileFlags** parameter to specify general information about the file.

Constants	Description
VS_FF_DEBUG	This file contains debugging information.
VS_FF_INFOINFERRED	The version resource for this file is dynamically allocated and some of the blocks in the resource may be incorrect.
VS_FF_PATCHED	This file has been patched. It may differ from the original file that has the same version number.
VS_FF_PRERELEASE	This is a pre-release version of the file.
VS_FF_PRIVATEBUILD	This version of the file is built specially as defined by the PrivateBuild StringFileInfo string.
VS_FF_SPECIALBUILD	This version of the file is built specially as defined by the SpecialBuild StringFileInfo string.

Version File Operating System Types

The following table lists the flags that are used in the **dwFileOS** parameter to specify general information about the file.

Constant	Target Operating System for this File
VOS_UNKNOWN	Undefined or unknown to Windows.
VOS_DOS	MS-DOS.
VOS_NT	Windows NT.
VOS_WINDOWS16	16 bit Windows (includes Windows 3.0 and 3.1).
VOS_WINDOWS32	32 bit Windows (includes Windows 95 and Windows NT).
VOS_OS216	16 bit OS/2.
VOS_OS232	32 bit OS/2.
VOS_PM16	16 bit Presentation Manager.
VOS_PM32	32 bit Presentation Manager.

The above flags can be combined (OR) to indicate that the file was designed for one operating system running on another. Here are just a few examples.

VOS_DOS_WINDOWS16	16 bit Windows (includes Windows 3.0 and 3.1) running under MS-DOS.
VOS_DOS_WINDOWS32	32 bit Windows running under MS-DOS.
VOS_OS16_PM16	16 bit Presentation Manager running under 16 bit OS/2.
VOS_NT_WINDOWS32	32 bit Windows running under Windows NT.

Version File Types

The following table lists the constants defined in verinfo.bas that are used in the **dwFileType** parameter to specify the type of the file.

Constant	Type of file
VFT_UNKNOWN	Unknown
VFT_APP	Application

VFT_DLL	Dynamic Link Library. This includes most Visual Basic custom controls.
VFT_DRV	Driver. The type is specified by the dwFileSubType parameter.
VFT_FONT	Font. The type is specified by the dwFileSubType parameter.
VFT_VXD	Virtual device driver.
VFT_STATIC_LIB	A static link library.

The Translation Table

This table defines the language and code page combinations that are included in the version resource. It takes the form of an array of integer pairs. The first integer is the language code as listed in the Windows Languages table that follows. The second integer defines the character set or code page to use for that language as shown in the Windows Character Sets table that follows.

The language and code page definitions are not followed consistently by every application. For this reason, it is important to look at the translation table if one exists. You need accurate language and code page information to access the StringFileInfo strings that are defined in the next section.

If a translation table is not defined, the most common language/code combinations are &H040904E4, indicating U.S. English and the standard multilingual Windows character set, and &H04090000, which indicates U.S. English and the 7 bit ASCII character set.

The VerInfo example program illustrates how the translation table for a version resource can be read, and shows how to find the U.S. English entry in the table.

The dwvstamp.vbx control allows you to specify up to 16 languages character set pairs in an application. Each of these languages may have its own set of StringFileInfo strings.

Windows Character Sets

Identifier Value	Character Set
0	7 bit ASCII
&H3A4	Windows – Japan

&H3B5	Windows – Korea
&H3B6	Windows – Taiwan
&H4B0	Unicode
&H4E2	Windows - Latin (Eastern Europe)
&H4E3	Windows – Cyrillic
&H4E4	Windows - Multilingual (U.S. Standard)
&H4E5	Windows – Greek
&H4E6	Windows – Turkish
&H4E7	Windows – Hebrew
&H4E8	Windows – Arabic

Windows Languages

Language Value	Language
&H401	Arabic
&H402	Bulgarian
&H403	Catalan
&H404	Traditional Chinese
&H405	Czech
&H406	Danish
&H407	German
&H408	Greek
&H409	U.S. English
&H40A	Castilian Spanish
&H40B	Finnish
&H40C	French
&H40D	Hebrew
&H40E	Hungarian
&H40F	Icelandic
&H410	Italian
&H411	Japanese
&H412	Korean
&H413	Dutch

&H414	Norwegian - Bokmål
&H415	Polish
&H416	Brazilian Portuguese
&H417	Rhaeto-Romanic
&H418	Romanian
&H419	Russian
&H41A	Croato-Serbian (Latin)
&H41B	Slovak
&H41C	Albanian
&H41D	Swedish
&H41E	Thai
&H41F	Turkish
&H420	Urdu
&H421	Bahasa
&H804	Simplified Chinese
&H807	Swiss German
&H809	U.K. English
&H80A	Mexican Spanish
&H80C	Belgian French
&H810	Swiss Italian
&H813	Belgian Dutch
&H814	Norwegian - Nynorsk
&H816	Portuguese
&H81A	Serbo-Croatian (Cyrillic)
&HC0C	Canadian French
&H100C	Swiss French

StringFileInfo Data

The StringFileInfo entries in a version resource are strings that describe certain characteristics of the file. A file may contain unique strings for each language supported, thus the language/code page information is used to access this data as well. Refer to the description of the VerQueryValue function for further information on retrieving these values.

The dwvstamp.vbx control allows you to set the majority of these strings. It supports up to 16 sets of strings at once, each with its own language.

Version StringFileInfo Data Names

The standard StringFileInfo strings are listed in the following table. Not all of the strings defined are present in every file.

StringFileInfo Name	Description
Comments	General comments.
CompanyName	The name of the company.
FileDescription	A description of the file.
FileVersion	The version of the file in string form.
InternalName	The internal module or application name.
LegalCopyright	A copyright notice.
LegalTrademarks	Trademark notices.
OriginalFilename	The original name of the file. Useful in determining if the file has been renamed.
PrivateBuild	A description of this build if the VS_FF_PRIVATEBUILD flag was set in the dwFileFlags field of the FIXEDFILEINFO structure.
ProductName	The name of the product to which this file belongs.
ProductVersion	The version in string form of the product to which this file belongs.

SpecialBuild	A description of this build if the VS_FF_SPECIALBUILD flag was set in the dwFileFlags field of the FIXEDFILEINFO structure.
OLESelfRegister	Indicates that this file can register itself into the Registry. Most ActiveX controls are "self register" controls. The dwvstamp.vbx does not have a corresponding property for this string.

Version API Function Declarations

GetFileVersionInfo

VB Declaration:

(32 bit) Declare Function GetFileVersionInfo& Lib "version.dll" Alias "GetFileVersionInfoA " (ByVal lptstrFilename As String, ByVal dwHandle As Long, ByVal dwLen As Long, lpData As Byte)

(16 bit) Declare Function GetFileVersionInfo% Lib "ver.dll" (ByVal lpszFileName$, ByVal handle&, ByVal cbBuf&, ByVal lpvData&)

Description: Retrieves the file version information from a module that supports version stamping.

Parameter	Type	Type/Description
lpszFileName	String	The name of the module file from which to load version information.
handle	Long	Not used with Win32 (set to zero). Under Win16, the handle returned by the GetFileVersionInfoSize buffer. May be zero to search for version information in the file.
cbBuf	Long	The size of the lpvData buffer.
lpvData	Long or Byte	The address of a buffer to load with version information. This buffer must be at least cbBuf bytes long. May reference the first byte in a Byte array.

Return Value: **Long (32) or Integer (16):** True (non-zero) on success, zero otherwise.
Comments: The **VerQueryValue** API can be used to extract data from the resource buffer loaded by this function.

GetFileVersionInfoSize

VB Declaration:

(32 bit) Declare Function GetFileVersionInfoSize& Lib "version.dll" Alias "GetFileVersionInfoSizeA " (ByVal lptstrFilename As String, lpdwHandle As Long)

(16 bit) Declare Function GetFileVersionInfoSize% Lib "ver.dll" (ByVal lpszFileName$, lpdwHandle&)

Description: Retrieves the size of the buffer required to hold the file version information for a module that supports version stamping.

Parameter	Type	Description
lpszFileName	String	The name of the module file from which to load version information.
lpdwHandle	Long	A long variable to load with a handle to the version resource information for the specified file. This parameter is not used under Win32, but must still be provided.

Return Value: **Long (32) or Integer (16):** True (non-zero) on success, zero otherwise. Zero is also returned if the file does not support version stamping.

VerFindFile

VB Declaration:

(32 bit) Declare Function VerFindFile& Lib "version.dll " Alias "VerFindFileA" " (ByVal fl As Long, ByVal FileName As String, ByVal WinDir As Long, ByVal szAppDir As String, ByVal szCurDir As String, CurDirLen As Long, ByVal DestDir As String, DestDirLen As Long)

(16 bit) Declare Function VerFindFile% Lib "ver.dll " (ByVal fl%, ByVal FileName$, ByVal WinDir&, ByVal AppDir$, ByVal CurrDir$, CurDirLen%, ByVal DestDir$, DestDirLen%)

Description: This function is used to determine where a file should be installed.

Parameter	Type	Type/Description
fl	Long (#@) or Integer (16)	Currently only VIFF_ISSHAREDFILE is defined to indicate that the file can be shared by multiple applications. If this flag is specified, this function will recommend that the file be installed in the Windows or System directory. If this parameter is zero, the function will recommend that the file be installed in the application directory.
Filename	String	The name of the file to be installed. This string should not include the path of the file.
WinDir	Long	Zero
AppDir	String	The full path name of the directory in which the application and all its related files are being installed.
CurrDir	Long (32) or Integer (16)	A buffer to load with the directory containing the existing version of the file. If a version of the file is not already present, the buffer will be loaded with the directory of the source file. Must be allocated to be at least 260 bytes long.
CurrDirLen	Integer	The length of the CurrDir buffer. This variable will be set to the actual number of characters loaded in the buffer.
DestDir	String	A buffer to load with the directory in which the new file should be installed. Must be allocated to be at least 260 bytes long.

DestDirLen	Long (32) or Integer (16)	The length of the DestDir buffer. This variable will be set to the actual number of characters loaded in the buffer.

Return Value: **LONG (32) or Integer (16) :** One of the following values defined in verinfo.bas:

VFF_CURNEDEST	Indicates that the existing version of the file is not in the directory specified by DestDir, which is where this function recommends that the new version be installed.
VFF_FILEINUSE	Indicates that the existing file is currently in use and that it may not be deleted at this time.
VFF_BUFFTOOSMALL	Indicates that one or both of the DestDir or CurrDir buffers are too small to hold the directory name.

VerInstallFile

VB Declaration:

```
Declare Function VerInstallFile& Lib “version.dll” Alias “ VerInstallFileA” (ByVal uFlags As Long, ByVal szSrcFileName As String, ByVal szDestFileName As String, ByVal szSrcDir As String, ByVal szDestDir As String, ByVal szCurDir As String, ByVal szTmpFile As String, lpuTmpFileLen As Long)
```

```
Declare Function VerInstallFile& Lib “ver.dll” (ByVal fl%, ByVal SrcFile$, ByVal DstFile$, ByVal SrcDir$, ByVal DstDir$, ByVal CurrDir$, ByVal TmpFile$, TmpFileLen%)
```

Description: This function is used to install a file. It uses the information provided by the VerFindFile function to determine where a file should be installed. The function works by first comparing the version stamps of the two files. If the source file is a newer and compatible version, the source file is copied to a temporary file in the destination directory, decompressing the file if it is compressed. Then the existing version of the file is deleted and the temporary file is renamed to match the destination file name.

If the versions are not compatible, the temporary file may still exist in the destination directory. At that time you have the option of either deleting the temporary file, or installing the file with the VIFF_FORCEINSTALL flag set to force the installation.

Parameter	Type	Description
fl	Long (32) or Integer (16)	A combination of the following constant values: VIFF_FORCEINSTALL: Forces installation of the source file without version checking. VIFF_DONTDELETEOLD: Does not delete the existing version of the file if it is not in the destination directory. If it is in the destination directory, it will be overwritten by the new file.
SrcFile	String	The name of the file to install. This should not include the path name for the file.
DestFile	String	The name that should be given to the file once it is installed. This is usually the same as SrcFile.
SrcDir	String	The source directory from which the new version of the file is copied.
DstDir	String	The directory in which to install the new version of the file. The DestDir buffer returned by the VerFindFile function is typically used for this parameter.
CurrDir	String	The directory that contains the current version of the file. The CurrDir buffer returned by the VerFindFile function is typically used for this parameter. If the string is empty, no previous version of the file exists on the system.

TmpFile	String	A buffer to load with the name of a temporary copy of the source file. Must be allocated to be at least 260 bytes long.
TmpFileLen	Long (32) or Integer (16)	The length of the TmpFile buffer. This variable will be set to the actual number of characters loaded in the buffer including the terminating NULL character. If VIFF_FORCEINSTALL is specified and TmpFileLen is not zero, the temporary file will be renamed to the name specified in the SrcFile parameter.

Return Value: **Long:** An integer which contains a combination of one or more of the constants listed in the table that follows:

VerInstallFile Result Constants

Constant	Description
VIF_TEMPFILE	A temporary file that is a copy of the new file which is present in the destination directory and needs to be deleted.
VIF_MISMATCH	The existing file differs from the new file in one or more version attributes. Can be overridden by specifying VIFF_FORCEINSTALL in the fl parameter.
VIF_SRCOLD	The new version of the file is older than the existing file based upon the version stamping of the files. Can be overridden by specifying VIFF_FORCEINSTALL in the fl parameter.

VIF_DIFFLANG	The new version of the file has a different language or code page value from the existing file. Can be overridden by specifying VIFF_FORCEINSTALL in the fl parameter.
VIF_DIFFCODEPG	The new version of the file needs a code page that is not present in the version of Windows now running. Can be overridden by specifying VIFF_FORCEINSTALL in the fl parameter.
VIF_DIFFTYPE	The new version of the file differs in type, subtype or target operating system from the existing version. Can be overridden by specifying VIFF_FORCEINSTALL in the fl parameter.
VIF_WRITEPROT	The pre-existing file is write protected.
VIF_FILEINUSE	The existing file is in use.
VIF_OUTOFSPACE	There is insufficient disk space on the destination drive.
VIF_ACCESSVIOLATION	Operation failed due to an access violation.
VIF_SHARINGVIOLATION	Operation failed due to a sharing violation.
VIF_CANNOTCREATE	The temporary file could not be created.
VIF_CANNOTDELETE	The existing version of the file could not be deleted.
VIF_CANNOTRENAME	The temporary file could not be renamed to the name of the existing file. The existing file has already been deleted.

VIF_OUTOFMEMORY	Operation failed due to lack of memory.
VIF_CANNOTREADDISK	The source file could not be read.
VIF_CANNOTREADDEST	The existing destination file cannot be read (thus version information can not be checked).
VIF_BUFFTOOSMALL	The TmpFileLen parameter is too small to hold the name of the temporary file.

VerLanguageName

VB Declaration:

```
Declare Function VerLanguageName& Lib "version.dll" Alias "VerLanguageNameA" (ByVal wLang As Long, ByVal szLang As String, ByVal nSize As Long)
Declare Function VerLanguageName% Lib "ver.dll" (ByVal Lang%, ByVal lpszLang$, ByVal cbLang%)
```

Description: This function retrieves the name of a language based upon the 16 bit language code. Language codes are used in the version resource of a file to determine the language for which the file was written. The list of language codes supported by Windows (also located in this chapter).

Parameter	Type	Type/Description
Lang	Long (32) or Integer (16)	The language ID.
lpszLang	String	A buffer to load with the text name of the specified language. This buffer should be pre-allocated to at least cbLang+1 bytes.
cbLang	Long (32) or Integer (16)	The length of the lpszLang buffer.

Return Value: **Long (32) or Integer (16) :** The number of characters loaded into the lpszLang buffer, zero on error.

VerQueryValue

VB Declaration:

Declare Function VerQueryValue& Lib "version.dll" Alias "VerQueryValueA" (pBlock As Byte, ByVal lpSubBlock As String, lplpBuffer As Long, puLen As Long)

Declare Function VerQueryValue% Lib "ver.dll" (ByVal lpvBlock&, ByVal SubBlock$, lpBuffer&, lpcb%)

Description: This function is used to retrieve information from the version resource. Before calling this function, the version resource information must be retrieved using the **GetFileVersionInfo** structure. This function examines the resource information and copies the requested data into a buffer.

Parameter	Type	Description
lpvBlock	Long	The address of a block of memory containing version information data as retrieved by the **GetFileVersionInfo** function.
SubBlock	String	One of the following: "\" to retrieve the FIXEDFILEINFO structure for this file. "\VarFileInfo\Translation" to retrieve the translation table for this file. "\StringFileInfo\.... Refer to the Comments section for a full description.
lpBuffer&	Long	The address of the buffer to load with the requested version information. This should be pre-allocated to at least lpcb bytes long.
lpcb	Long (32) or Integer (16)	The length, in bytes, of the lpBuffer buffer.

Return Value Long (32) or Integer (16): True (non-zero) on success, zero if the requested information does not exist or lpvBlock does not reference valid version information.

Comments: When the **SubBlock** parameter is "\VarFileInfo\ Translation", the buffer will loaded with an integer array. Each pair of integers represents a language and code page pair describing available string information.

The StringFileInfo string data is retrieved by specifying a string with three parts as follows:

"\StringFileInfo*languagecodepagestringname*"**

languagecodepage is an 8 character hex number in string form. If the language codepage entry in the translation table is &H04090000, then this string should be "04090000 ".

stringname is a string name from those listed in the **StringFileInfo** table earlier in this chapter. An example of this parameter would be:

"\StringFileInfo\04090000\CompanyName"

The Windows Registry

The Windows Registry is a central depository containing a variety of program and configuration information. In this manual, the term "Registry" refers to two similar implementations: the Registration Database in Microsoft Windows 3.1 and the Registry in Windows NT and Windows 95/98. A thorough technical discussion of the Registry is beyond the scope of this manual, this section will focus on the key elements of the Registry that can affect the use of shared components, in particular ActiveX controls and components, with executables. Desaware also has a product available called "StorageTools" which includes a complete registry and registration database toolkit including ActiveX controls and extensive sample code.

Evolution of the Registry

One of the key features of Microsoft Windows is its ability to be customized to suit the user. Not only can a user specify color schemes or pictures on the desktop, but programs can remember where they were last placed and which documents were most recently used. Microsoft originally specified that Windows applications keep configuration information inside a text file called **WIN.INI**. This created a standard for programmers, while allowing applications and users to change program and system settings directly. As the number of Windows applications exploded, limitations in this standard became apparent. Initialization files became large and—because of their flat arrangement—hard to understand and edit. With the release of

Windows 3.0, Microsoft tried to alleviate the first problem by creating API functions for handling private .INI files. This, of course, created problems of its own, -- filling the user's \WINDOWS directory with countless .INI files and making it difficult for applications to communicate with each other.

The beginning of the solution was introduced in Windows 3.1. With the introduction of inter-process forms of communication such as Dynamic Data Exchange (DDE) and Object Linking & Embedding (OLE), it became vital to create a centralized database of program descriptions. The **Registration Database** is the result. It has a hierarchical structure, allowing complex data structures to be stored cleanly. It is in binary format for quicker access. In addition to containing descriptions of DDE and OLE entry points, it also contains associations between extensions and applications, user readable names for programs and files, and other useful information.

The primary elements of the hierarchical structure are **keys**. Keys can contain other keys. The key at the root of this structure is named **HKEY_CLASSES_ROOT**. Any key can have one **value** associated with it. This value is in the form of a string of characters.

Because the Registry is in binary format, it cannot simply be loaded into a text editor as is customary with initialization files. Microsoft provides a program called REGEDIT which allows the user to view and modify the Registration Database. Running REGEDIT with the command line flag "-v" turns on the advanced editing mode that reveals more information from the registration database.

However, the Registration Database was (and still is) hampered by the 64K size limit. Additionally, the Registration Database is limited to one file (REG.DAT), so it cannot be split when the size limit is reached. It also has a limited purpose, and could not replace the .INI file as an application configuration storage.

One of Microsoft's priorities in designing Windows NT was to overcome limitations discovered in 16 bit Windows. To solve the problems with initialization, Microsoft programmers created the **Registry**. At its heart, it is similar to the Registration Database—but much more useful. Instead of the lone HKEY_CLASSES_ROOT root key, the Registry has four root keys—of which HKEY_CLASSES_ROOT is one. These new keys provide the structure and the space for any configuration information a program might need to store. There is also space to put version information and other data a program might want to make public. The whole database can be of almost unlimited size. It is specially formatted and buffered to allow extremely fast access.

Keys can now hold any number of named values, and these values can be any of a number of different data types. The .INI files became obsolete—in fact, when 16 bit applications make API calls to read or write information to system .INI files, Windows NT actually re-routs them to a special location inside the Registry. The 32 bit version of the Registry editor is named REGEDT32.EXE (or RegEdit.exe) You may not be able to locate this program on your machine as system administrators do not always make it available to end users due to the possibility of accidental destruction of important Windows data.

Windows 95 and 98 also depend heavily upon the Registry, although not quite as much as Windows NT. Windows 95 and 98 do not need WIN.INI and SYSTEM.INI files, but will use them if they exist. Any attempt to change these initialization files will not result in a change in the Registry. Windows 95 and 98 adds two new root keys to the NT Registry while removing areas devoted to security and other Windows NT-only features. The Windows 95 and 98 Registry editor is named REGEDIT.EXE.

ActiveX Controls Must be Registered

Before you can use an ActiveX control, you must register it with the system. Registering an ActiveX control places information about the control such as version, path, and platform in the Registry. Once the control has been registered, applications and development environments can search the Registry to determine which controls have been installed. Unlike a Visual Basic custom control (VBX), an ActiveX control does not need to be registered by each application that uses it. Once the control has been registered with the system, any application can find it (until it is unregistered).

Registering the ActiveX Control

Microsoft provides the REGEDIT.EXE program which can register ActiveX components. Most executables that use ActiveX components will register all the components automatically upon installation. By definition, an ActiveX control must be "self-registering", that is, it can register and unregister itself.

Self-registration simplifies the process of registering a control because the control does the actual work of adding the necessary information to, or removing it from, the Registry. ActiveX controls export two functions, DllRegisterServer and DllUnregisterServer to handle self registering and unregistering. The ActiveX container need only call the DllRegisterServer function from the .OCX file to get the control to register itself. Most ActiveX containers also provide some method to "Browse" for objects. When an ActiveX component is selected, the container calls the exported function from the component file to initiate the self-register sequence. One method to determine whether a file is an ActiveX control is to check for the existence of the "OLESelfRegister" string in the file's StringFileInfo version section, however - this approach is not 100% reliable as it assumes that the control vendor has followed the standard. You can assume that any control that has this string set is self-registering, but you may find self-registering controls that do not have this string set. Refer to the VerInfo project for an example on retrieving this field.

Missing ActiveX Controls

You can delete an ActiveX control without unregistering it. Therefore, the Registry may contain entries for ActiveX controls that no longer exist on the system. VersionStamper confirms that registered ActiveX controls actually exist.

Unregistering ActiveX Controls

Care should be taken when unregistering an ActiveX control. If an ActiveX control is unregistered while it is in use, the control can stop functioning altogether in every application. In addition to verifying the required components when your application loads, you can also add a command to your application that invokes VersionStamper to verify the required components at any time. You can run this command if the ActiveX controls in your application starts behaving differently to confirm that all ActiveX controls are still registered.

Some Important Keys in the Registry

HKEY_CLASSES_ROOT: This is available in Windows 3.1, Windows NT, Windows 95 and 98. It is mapped to HKEY_LOCAL_MACHINE\Software\Classes in all but Windows 3.1., where it is the only root available. It is where file extensions are mapped to applications, DDE links are described, and OLE objects are defined. Keys in this root usually do not have any named values, and all values are plain strings in order to be compatible with the Windows 3.1 Registration Database. OLE objects are defined in the CLSID subkey.

HKEY_LOCAL_MACHINE\SOFTWARE\(Company Name)\(Program Name)\CurrentVersion: Version information is placed in this key. Some of the standardized value names you should use are:

Value	Description
Description	Text description of the software.
InstallDate	Long integer holding the date.
RegisteredOwner	The registered owner of the software.
MajorVersion	A string containing the most significant version number.
MinorVersion	A string containing the least significant version number.
ServiceName	A short human-readable name.
SoftwareType	The type of program (i.e. “driver”, “service”, “application”, etc.).
Title	The title of the program.

Any other version information should also be put here.

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\Current Version\SharedDLLs:

For use with the Windows 95 style shell only. In this key are the names and full paths of all .DLLs used by installed applications with reference counts of how many applications use each one. This is the method Windows 95 and 98 uses to decide if it can delete components of uninstalled programs. Make sure your install and uninstall programs correctly update this area.

HKEY_LOCAL_MACHINES\SOFTWARE\Microsoft\Windows\ CurrentVersion\AppPaths:

For use with the Windows 95 style shell only. This area is used to specify directories for applications and support files. Add a key named after your application here. Set the default value to the path where your .EXE files is located. If you need to, create a value named Path and set it to the path where your DLL's are located (if not in the same place as your .EXE or \Windows\System). The Windows shell automatically updates this area if you move the executable.

When creating a future update, you might not want to overwrite the previous Registry entry. Doing so might render the previous version of your software unusable. You can overcome this by setting up different areas with keys for different version of you software. For example, you could have a HKEY_LOCAL_MACHINE\Software\(Company Name)\(ProgramName)\1.0 key and a HKEY_LOCAL_MACHINE\ Software\(CompanyName)\(ProgramName)\2.0 key to store configuration information of both programs safely.

The best way to learn more about the Registry is to do some exploring with the Registry browser tools (e.g. REGEDIT that comes with Windows or REGBRSR that comes with StorageTools) that are available.

Other Sources of Information

Here are several other resources that we recommend for advance Windows development.

Dan Appleman's Visual Basic Programmer's Guide To The Win32 API

Written by Daniel Appleman (president of Desaware) and published by McMillan, (ISBN 0-672-31590-4) - this sequel to the original 16 bit API Guide applies the same philosophy to teaching the Win32 API to developers using Visual Basic and VBA based applications. With more examples, more functions, more tutorial style explanations and a full text searchable electronic edition on CD-ROM, this book should prove a worthy successor to the 16 bit API book. Covers Visual Basic version 4 through 6.

Available at most good bookstores, or directly from Desaware at a 20% discount - call (408) 404-4760 or email support@desaware.com.

An upgrade CD is available for owners of the "PC Magazine's Visual Basic Programmer's Guide to the Win32 API" ISBN: 1-56276-287-7 for $24.99 + s&h directly from Desaware. Refer to our web site at www.desaware.com for additional information.

Dan Appleman's Developing COM/ActiveX Components with Visual Basic 6.0: A Guide to the Perplexed

Written by Daniel Appleman (president of Desaware) and published by McMillan, (ISBN 1-56276-576-0) - this book is designed for those programmers interested in using Visual Basic's object oriented technology to develop ActiveX components including EXE and DLL servers, ActiveX controls and ActiveX documents. Unlike many books that simply rehash the Visual Basic documentation, this one serves as a commentary to clarify and extend the documentation. Of special interest to VersionStamper customers will be the chapters on OLE and COM technology that will help them further understand the process of registering components, and the chapters on versioning and licensing.

The VB6 version also includes two new chapters on IIS Application development.

Available at most good bookstores, or directly from Desaware at a 20% discount - call (408) 404-4760 or email support@desaware.com.

Dan Appleman's Win32 API Puzzle Book and Tutorial for Visual Basic Programmers

Written by Daniel Appleman (president of Desaware) and published by Apress (ISBN# 1-893115-01-1). Appleman's Win32 API Guide covers 700 API functions. This book covers the other 7800. How? By teaching you everything you need to know to read and understand the Microsoft C documentation and create correct API declarations for use in Visual Basic. Presented in an entertaining puzzle/solution format that challenges you to solve real world API problems. In depth tutorials take you behind the scenes to understand what really happens when you call an API function from VB.

The Desaware Visual Basic Bulletin

and other related technical articles. At the Desaware website: http://www.desaware.com.

PC Magazine's Visual Basic Programmer's Guide To The Windows API

Written by Daniel Appleman (president of Desaware) this book is intended to help Visual Basic programmers navigate the complexities of Windows. It is the only text on Windows that is designed specifically for Visual Basic programmers, and the only one that covers the interactions between Visual Basic and Windows.

Available on CD Rom only from Desaware. Call (408) 404-4760 or email support@desaware.com.

Windows API Online Help

The Professional Edition of Visual Basic includes Win31api.hlp and/or win32api.hlp - an online help reference for all API functions. These functions are declared in C and do not consider Visual Basic compatibility issues, however the information in chapter 3 of the Visual Basic Programmer's Guide to the Windows API (chapters 3 and 4 of the 32 bit book) will provide you with information on how to translate these functions to Visual Basic.

Microsoft's Developers Network CD Rom

This amazing CD-ROM contains a wealth of information and sample code, plus the latest Visual Basic knowledge base.

Microsoft's Windows Software Development Kit and Win32 Software Development Kit

The sample code is all in C, but by the time you've read the Visual Basic Programmer's Guide to the Windows API or Win32 API, you'll know enough to be able to translate the C code to Visual Basic.

Index

Desaware Product Descriptions

Thank you for your purchase of this Desaware product. We have additional quality software to enhance your programming efforts. Please visit our web site at www.desaware.com for detailed descriptions and product demos.

SPYWORKS Standard 6/Professional 7

SpyWorks in a nutshell? Impossible!

You're going to want to download the SpyWorks demo to even begin to understand its capabilities. This product has been evolving for several years, and it includes so many features it's hard to know where to begin. SpyWorks is a VB power tool. When you need to override VB's default behavior or to extend VB's functionality, you will want to use SpyWorks.

Do *That* in Visual Basic??

Want to put VB to the test? Want to learn advanced programming techniques? Want to keep the productivity of VB and have the functionality of C++? SpyWorks contains the low level tools that you need to take full advantage of Windows. Here are just a few of the features of this multi-faceted software package. For instance, have you ever wanted to detect keystrokes on a system-wide basis or detect when an event occurs in another application or thread using subclassing or hooks? SpyWorks can help you solve these problems by letting you tap into the full power of the Windows API without having to be an expert. SpyWorks lets you export functions from VB DLL's so that you can create function libraries, control panel applets, and NT Services. With its ActiveX extension technology, you can call and implement interfaces that VB5 or 6 do not support. SpyWorks includes the Desaware API Class Library, which assists programmers in taking advantage of the hundreds of functions that are built into the Windows API. SpyWorks is available in either the Professional (Pro) or Standard edition.

The Pro Edition includes a WinSock component with comprehensive VB source code that gives you complete control for Internet/intranet programming. Other features are the NT Service Toolkit *Light Edition*. This new application is a subset of the Desaware NT Service Toolkit product. It allows a developer to create true NT services using Visual Basic. A background thread component that allows you to easily create objects that run in a separate background thread.

It also contains extensive sample code and three product updates.

- The Professional Edition includes the Winsock Library, NT Service support and many other additional features & samples, plus three free updates. SpyWorks 2.1 (VBX Edition) is included in the Pro Edition.
- SpyWorks Standard is a subset of Professional. A feature comparison is available on our web site.
- Supports VB 4, 5 & 6, .NET (Pro), Windows 95, 98, NT, 2000 & XP.

VERSIONSTAMPER 6.5

Distributing Component-Based Applications? Beware DLL HELL!

You've distributed your application and it's working fine. But your end user is still in charge of their system. What happens when they install a program that overwrites a component that your software needs to run? Can you verify that your users have the correct files required by your application? Can you really afford to spend two hours on the phone trying to figure out exactly what went wrong? Now you can easily avoid component incompatibilities by adding VersionStamper to your toolkit. It lets you check the versions of your program's components on your end user's system, and correct the problem.

You are in control!

DLL Hell is a big problem, and with VersionStamper you can be in control of how this problem is detected and corrected. You determine dependency scanning (file size, date, version or other parameter), how and when the dependency scanning is done (upon start up, at midnight, at user's discretion), and how you want the problem resolved (automatically, an email message to your help desk, from a dependency list on your web site and more). This means you can handle versioning problems as simply as using a message box to call tech support, or even automatically updating the invalid components over the internet or corporate network. Imagine your application updating itself without user (or programmer) intervention! Imagine the hours and money saved in tech support calls! You can even use VersionStamper for incremental updates and bug fixes.

Is This For Real?

No, you don't have to pay a fortune in distribution fees - there are no run-time licensing fees. VersionStamper comes with a great deal of sample code. Don't distribute a component-based application without it!

- Checks the versions of your dependent files and notifies you or the user of potential problems.
- Internet extensions allow you to update versions across the Internet/intranets.
- Cool and USEFUL sample programs show you how it works.

Includes VB source code for the VersionStamper components that you can use in your applications.

NT SERVICE TOOLKIT 2.0

Create a fully featured service in minutes using Visual Basic – even debug your service using the Visual Basic environment! Supports all NT service options and controls. Adheres to all Visual Basic threading rules. Background thread support allows easy waiting on system and synchronization objects. Client requests supported on independent threads for excellent scalability, with client impersonation available allowing services to act on behalf of clients in their own security context. Client requests and service control possible via COM/COM+/DCOM.

Simulation mode for testing as an independent executable. Create control panel applets for service control and other purposes.

DESAWARE EVENT LOG TOOLKIT 1.0

Visual Basic allows you to log events to the NT/2000 event log, but does not allow you to create custom event sources - so every event belongs to the application VB runtime, descriptions are limited, and event categories unavailable. Even if you use the API to log events, creating custom event sources for your application is not supported by VB, and is difficult with C++.

Desaware's new Event Log Toolkit makes creation of event sources easy, and provides all the tools needed to create and log custom events. Now your applications and services can support event logs in a professional manner, as recommended by Microsoft

STORAGETOOLS ver 3.0

StorageTools is your key to the OLE 2.0 Structured Storage Technology. Structured Storage allows you to create files that organize complex data easily in a hierarchical system. It is like having an entire file system in each file. 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 also disregard the actual location of data in the file. Additionally, 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. In fact, we include programs (with Visual Basic source code) that let you examine the structure of any OLE 2.0 based file so that you can see exactly how they do it!

StorageTools includes documentation and controls to make it easy to work with the registration database under Windows 3.1, Windows NT & Windows 95/98 and 2000. For Visual Basic 4-6. We also include a simple resource compiler (with Visual Basic source code) so that you can create your own .RES files for use with Visual Basic.

StorageTools version 3.0 also includes the Desaware File Property component and .NET support.

StorageTools includes 16 & 32 bit ActiveX/ATL controls, extensive documentation and sample code.

DESAWARE ACTIVEX GALLIMAUFRY Ver. 2

What is it?

gal·li·mau·fry (gàl´e-mô¹frê) noun
plural gal·li·mau·fries
A jumble; a hodgepodge.

[French galimafrée, from Old French galimafree, sauce, ragout : probably galer, to make merry. See GALLANT + mafrer, to gorge oneself (from Middle Dutch moffelen, to open one's mouth wide, of imitative origin).]
(From The American Heritage® Dictionary of the English Language, Third Edition copyright © 1992 by Houghton Mifflin Company)

What does a Twain control, spiral art program, set of linked list classes, a quick sort routine, a hex editor and a myriad of other custom controls have in
common?

They are all part of Desaware's ActiveX Gallimaufry.

You'll find most of these controls useful, the rest entertaining – but we guarantee that you'll find them all educational, because they come with complete Visual Basic 6.0 source code.

Curious?

Want to learn some advanced API programming techniques? Visit our web site for a full description and demo.

THE CUSTOM CONTROL FACTORY V 4.0
The Custom Control Factory is a powerful tool for creating your own animated buttons, multiple state buttons, toolbars and enhanced button style controls in Visual Basic and other OLE control clients, without programming. With 256 & 24 bit color support, automatic 3D backgrounds, image compression, over 50 sample controls and more. Plus MList2 - an enhanced listbox control. 16 & 32 bit ActiveX controls and 16 bit VBXs included.