CMP
United Business Media

Utilities

Book Review: **Practical C++ Programming**

AUG
2003

windows:: developer

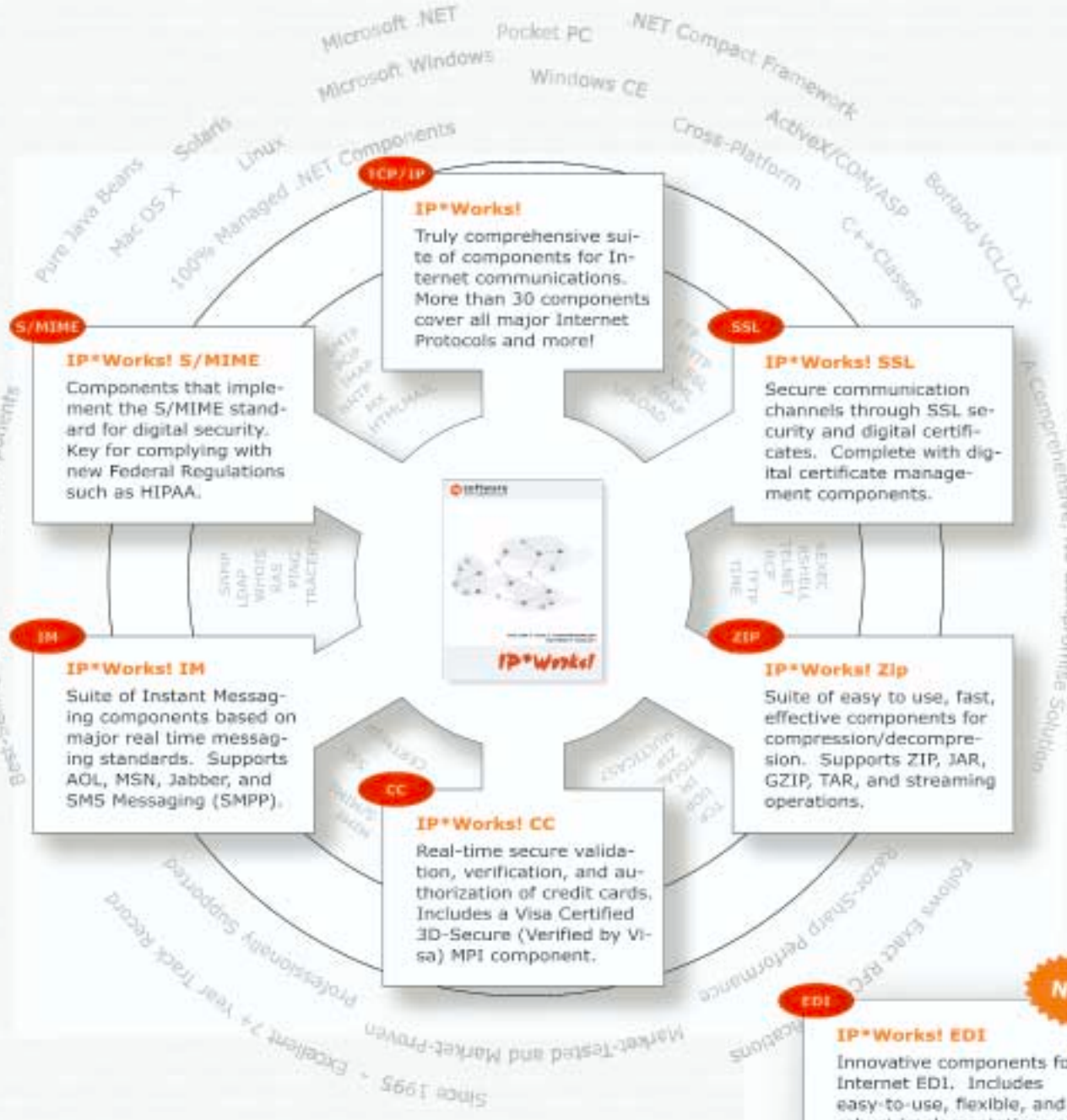
APPLICATION DEVELOPMENT FROM WINDOWS TO WEB NETWORK

Volume 2 / No. 8

www.windevnet.com

Remote Reboot Shell Extension

from email to secure credit card transactions, only one component suite does it all!



/n software

Name:	Daiva Venckus
Application:	Ringster
Interest:	The next big thing – and being a part of it
Income:	Millions of potential customers – you do the math

Daiva Venckus is a senior product manager with Moviso LLC, a subsidiary of Vivendi Universal. She's the creator of Ringster, a cool BREW™ application that allows for the downloading of over a thousand ringtones from leading entertainment companies. "The most attractive thing about BREW is that even the smallest developers can now get their product to the carriers for download," said Venckus. "And those BREW subscribers number in the multi-millions – this is a very real revenue opportunity." Other developers agree. Commercial services are launched and BREW applications such as games, email, news, weather, stock trades, position location and ringers are now in the market – a market of millions upon millions of customers. If you aren't developing for BREW, you aren't developing to your potential. To get started, go to www.qualcomm.com/brew/wdm.

brew™
Customize. Personalize. Realize.™

GAIN THE ADVANTAGE

For over 16 years, SD has delivered reliable, trustworthy, and practical developer training to thousands of developers and managers seeking the best education in C++, Java, XML, Windows, Web Services, modeling, and development process. From the producers of SD, comes a new East Coast conference - SD Best Practices.

CONFERENCE TRACKS

- Requirements and Analysis
- Testing and Quality
- Build and Deploy
- Design and Architecture
- Process and Methods
- People, Projects and Teams

YOU WILL GAIN

- Practical skills on how to incorporate best practices, quality design, and proven management techniques into your software applications
- The benefits of modeling, requirements, analysis, and testing
- A deep understanding of Agile, XP and other key methodologies
- Management tools to help you increase team productivity

EARLY BIRD DISCOUNT
Register by August 15 and save up to $300

SD BEST 2003 PRACTICES
CONFERENCE & EXPO

A New Conference from the **SD Events Group**

September 15-18, 2003, Boston, MA

Register today at www.sdexpo.com

windows::developer
APPLICATION DEVELOPMENT FROM WINDOWS TO WEB NETWORK

PUBLISHER Kerry Gates
EDITOR IN CHIEF John Dorsey

BPA

EDITORIAL

MANAGING EDITOR **Amy Stephens**

CONTRIBUTING EDITORS **Jeff Claar, Dino Esposito, George Frazier, Richard Grimes, Petter Hesselberg, Paula Tomlinson, Victor R. Volkman**

EDITORIAL ADVISORY BOARD **Mark Baker, Jeff Claar, Dino Esposito, George Frazier, Richard Grimes, Petter Hesselberg, Mark Nelson, Mark Russinovich, Paula Tomlinson, Victor R. Volkman**

ASSOCIATE EDITOR **Della Song**

ART DIRECTOR **Beatriz Américo**

WEBMASTER **Joe Lucca**

SEND READER MAIL TO: **wdletter@cmp.com**

SUBSCRIPTION INQUIRIES: **wdnetwork@halldata.com**

ADVERTISING AND MARKETING

DIRECTOR OF SALES **David Timmons**

REGIONAL MANAGER, EAST
Jon Hampson 603-924-8500 *jhampson@cmp.com*

REGIONAL MANAGER, CENTRAL/SOUTHEAST
Ed Day 785-838-7547 *eday@cmp.com*

REGIONAL MANAGER, WEST
Michele Hurabiell 415-947-6199 *mhurabiell@cmp.com*

ACCOUNT MANAGER, ALL REGIONS
Julie Thibault 603-924-8400 *jthibault@cmp.com*

PRODUCTION COORDINATOR
Michael Penne *mpenne@cmp.com*

DIRECTOR OF MARKETING **Karen Tom**

CIRCULATION

SENIOR CIRCULATION MANAGER **Cherilyn Olmsted**

ASSISTANT CIRCULATION MANAGER **Gwen Olson**

SUBSCRIPTIONS: Annual renewable print subscriptions to *Windows Developer Network* are $34.99 U.S., $45 Canada and Mexico, $65 elsewhere. Payments must be made in U.S. dollars. Make checks payable to *Windows Developer Network*.

CUSTOMER SERVICE: For subscription orders and questions, contact lawrencecs@cmp.com.

ADVERTISING: For rate cards or other information on placing advertising in *Windows Developer Network*, contact the advertising department at 785-841-1631, or write *Windows Developer Network*, 1601 W. 23rd St., Suite 200, Lawrence, KS 66046-2700 USA.

Windows Developer Network (ISSN 1543-6462) is published monthly by CMP Media LLC, 600 Harrison St., San Francisco, CA 94107 USA, 415-947-6000.

CMP MEDIA LLC

CORPORATE
PRESIDENT AND CEO **Gary Marshall**
EXECUTIVE VICE PRESIDENT AND CFO **John Day**
EXECUTIVE VICE PRESIDENT AND COO **Steve Weitzner**
EXECUTIVE V.P., CORPORATE SALES AND MARKETING **Jeff Patterson**
CHIEF INFORMATION OFFICER **Mike Mikos**
SENIOR V.P., OPERATIONS **William Amstutz**
SENIOR V.P., H.R. AND COMMUNICATIONS **Leah Landro**
VICE PRESIDENT AND GENERAL COUNSEL **Sandra Grayson**
MARKET
PRESIDENT, GROUP PUBLISHER TECHNOLOGY SOLUTIONS **Robert Faletra**
PRESIDENT, GROUP PUBLISHER HEALTHCARE MEDIA **Vicki Masseria**
V.P., GROUP PUBLISHER APPLIED TECHNOLOGIES **Philip Chapnick**
V.P., GROUP PUBLISHER INFORMATION TECHNOLOGY **Michael Friedenberg**
V.P., GROUP PUBLISHER ELECTRONICS **Paul Miller**
V.P., GROUP PUBLISHER NETWORK TECHNOLOGY **Fritz Nelson**
V.P., GROUP PUBLISHER SOFTWARE DEVELOPMENT MEDIA **Peter Westerman**
CORPORATE DIRECTOR, AUDIENCE DEVELOPMENT **Shannon Aronson**
CORPORATE DIRECTOR, AUDIENCE DEVELOPMENT **Michael Zane**

CMP
United Business Media

CONTENTS

AUG 2003

windows::developer NETWORK VOL 2 • NO. 8

FEATURES

COLUMNS

VISIT US ONLINE: www.windevnet.com

DEPARTMENTS

PDF EXTRAS

Download the PDF version of this month's issue to access bonus features. This content includes:

- Additional listings and figures for "Remote Reboot Shell Extension" plus a sidebar on WTL & ATL

| Download code › **windevnet.com/wdn/code/** |

COMING NEXT MONTH: **SECURITY**

FROM THE EDITOR

BORLAND HAS BEEN CONTINUING its efforts to be the Switzerland of development tools. This summer it released several new products promoting both the .NET Framework as well as Java 2 Enterprise Edition. In June, Borland was on hand at TechEd in Dallas demonstrating its new C# Builder IDE (formerly codenamed "SideWinder"). John Kaster, Borland's Senior Developer Relations Manager, gave two talks on building database apps with C# Builder and SQL Server. C# Builder will be bundled with developer licenses for SQL Server as well as Borland's own InterBase and IBM's DB2. The IDE's interface should be familiar to anyone who has used C++ Builder.

In the week after TechEd, the Borland team was in San Francisco at JavaOne promoting Janeva, its new bridging technology for connecting .NET front ends to CORBA apps and J2EE-based servers. Janeva incorporates into C# Builder or VS.NET to let developers generate C# stubs and assemblies. Janeva translates between the .NET data types and Java data types. No modifications are needed on the server side, and Janeva-enabled client apps do not require a JVM to access the J2EE server.

Boz Elloy, Vice President and General Manager of Enterprise Solutions, discussed the genesis of Janeva at a JavaOne roundtable discussion on Java and .NET interop. "We're seeing a lot of demands from all the J2EE customers that still might get a significant investment in their back-end systems. And what I want to do is to leverage the strength of Microsoft .NET with the high fidelity UI client."

Of course, web services are designed to accomplish this kind of cross- platform interop, but if the server components do not provide a web service interface, they would need to be modified to do so. Janeva provides a solution that requires no modification to the back-end components.

In addition, security strategies are still evolving for web services, and the overhead of converting data to and from plain text is also a concern in large-scale J2EE systems. According to Elloy, the transfer of data in a Janeva-based solution would be considerably faster.

Janeva is free for developers to download and develop with, but licensing fees are required for deployed apps. For more information, see http://www.borland.com/janeva/.

Borland's support to the .NET Framework will continue to grow, too—it has been providing previews of Delphi for .NET to its Delphi 7 users, and Delphi for .NET will probably be timed to launch at the next BorCon in November.

John Dorsey
Editor in Chief
wdeditor@cmp.com

windows:: developer NETWORK
APPLICATION DEVELOPMENT FROM WINDOWS TO WEB

CD-ROM Version 6.0 On Sale Now!

The *Windows Developer* CD-ROM gives you all editorial content and source code from **December 1991 through December 2002 issues** along with instant access to all installments of:

- George Frazier's *Tech Tips*
- Jeff Claar's *Bug ++ of the Month*
- Petter Hesselberg's *UI Programming*
- Victor Volkman's *Books in Brief*
- Paul Kimmel's *Frameworks*
- and much more!

Order Online!

www.windevnet.com/wdn/cdrom/

for only $29.95*

Discount Key Code: WDCDROM

For eleven years *Windows Developer* has been the independent technical magazine for Windows programmers covering articles ranging from quick technical tips to ready to use solutions for complex programming problems.

* *Windows Developer* CD-ROM is only $29.95 plus shipping and handling ($4 U.S. and Canada, $10 all other countries) for Internet orders only. Extra charges such as multiple copy orders, and delivery outside the U.S. will apply; exact charges will appear when online order is placed. CD-ROM orders placed by phone (800-444-4881) or mail (*Windows Developer* CD-ROM, 1601 West 23rd Street, Ste. 200, Lawrence, KS 66046, USA) are $39.95 plus shipping and handling. Mail orders accepted for U.S. delivery only. Credit cards accepted on Internet and phone orders only.

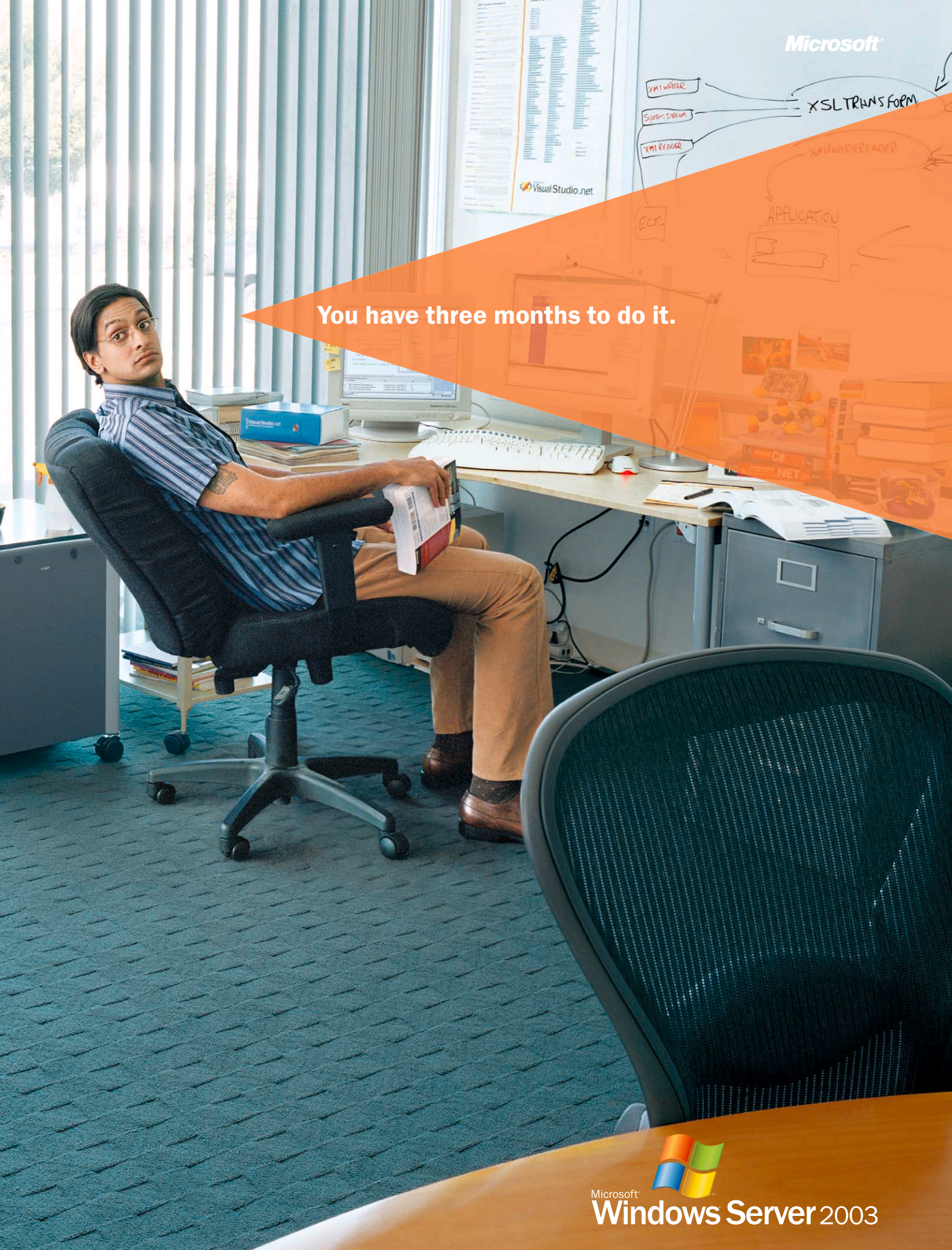
www.windevnet.com/wdn/cdrom/

You've got a 12-month development project.

Introducing Microsoft Windows Server 2003. *Do more with less.*

Microsoft® Windows® Server 2003 and the new Visual Studio® .NET 2003 provide you with the most productive application platform for building, deploying, and managing connected applications. Everything from design through deployment is now more efficient. Because the .NET Framework is tightly integrated into both products, you're freed from writing plumbing code so you can focus on delivering real business value. You also get simpler and faster deployment (thanks to the elimination of DLL version conflicts). Try this new application platform for yourself at msdn.microsoft.com/windowsserver2003/tryit **Software for the Agile Business.**

*The **Middleware Company** compared the .NET Pet Shop Web application on Windows Server 2003 to the performance and scalability of a comparable, optimized J2EE™ application. The .NET connected application on Windows Server 2003 is more than 250% faster, 76% less expensive based on price/performance, and required 11,000 fewer lines of code.*


Microsoft
XSLTRANSFORM
APPLICATION
Visual Studio .net
You have three months to do it.
Microsoft
Windows Server 2003

Remote Reboot Shell Extension

Shutdown or reboot a remote host via an Explorer shortcut

AS SOMEONE WITH MORE machines than monitors, I use an active keyboard/screen/mouse switcher. I'm also known to roam about the home with my laptop when the ambient noise from the kids exceeds a tolerable threshold, plugging into the various ports I've had installed. Whether you work in similar conditions or operate in a large distributed office, it can be a bit of a pain when you need to shutdown or reboot another machine. Rather than having to switch to an active monitor, or log on to pcAnywhere, or walk across the office, wouldn't a simple mouse click be nicer? (Of course, physiologists would say that a walk across the office is exactly what is needed to forestall our drooping statures, but that's another issue.)

A few years ago, I wrote a control panel applet that allows one to reboot/shutdown remote hosts, and that served its purpose nicely. However, since I am a big fan of shell extensions—and provide a number of them for free at http://shellext.com/—I thought it might be nice to write one that provides shutdown/reboot of a remote host via an Explorer shortcut. This article will describe the main technical aspects of the solution—the Remote Reboot context menu handler shell extension—and highlight some issues one must consider when creating such shell extensions. The implementation is largely ATL, with various STLSoft (my project for bringing STL to the masses, located at http://stlsoft.org/) and WTL (see the online sidebar "WTL & ATL") components thrown in for good measure. The finished component is available for free along with the other Synesis Software Shell Extensions (from http://shellext.com/) from Version 1.5.1 onwards.

Rebooting a Remote Host

Rebooting a network server is pretty straightforward. You call the Win32 function `InitiateSystemShutdown()`, passing the host name, a timeout, and specifying whether to reboot rather than shutdown, and whether to force application closure. You also pass a message string that will be displayed on the remote host during the period between the start of the shutdown and the machine actually shutting down, as can be seen in Figure 1.

There are two issues we must face when using this function. First, `InitiateSystemShutdown()` is only supported on NT-family (NT4, 2000, XP) systems; on 95-family (95, 98, Me) systems it simply returns a failure code. Second, you must have appropriate permissions to affect the shutdown. Specifically, you need the SE_REMOTE_SHUTDOWN_NAME privilege on any remote hosts that you wish to close. If you don't have this, then the Remote Reboot shell extension is not going to work for you; it will report an Access Denied message box. Since the rights and privileges associated with your logon identity are written into your user token at log on, and do not change during the course of your user session, any changes made on your behalf by the system administrators will require you to log off. (In such circumstances it'll probably be less hassle to walk over to the machine and reboot it manually.) For those who administer their own systems, the remote shutdown right is set by adding the requisite user/group to the "Force shutdown from a remote system" right as shown in Figure 2, which shows an NT 4 server dialog.

Context Menu Handler Shell Extensions

Context menu handler shell extensions, like all active shell extensions, are in-process COM servers that implement certain interfaces and provide, upon registration, certain registry entries. When the user right-clicks on one or more items within Explorer, on the desktop, or within standard File dialogs (`GetOpenFileName()`, `GetSaveFileName()`), the registry entries for the particular file type(s) are consulted and the appropriate context menu handler shell extensions loaded and initialized. For example, when right-clicking on the file "kernel32.dll," the registry will be searched for at least the keys HKEY_CLASSES_ROOT\dllfile\shellex\ContextMenuHandlers and HKEY_CLASSES_ROOT*\shellex\ContextMenuHandlers. If either of these keys exist, and have subkeys, the GUIDs in the default values of the subkeys represent the CLSIDs of the context menu handlers to be loaded and activated.

For network server shortcuts, the requisite key is NetServer, so your context menu handler must install under HKEY_CLASSES_ROOT\NetServer\shellex\ContextMenuHandlers. At this point it is worth noting that we are talking about right-clicking on shortcuts to network servers. The extension here does not operate on the items within "Network Neighborhood"/"My Network Places," which I presume is another kind of shell extension—a namespace extension. (And that's another story…)

MATTHEW WILSON *holds a degree in Information Technology and a Ph.D. in Electrical Engineering, and is a software-development consultant for Synesis Software. Matthew's work interests are in writing bulletproof real-time, GUI, and software-analysis software in C, C++, C#, and Java. He has been working with C++ for over 10 years, and is currently bringing STLSoft.org and its offshoots into the public domain. Matthew can be contacted via matthew@synesis.com.au or at http://stlsoft.org/.*

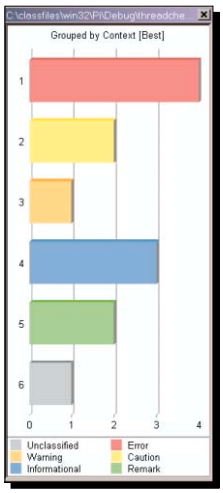
Introducing a Powerful New Debugging Tool for Threaded Applications

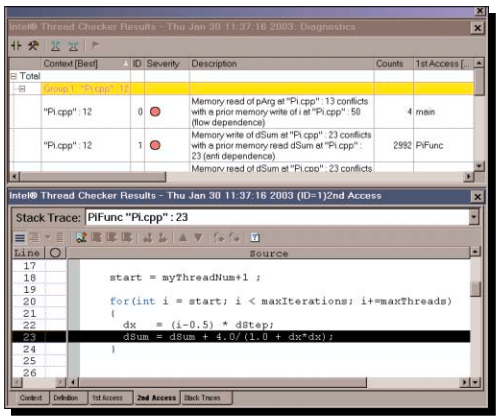
Intel® Thread Checker 1.0 for Windows*

Intel® Thread Checker 1.0 for Windows* helps you debug Win32* and OpenMP* threaded applications. The Intel® Thread Checker uses an advanced error detection engine to find bugs that might otherwise go undetected in your QA process. Quickly find threading bugs that would take days or weeks to find using traditional tools and methods!

If you work with threaded code, you should download a free trial. When you purchase Intel® Thread Checker you also receive the Intel® VTune™ Performance Analyzer.

Error Classification *Identifies 6 levels of threading issues from errors and warnings to informative comments*





Diagnostics View *Lists specific information on each error—race conditions, stalled threads, deadlocks and more…*

Source Code View *Clicking in diagnostics view takes you directly to the specific source code line.*

Intel® VTune™ Performance Analyzer 7.0

The award-winning VTune™ Performance Analyzer helps you improve your application performance by enabling you to locate and remove bottlenecks in your code. Features like the Intel Tuning Assistant give detailed guidence on tuning your code.

> ***"Using Intel Thread Checker we discovered two elusive bugs on the very first day..."***
>
> **—Farzin Shakib**
> *President ACUSIM Software, Inc.*

YOU SAVE UP TO $272!	Paradise #	Retail	Discount
Intel® Thread Checker 1.0 for Windows (includes VTune™ Performance Analyzer)	**I23 0A3A**	$1,198.00	**$925.99**
Intel® VTune™ Performance Analyzer v7.0	**I23 0A2N**	$699.00	**$539.99**

Download a Free Trial at:
programmersparadise.com/intel/wdj

Programmer's Paradise®
To order or request additional information call:
800-423-9990
Email: intel@programmers.com

03CT08F

Figure 1
Shutdown notification dialog

Figure 2
Enabling SE_REMOTE_SHUTDOWN_NAME

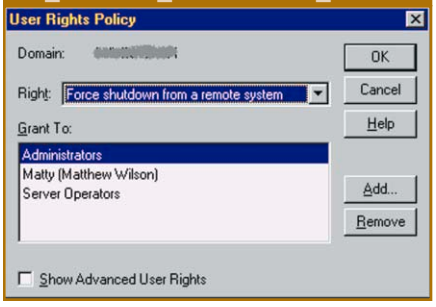
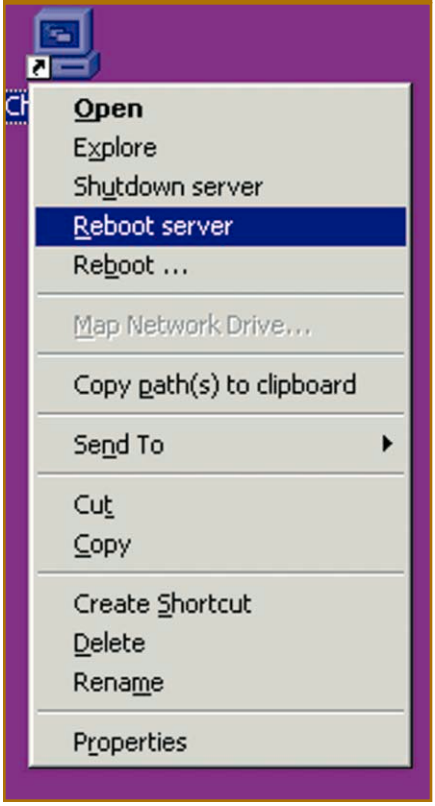


Figure 3
Remote reboot context menu items



Listing 1
Context menu handler shell extension

```
interface IShellExtInit
  : public IUnknown
{
  STDMETHOD(Initialize)(LPCITEMIDLIST pidlFolder,
                        LPDATAOBJECT  lpdobj,
                        HKEY          hkeyProgID) = 0;
};

interface IContextMenu
  : public IUnknown
{
  STDMETHOD(QueryContextMenu)(HMENU hmenu,
                              UINT  indexMenu,
                              UINT  idCmdFirst,
                              UINT  idCmdLast,
                              UINT  uFlags) = 0;
  STDMETHOD(InvokeCommand)(   LPCMINVOKECOMMANDINFO lpici) = 0;
  STDMETHOD(GetCommandString)(UINT  idCmd,
                              UINT  uType,
                              UINT  * pwReserved,
                              LPSTR pszName,
                              UINT  cchMax) = 0;
};
```

Listing 5 Implementing IShellExt::Initialize()

```
STDMETHODIMP CRemoteReboot::Initialize( LPCITEMIDLIST   /* pidlFolder */,
                                        LPDATAOBJECT    lpdobj,
                                        HKEY            /* hkeyProgID*/)
{
  HRESULT hr;

  if(!system_version::winnt())
  {
    // Can only use remote shutdown functions from NT family machines.
    hr = E_FAIL;
  }
  else
  {
    SynesisAtl::DumpFormats(lpdobj);

    // Determine whether there is a single selection

    HWND  hwndFocus = ::GetFocus();
    int   cSelections = ListView_GetSelectedCount(hwndFocus);

    if(cSelections != 1)
    {
      hr = E_FAIL;
    }
    else
    {
      // Retrieve the file-name
      FORMATETC fe;
      STGMEDIUM sm;

      fe.cfFormat = ::RegisterClipboardFormat(TEXT("Shell IDList Array"));
      fe.ptd      = NULL;
      fe.dwAspect = DVASPECT_CONTENT;
      fe.lindex   = -1;
      fe.tymed    = TYMED_HGLOBAL;

      hr = lpdobj->GetData(&fe, &sm);

      if(SUCCEEDED(hr))
      {
        // CF_IDLIST handling
#define HIDA_GetPIDLFolder(pida)    (LPCITEMIDLIST)(((LPBYTE)pida)+(pida)->aoffset[0])
#define HIDA_GetPIDLItem(pida, i)   (LPCITEMIDLIST)(((LPBYTE)pida)+(pida)->aoffset[i + 1])

        LPIDA          pida        = (LPIDA)sm.hGlobal;
        LPCITEMIDLIST pidlFolder  = HIDA_GetPIDLFolder(pida);
        UINT           cItems      = pida->cidl;
        LPCITEMIDLIST pidlItem0   = HIDA_GetPIDLItem(pida, 0);

        IShellFolder  *pdesktop;

        hr  = ::SHGetDesktopFolder(&pdesktop);

        if(SUCCEEDED(hr))
        {
          struct
            : public NETRESOURCE
          {
            BYTE  bytes[1024];
          } nr;

          ZeroMemory(&nr, sizeof(nr));
```

Programmer's Paradise®

Your best source for software development tools!

GUARANTEED BEST PRICES*

Should you see one of these products listed at a lower price in another ad in this magazine, CALL US! **We'll beat the price**, and still offer our same quality service and support!

*Terms of the offer:

- Offer good through August 31, 2003
- Applicable to pricing on current versions of software listed
- August issue prices only
- Offer does not apply towards obvious errors in competitors' ads
- Subject to same terms and conditions

PROGRAMMER'S PARADISE GUARANTEED BEST PRICES

Prices subject to change. Not responsible for typographical errors.

xmlspy® 5

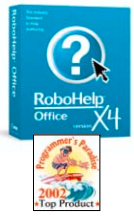
by Altova

xmlspy® 5 is the industry standard XML Development Environment for designing, editing and debugging enterprise-class applications involving XML, XML Schema, XSL/ XSLT, SOAP, WSDL and Web Services technologies. It is the ultimate productivity enhancer for J2EE, .NET and database developers.

Enterprise Edition
Paradise #
IOD 0166
$965.99

Professional Edition
Paradise #
IOD 0160
$389.99

www.programmersparadise.com/altova



Programmer's Paradise #1 Best-Selling Help Authoring Tool for 7 Years Running!

RoboHelp Office

The Industry Standard in Help Authoring

Create professional Help systems for desktop and Web-based applications, including .NET.

- Create all popular Help formats
- Create standard and advanced Help-specific features
- Work in WYSIWYG or true code
- Easily create context-sensitive Help
- Generate printed documentation
- Winner of 55 industry awards

** Price after manufacturer's mail-in rebate. Limited time offer; expires 8/31/03.*

Paradise #
E75 0311
$959.99*

www.programmersparadise.com/ehelp

LEADTOOLS Document Imaging

by LEAD Technologies

Document imaging including annotations, specialized bitonal (b/w) image display and processing like scale-to-gray and favor-black, performance and memory optimizations for bitonal images, image clean-up like hole-punch, line and staple removal, high-speed scanning.

Paradise #
L05 048V
$1,639.99

www.programmersparadise.com/lead

Intel® C++ and Fortran Compilers

by Intel

Increase the Performance of Your Application with Intel's High-Performance Compilers

Version 7.0

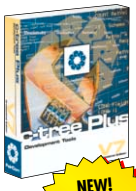
Intel's expertise in processors shows in this latest release of its flagship compiler product-line: Version 7 of its C++ and Fortran Compilers for Windows and Linux. Intel Compilers deliver outstanding performance on Pentium® 4 and Intel® Xeon® processors, and the 64-bit Itanium and Itanium 2 processors and take advantage of Multi-processor systems and Hyper-Threading technology.

Intel C++ for Windows
Paradise #
I23 0A10
$308.99

www.programmersparadise.com/intel

c-tree Plus®

by FairCom



NEW! SQL Support!

With unparalleled performance and sophistication, c-tree Plus gives developers absolute control over their data management needs. Commercial developers use c-tree Plus for a wide variety of embedded, vertical market, and enterprise-wide database applications. Use any one or a combination of our flexible APIs including low-level and ISAM C APIs, simplified C and C++ database APIs, SQL, ODBC, or JDBC. c-tree Plus can be used to develop single-user and multi-user non-server applications or client-side application for FairCom's robust database server—the c-tree Server. Windows to Mac to Unix all in one package.

Paradise #
F01 0131
$850.99

www.programmersparadise.com/faircom

TX Text Control ActiveX 10.0

by The Imaging Source

Add RTF, DOC, HTML, CSS and PDF Support to Your Application

NEW .NET VERSION!

TX Text Control is royalty-free, robust and powerful word processing software in reusable component form. The new Enterprise/XML version features a rich set of properties for the manipulation of XML and CSS. Developers can now offer end-users the ability to separate their textual content from their formatting rules.

Download a demo today.

DEMO AVAILABLE
programmersparadise.com

Enterprise Edition
Paradise #
T79 0214
$1,495.99

Professional Edition
Paradise #
T79 0215
$729.99

www.programmersparadise.com/theimagingsource

DataDirect Connect *for* .NET

by DataDirect Technologies

Use the right .NET data provider now and avoid reprogramming later. With DataDirect Connect *for* .NET Developer's License you'll get secure, high-performance data connectivity to Oracle, Sybase, DB2 and SQL Server. Our providers are 100% managed code, running entirely within the .NET CLR for better performance and fewer security risks.

Paradise #
DB1 013R
$197.99

www.programmersparadise.com/datadirect

PR-Tracker™ v5.1

by Softwise Company

Affordable scalable enterprise level bug tracking system featuring classification, assignment, sorting, searching, reporting, access control, user permissions, attachments and email notification. Integrates with PR-Tracker Web Client (included) and ProblemReport.asp (included for your betatest or customer support interface). Supports Access and SQL Server.

Download Today!

DOWNLOAD NOW!
programmersparadise.com

Paradise #
S3R 0147
$117.99

www.programmersparadise.com/softwise

Paradise Picks

DevTrack 5.5

Powerful Defect and Project Tracking

by TechExcel

DevTrack, the market-leading defect and project tracking solution, comprehensively manages and automates your software development processes. DevTrack 5.5 features sophisticated workflow and process automation, seamless source code control integration with VSS, Perforce and ClearCase, QA test plan management, robust searching, and built-in reports and analysis. Intuitive administration and integration reduces the cost of deployment and maintenance.

programmersparadise.com/techexcel

Paradise #
T34 0199
$482.99

Sun™ ONE Studio 7 Enterprise Edition Solaris

by Sun Microsystems

Sun™ ONE Studio 7, Enterprise Edition for Solaris is a productive environment for developing reliable, scalable, high-performance applications in the C, C++, Fortran, and Java languages for the Solaris Operating Environment. Sun ONE Studio 7, Enterprise Edition for Solaris software is a bundle that includes the Sun ONE Studio 7, Compiler Collection and the Sun ONE Studio 4, Enterprise Edition for Java products.

www.programmersparadise.com/sunone

Paradise #
S69 0U58
$2,890.99

Listing 5 Continued

```
IShellFolder  *pFolder;

          hr = pdesktop->BindToObject(pidlFolder, NULL,
                         IID_IShellFolder, (void**)&pFolder);

          if(SUCCEEDED(hr))
          {
            hr = ::SHGetDataFromIDList(pFolder, pidlItem0,
                           SHGDFIL_NETRESOURCE, &nr, sizeof(nr));

            if(SUCCEEDED(hr))
            {
              LPCTSTR host  = nr.lpRemoteName;

              // Elide the \\ prefix
              if( host[0] == '\\' &&
                  host[1] == '\\')
              {
                host += 2;
              }

              lstrcpyn(m_szHost, host, stlsoft_num_elements(m_szHost));
            }

            pFolder->Release();
          }

          pdesktop->Release();
        }

        ReleaseStgMedium(&sm);
      }

      ATLTRACE( _T("CRemoteReboot::Initialize() %s: [%s]\n"),
                SUCCEEDED(hr) ? _T("succeeded") : _T("failed"),
                m_szHost);
    }
  }

  return hr;
}
```


A context menu handler will, at minimum, support the interfaces `IContextMenu` and `IShellExtInit`. (There are additional interfaces `IContextMenu2` and `IContextMenu3` that help with custom drawing of the menu items.) The definitions of these interfaces (from ShlObj.h) are shown in Listing 1.

IShellExtInit::Initialize()

`IShellExtInit::Initialize()` is implemented by various shell extension types: property sheet handlers, drag-and-drop handlers, and context menu handlers, which are what we'll be talking about here.

Upon initialization, the shell extension is passed data from the shell via the `IDataObject` instance passed as the second parameter in `IShellExtInit::Initialize()`. Most of the shell extensions I've previously written have operated on filesystem types such that the shell provided data formats (in the `IDataObject` instance) that included the clipboard format CF_HDROP, which denotes a drop handle (HDROP), along with the shell-specific formats of Shell IDList Array, FileName, and FileNameW. The latter three are custom clipboard formats registered by the shell with the Win32 function `RegisterClipboardFormat()`. To use them you should call `RegisterClipboardFormat()` yourself, which will return a UINT representing the system-wide ID of the format. If it is not already registered, your call will register it (though this is unlikely since the shell itself registers them at startup).

Since working with a drop handle is very straightforward, this has been my preferred approach until now. A drop handle is an opaque handle that represents a system-managed set of paths. It is retrieved from an `IDataObject` instance using code such as that shown in Listing 2 (available online), which shows the implementation of a helper function I use for this purpose. (It resides in a C file, but is written to be compatible with C++ compilation if included into a C++ project file.) The function populates a FORMATETC structure describing the type of data required (CF_HDROP) and how it is to be received (TYMED_HGLOBAL), and passes a STGMEDIUM structure in which the data will be written.

Once you have a drop handle, you can access the paths it represents by calling `DragQueryFile()`, which places a specific path according to a given index into a caller-supplied buffer, or returns the number of files for the sentinel index value 0xFFFFFFFF. Once you've finished with the handle, you must call `DragFinish()` to release the resources.

For those who are comfortable with STL, the use of the handle can be simplified by using WinSTL's basic_drophandle_sequence class, as shown in Listing 3 (available online). (WinSTL is the

Win32-related subproject of STLSoft, located at http://winstl.org/.)

Unfortunately, the CF_HDROP format is not provided for all filesystem types. The NetServer type, which is what we need to use for network servers, does not come with CF_HDROP. The MSDN documentation provides very little help on this (try a search for "+NetServer +shell"), so I resorted to some practical measures. `IDataObject` instances are able to report all their accessible formats in a COM enumerator implementing `IEnumFORMATETC`, which is retrieved via the `EnumFormatEtc()` method. Hence, I used a helper function `DumpFormats()` (Listing 4, available online) to trace all the formats for the data object. As you can see, the function uses a COMSTL template, `enum_simple_sequence`, which provides a parameterized mapping from a COM enumerator (something implementing one of the `IEnumXXXX` interfaces) to an STL-compliant sequence providing Input or Forward Iterator semantics. (COMSTL is the STLSoft subproject pertaining to COM, located at http://comstl.org/.) Combining this with the trace_FORMATETC function object provides a neat and simple mechanism of tracing the supported clipboard formats for our `IDataObject` instance.

Once I plugged this in, I was able to determine that the NetServer shell type provides the registered clipboard formats Shell IDList Array, FileName, FileNameW, and Net Resource. (Table 1 shows the various formats supported for the different types.)

Since the STGMEDIUM structure contains the union member `lpszFileName` (of type LPOLESTR), I decided to have a go at retrieving the filename as a (Unicode) string, rather than worry about the other formats. This worked well, but there are two (and a bit) problems. First, it transpires that the operating system on which I was working, XP, is the only one of the NT family that provides the FileName and FileNameW formats for NetServer types; as soon as I went to test on NT 4 or 2000, the shell extension silently failed to do anything. I think that such inconsistencies are quite inappropriate, but that's the world of shell extensions: There's a great deal of variation over the various Win32 incarnations. (The partial problem is that FileNameW is only provided on NT-family systems, albeit that's irrelevant for this shell extension, as we've seen that remote shutdowns can only be done from NT family machines.) The more serious problem is that FileNameW returns only a single filename, so if we had multiple selections only one would appear.

So FileNameW not being appropriate, and Net Resource documented to also return only a single path name, I decided to bite the bullet and deal with the Shell IDList Array format. Alas, it turns out that for NetServer types, even this format returns only one item.

Table 1 Clipboard formats for shell types

Format	Files	Directories	Network Server
CF_HDROP	yes	yes	no
Shell IDList Array	yes	yes	yes
FileName	yes	yes	XP-only
FileNameW	yes	yes	XP-only
Net Resource	no	no	yes

When you have more than one server shortcut selected, it is the one with the caret whose name is passed through, whether that is in Shell IDList Array, Net Resource, or FileNameW formats. I guess this makes sense as far as it goes, and in this case is not contradictory; I would not want to write the extension to be able to shutdown multiple machines simultaneously as it is a very serious thing to be doing on one machine, never mind several at a time. However, it is conceivable that one may write extensions to do many useful (and benign) operations with network servers, in which case this restriction would be onerous. Moreover, despite not wanting to remotely reboot multiple machines, we still have a problem, as we do not want the reboot context menu items to appear when multiple machines are selected since we have no control over (and the user would have no idea as to) which of the selected machines would be operated on. Nasty.

I was able to find nothing in the shell extension documentation to help out here, so my somewhat hacked solution is to call `GetFocus()`, which retrieves the window in the current thread that has the focus. Since the shell extensions are in-process COM servers, they operate within the shell process (Explorer.exe) and of course, the window that has the focus is the one in which the selections have been made and right-clicked. Whether in one of Explorer's SDI tree-list windows or the desktop itself, the window concerned is a list view ("SysListView32"), so my solution is to send the focused window the LVM_GETSELECTEDCOUNT message. If the result is greater than one, then there are multiple items selected. If the result is 0, then the window is not a list view and the user is probably using

Listing 6 WTLSTL's SimpleContextMenuHandler‹›

```
/* ////////////////////////////////////////////////////////////
 *
 * ...
 *
 * Extract from wtlstl_simple_context_menu_handler.h
 *
 * www:     http://www.synesis.com.au/wtlstl
 *          http://www.wtlstl.org/
 *
 * Copyright (C) 2002, Synesis Software Pty Ltd.
 * (Licensed under the Synesis Software Standard Source License:
 *  http://www.synesis.com.au/licenses/ssssl.html)
 *
 * ...
 *
 * ////////////////////////////////////////////////////////////// */

...

template <ss_typename_param_k C>
struct SimpleContextMenuItem_
{
  typedef void (C::*PFN)();

  UINT  idsMenuItem;
  UINT  idsCommandString;
  PFN   pfn;
};

#define BEGIN_SIMPLE_CONTEXT_MENU_MAP() \
private: \
  typedef wtlstl::SimpleContextMenuItem_)<class_type> \
                                                ContextMenuItem; \
public: /* Alas must be public, but at least ContextMenuItem isn't */ \
  static ContextMenuItem const *WINAPI \
                      _GetContextMenuEntries(size_t *psize = NULL) \
  { \
    static ContextMenuItem const _entries[] = \
    { \

#define SIMPLE_CONTEXT_MENU_ENTRY(idsm, idscs, pfn) \
      { idsm, idscs, pfn },

#define END_SIMPLE_CONTEXT_MENU_MAP() \
      { 0, 0, NULL } \
    }; \
    if(NULL != psize) \
    { \
      *psize = stlsoft_num_elements(_entries) - 1; \
    } \
    return _entries; \
  }



template <ss_typename_param_k C>
class SimpleContextMenuHandler
  : public IContextMenu
{
protected:
  typedef C                               boltee_type;
  typedef SimpleContextMenuHandler<C> context_menu_handler_type;
private:
  typedef SimpleContextMenuItem_<C>   ContextMenuItem;

// IContextMenu
private:
  STDMETHOD(QueryContextMenu)(HMENU hmenu, UINT indexMenu,
                    UINT idCmdFirst, UINT idCmdLast, UINT uFlags)
  {
    HRESULT hr;

    if(uFlags & CMF_DEFAULTONLY)
    {
      hr = S_OK;
    }
    else
    {
      size_t                cItems;
      ContextMenuItem const *items =
boltee_type::_GetContextMenuEntries(&cItems);

      // Ensure we have enough room for all the menu items we want to present
      if(idCmdLast < idCmdFirst + cItems + 1)
      {
        hr = E_FAIL;
      }
      else
      {
        int   index;

        // For each item, load the string and add to the menu.
        for(hr = S_OK, index = 0; index < cItems; ++index)
        {
          TCHAR                 szMenuItem[256];
          ContextMenuItem const &item = items[index];

          if( 0 == ::LoadString(_Module.GetResourceInstance(),
                                item.idsMenuItem,
                                szMenuItem,
                                stlsoft_num_elements(szMenuItem)) ||
              !::InsertMenu(hmenu,
                            indexMenu++,
                            MF_STRING | MF_BYPOSITION,
                            idCmdFirst++,
                            szMenuItem))
          {
            hr = HRESULT_FROM_WIN32(::GetLastError());

            break;
          }
          else
          {
            ATLTRACE(_T("Menu item: %d => %d, %d\n"),
                        index, item.idsMenuItem, item.idsCommandString);
          }
        }

        if(!FAILED(hr))
        {
          // Note: return S_OK + # of items
          hr = MAKE_HRESULT(SEVERITY_SUCCESS, FACILITY_NULL, index);
        }
      }
    }

    return hr;
  }

  STDMETHOD(InvokeCommand)(LPCMINVOKECOMMANDINFO lpici)
  {
    HRESULT hr;

    if(HIWORD(lpici->lpVerb) == 0)
    {
      UINT                  cmdOffset = LOWORD(lpici->lpVerb);
      size_t                cItems;
      ContextMenuItem const *items    =
boltee_type::_GetContextMenuEntries(&cItems);

      if(cmdOffset < cItems)
      {
        // Run the handler for the item
        (static_cast<boltee_type*>(this)->*items[cmdOffset].pfn)();

        hr = S_OK;
      }
      else
      {
        _ASSERTE(!"Unexpected menu item invoked");
        hr = E_UNEXPECTED;
      }
    }
    else
    {
      // No need to support string based command.
      hr = E_INVALIDARG;
    }

    return hr;
  }

  STDMETHOD(GetCommandString)(UINT cmdOffset, UINT uType,
                    UINT *pwReserved, LPSTR pszName, UINT cchMax)
  {
    HRESULT hr;

    if(pszName != NULL)
    {
      *pszName = 0;
    }

    typedef int (WINAPI *PfnLoadString)(HINSTANCE , UINT , LPVOID , int);

    PfnLoadString fns[2] =
    {
        (PfnLoadString)LoadStringA
      , (PfnLoadString)LoadStringW
    };

    switch(uType)
    {
      default:
        _ASSERTE(!"Unrecognised GetCommandString() type");
```

a custom shell process, within which we're not going to be able to operate anyway. In either case, the E_FAIL code is returned from `IShellExt::Initialize()` and the shell will not then proceed to call the methods of `IContextMenu` and the menu items are not shown. Only when the selection count is one does the initialization proceed.

The remainder of the method (Listing 5) shows how to extract the path information from the Shell IDList Array format, which provides a memory block in the `hGlobal` member of the STGMEDIUM structure containing a CIDA structure, which is defined as

```
typedef struct _IDA {
    UINT cidl;
    UINT aoffset[1];
} CIDA, * LPIDA;
```

This innocent-looking structure definition belies a complex and troublesome nature. It is actually used to represent a contiguous layout of ITEMIDLISTs. `aoffset` is an array, of dimension `1 + cidl`, of offsets into the block where the ITEMIDLISTs reside. The CIDA always contains an entry for the parent folder of the items concerned, so `cidl` represents only the number of child items. The parent ITEMIDLIST is located immediately `aoffset[0]` bytes from the start of the block. Each child item `n` is located at `aoffset[1 + n]` bytes from the start of the block. All this mind-numbing stuff can be more easily handled by using the macros `HIDA_GetPIDLFolder()` and `HIDA_GetPIDLItem()` suggested in the MSDN help (they do not appear in headers, hence their inclusion in the implementation file). Since we are dealing with only one child item, we just retrieve the parent folder and item 0.

When writing shell extensions (or other code that operates with shell structures), obtaining the filesystem path from ITEMIDLISTs is as simple as calling `SHGetPathFromIDList()`; passing in a pointer to the list and a pointer to a character buffer (of sufficient size to handle any valid path). Unfortunately, obtaining the information about a network server is not as simple. We need to call `SHGetDataFromIDList()` on the item's ITEMIDLIST and request a NETRESOURCE structure, but that function also requires the parent folder (as an `IShellFolder` instance) of the given item against which to bind the data. In order to get an `IShellFolder` instance from an ITEMIDLIST, we need to call `IShellObject::BindToObject()`. But what do we call it on? The answer is the `IShellObject` that represents the root of the desktop namespace; in other words, the desktop folder itself, which we obtain from `SHGetDesktopFolder()`. (All these interfaces follow COM rules in that they must be released when finished with.)

Once we have the folder object we can now call `SHGetDataFromIDList()`, passing the

Listing 6 Continued

```
      case  GCS_VERBA:
      case  GCS_VERBW:
        // Verbs are not supported currently
        hr = E_NOTIMPL;
        break;
      case  GCS_VALIDATEA:
      case  GCS_VALIDATEW:
        _ASSERTE(!"GCS_VALIDATE never expected by menu handlers");
        hr = E_UNEXPECTED;
        break;
      case  GCS_HELPTEXTA:
      case  GCS_HELPTEXTW:
        {
          size_t          cItems;
          ContextMenuItem const *items  =
boltee_type::_GetContextMenuEntries(&cItems);

          if(cmdOffset < cItems)
          {
            ContextMenuItem const &item = items[cmdOffset];

            if(fns[(uType & GCS_UNICODE) != 0]( _Module.GetResourceInstance(),
                          item.idsCommandString,
                          pszName,
                          cchMax) > 0)
            {
              ATLTRACE(_T("Menu item: %d => %d, %d\n"),
                    cmdOffset, item.idsMenuItem, item.idsCommandString);

              hr = S_OK;
            }
            else
            {
              hr = E_INVALIDARG;
            }
          }
          else
          {
            hr = E_UNEXPECTED;
          }
        }
        break;
    }

    return hr;
  }
};

. . .
```

Listing 7 Using SimpleContextMenuHandler‹›

```
class ATL_NO_VTABLE CRemoteReboot
  : public CComObjectRootEx<CComSingleThreadModel>
  , public CComCoClass<CRemoteReboot, &CLSID_RemoteReboot>
  , public IShellExtInit
  , public SimpleContextMenuHandler<CRemoteReboot>
{
  typedef CRemoteReboot class_type;

/// Construction
public:
  CRemoteReboot()
  {
    m_szHost[0] = 0;
  }

DECLARE_REGISTRY_RESOURCEID(IDR_REMOTEREBOOT)

BEGIN_SIMPLE_CONTEXT_MENU_MAP()
  SIMPLE_CONTEXT_MENU_ENTRY(IDS_SHUTDOWN_SVR, IDS_SHUTDOWN_SVR_CMD,
                            OnShutdownServer)
  SIMPLE_CONTEXT_MENU_ENTRY(IDM_REBOOT_SVR, IDS_REBOOT_SVR_CMD,
                            OnRebootServer)
  SIMPLE_CONTEXT_MENU_ENTRY(IDM_REBOOT, IDS_REBOOT_CMD, OnReboot)
END_SIMPLE_CONTEXT_MENU_MAP()

BEGIN_COM_MAP(CRemoteReboot)
  COM_INTERFACE_ENTRY_IID(IID_IShellExtInit, IShellExtInit)
  COM_INTERFACE_ENTRY_IID(IID_IContextMenu, IContextMenu)
END_COM_MAP()

// IShellExtInit
private:
  STDMETHOD(Initialize)(LPCITEMIDLIST   pidlFolder,
                        LPDATAOBJECT    lpdobj,
                        HKEY            hkeyProgID);

// Implementation
private:
  // Map handlers
  void OnShutdownServer();
  void OnRebootServer();
  void OnReboot();

  // Helpers
  void ShutdownServer(bool bReboot, bool bForce, LPCTSTR pszMessage);

// Members
private:
  TCHAR m_szHost[1 + _MAX_PATH];
};
```

folder, the item we want to resolve, the format we require (SHGDFIL_NETRESOURCE), and the buffer to receive the information. More contiguous memory complexities here, so we derive from the NETRESOURCE structure and thereby add the required 1024 bytes. (You'll note the interesting trick of deriving an anonymous structure from a named one—it looks weird but works fine on most compilers, even the very canny Metrowerks CodeWarrior, but it still may not be standards compliant.)

The final step is to test for, and then remove, the double backslash from the server name. This is for cosmetic purposes really, since most network functions work well with server names that are double-backslash prefixed just as well as those that are not. Then we save the name in the `m_szHost` member, and return.

IContextMenu::QueryContextMenu()

After successful initialization, the shell will then ask the shell extension for its menu items. This is done by a call to `IContextMenu::QueryContextMenu()`. The shell passes in the following information: the menu handle, the index of the insertion point in the menu, a range of command identifiers, and flags.

If the flags specify the value CMF_DEFAULTONLY, then the method simply returns, since we do not affect the default menu item in this context menu handler. Otherwise, we step through our list of items and insert them into the menu. For each insertion, we use the current index point and the current command identifier, each of which is then incremented. Hence, on a third item, we'll be inserting at index `indexMenu + 2` with `ID idCmdFirst + 2`. In terms of identifying commands, context menu handler shell extensions work on an indexed basis. Thus, when the shell later calls back because the user has either selected or clicked on an item from this shell extension, it passes the index of the item, not the command ID. Furthermore, it passes an index that is relative to the absolute index (`indexMenu`) that it passed in the call to `QueryContextMenu()`. So, when the user clicks on our third item, we will get a call back to say that item 2 has been clicked. This is actually the simplest and easiest way to do it, although at first it seems a little strange (and I recall my first shell extension failing to work at all as first I'd remembered the menu ID, and then later the absolute index, before I finally got it correct).

For simple context menu handlers, I use the WTLSTL template class `SimpleContextMenuHandler` and its associated macros, shown in Listing 6. (WTLSTL is another subproject

of STLSoft, located at http://wtlstl.org/, pertaining to WTL—see the online sidebar "WTL & ATL.") The SIMPLE_CONTEXT_MENU_ENTRY() macros associate two string resource identifiers (one for the menu item, one for the help string) with a handler method, as shown in the class definition for the Remote Reboot handler in Listing 7. This makes it easy to internationalize the menu and help strings, and also provides a simple and neat framework within which one can focus on operations rather than infrastructure.

IContextMenu::GetCommandString()

When the user moves the mouse over a menu item that was inserted by your context menu handler, the shell will call you back via the `IContextMenu::GetCommandString()` method to get a help string to display (in the status bar of the Explorer window). As mentioned earlier, the given index (the `cmdOffset` parameter) corresponds to the position in our list of SIMPLE_CONTEXT_MENU_MAP() entries. The implementation is very straightforward: Index the item and load the string.

There is, however, a small complication. The method is declared with the parameter `pszName` being of type LPSTR, but in order to support Unicode systems as well as ANSI, we must cast it to LPWSTR. Even though this shell extension will work only on Unicode systems, as a general rule I like to support both, and `SimpleContextMenuHandler` does so by calling either `LoadStringA()` or `LoadStringW()` depending on whether GCS_HELPTEXTA or GCS_HELPTEXTW is the command type passed to the method. Of the other command types, GCS_VALIDATEA/W are not sent to context menu handlers, so we can ignore them, and this context menu handler does not support verbs, so we can ignore them also.

IContextMenu::InvokeCommand()

This is the method where everything happens but, thanks to our index entry scheme, it is the simplest. If the high word of the `lpVerb` member of the CMINVOKECOMMANDINFO structure passed to the method is 0, then the low word is the index of the command. We validate the index and then call the appropriate method.

CRemoteReboot

So we've covered the basics of context menu handler shell extensions. We've seen how `CRemoteReboot`'s `IShellExtInit::Initialize()` method dealt with getting the data from the shell, and also how the WTLSTL `SimpleContextMenuHandler` class can simplify the functionality of `IContextMenu` for us. Now it's time to focus on the specifics of Remote Reboot itself. This simply involves the implementation of the three handler methods, as shown in Listing 8 (available online), which handle the three menu items shown in Figure 3.

`OnRebootServer()` and `OnShutdownServer()` both call the helper method `ShutdownServer()`, which is where all the action happens, passing True and False, respectively, to stipulate whether to reboot or just to shutdown. The second parameter of `ShutdownServer()` is a Boolean stipulating whether to forcibly terminate the hosts. Both handlers call the WinSTL function `IsKeyPressedAsync(VK_SHIFT)`, which means that the user can hold the shift key down when selecting the menu item rather than having to open the Remote Reboot dialog (see Figure 4, available online) in order to effect a forced termination. Forcing reboot/shutdown simply means that if an application on the remote host does not shutdown cleanly (i.e., because it has a dialog open), then it will be terminated. Whether you are forcing or not, any users on the remote host are likely to lose their work, so don't think about using this tool maliciously in your office unless you are looking for a swift change of scenery!

`ShutdownServer()` is pretty straightforward. It loads the timeout and message values for the

Listing 9 Registry entries

```
HKCR
{
  NoRemove CLSID
  {
    ForceRemove {00537963-0000-0008-0004-00c0dfe64a64} = s 'Remote Reboot Context Menu Handler'
    {
      InprocServer32 = s '%MODULE%'
      {
        val ThreadingModel = s 'Apartment'
      }
    }
  }

  NoRemove NetServer
  {
    NoRemove shellex
    {
      NoRemove ContextMenuHandlers
      {
        ForceRemove 'Remote Reboot Context Menu Handler' = s '{00537963-0000-0008-0004-00c0dfe64a64}'
        {
        }
      }
    }
  }
}

HKLM
{
  SOFTWARE
  {
    Microsoft
    {
      Windows
      {
        CurrentVersion
        {
          'Shell Extensions'
          {
            Approved
            {
              val {00537963-0000-0008-0004-00c0dfe64a64} = s 'Remote Reboot Context Menu Handler'
            }
          }
        }
      }
    }
  }
}
```

reboot/shutdown operation from the registry. If they are not yet present, then it uses default values. Then it calls the Win32 function `InitiateSystemShutdown()`, which commands the given host (`m_szHost`, elicited in the `IShellExtInit::Initialize()` method) to shutdown/reboot according to the given parameters. (Actually, `InitiateSystemShutdown()` is erroneously prototyped to take pointers to nonconst characters for the hostname and message strings, so what is called throughout the implementation is an inline overload, defined in stdafx.h, that takes const parameters.)

If the shutdown call fails, then `GetLastError()` is called, and the message text is `sprintf()`-ed into a dialog with the `MessageBox_printf()` function (which I described in a "Tech Tip" in the May 2003 issue), as shown in Figure 5 (available online).

If the function succeeds, then the pending dialog (see Figure 6, available online) is shown. It operates with a timer, and provides the progress of the timeout period as a countdown and an abort button to allow the shutdown/reboot to be cancelled. It is worth noting that it is appropriate (not to say necessary) to create a modal dialog here, because Explorer creates a new (user interface) thread within which to run any activated shell extensions.

The final handler, `OnReboot()`, invokes the `CRebootDialog`, as seen in Figure 5 (available online). There's no space here to discuss its implementation in detail. It's pretty standard fare for ATL dialogs although I do make use of various STLSoft and WTL control classes to simplify the manipulation of the dialog controls, the other dialogs, and the context-sensitive help (see the online sidebar "WTL & ATL").

Registering Shell Extensions

In order to be recognized and invoked by the shell, shell extensions must be registered. Registration of the Remote Reboot shell extension is effected via an ATL registry script, shown in Listing 9.

As for any in-proc COM server, there is an entry under HKEY_CLASSES_ROOT\CLSID, providing the `InprocServer32` subkey, and the associated threading model. However, there are two other keys. Under HKEY_CLASSES_ROOT\NetServer\shellex\ContextMenuHandlers there is an entry providing the CLSID of the shell extension. It is this entry that allows the shell to determine that this shell extension is provided for network servers.

Shell extensions can be installed on Windows 95-family systems at any user's discretion, but installing on NT-family systems requires that you have rights to write to the registry. Furthermore, on these systems, the administrator can restrict the launch of shell extensions to those on the approved list, which reside in:

```
HKEY_LOCAL_MACHINE
  Software
    Microsoft
      Windows
        CurrentVersion
          Shell Extensions
            Approved
```

All of the Synesis Software Shell Extensions include entries for the Approved section (which are benignly ignored on 95-family systems), since I want them to be available on secure systems. Registering on NT-family requires sufficient rights to be able to write to HKEY_LOCAL_MACHINE, so it may require installation by the machine's administrator.

Debugging Shell Extensions

So we've looked at how to shutdown remote systems, learned about how context menu handler shell extensions interact with the shell, how simple ones can be implemented, and how to register them. The only thing that remains is how to debug them.

Since shell extensions are in-process COM servers, they have to be debugged within a host process. Unless you have written a fully functional custom test harness (which I doubt), the host process will be Explorer itself. If you're using Visual C++, you need to set c:\winnt\explorer.exe (or whatever the equivalent path is on your system) to be the "Executable for debug session." That's only half the picture, however, since Explorer is very likely already running. Running another instance of Explorer causes the first process to open up another window, and the second process to terminate quietly. (Only on systems experiencing some kind of problem are you likely to see more than one instance of the process running, and in such cases you're going to be crashing pretty soon anyway.)

We need to be able to start explorer.exe in the debugging session on a system where Explorer is not running. The answer to this is to kill the existing one. A crude method is to run up task manager and kill explorer.exe, but this can leave the system in an unstable state. The sophisticated way of doing it is to invoke the system shutdown dialog—either via Ctrl-Alt-Del, Shutdown or from Start, Shutdown—

and then holding down Ctrl-Alt-Shift (left-hand keys) and clicking on the Cancel button. (You can also hit the Esc key rather than clicking on Cancel, but on my laptop this invokes system hibernation, which is somewhat inconvenient.)

As well as persuading Explorer to close itself down gracefully, this sequence tells the system not to try and restart the shell, which it otherwise may do. There are varying degrees of compliance, of course: XP never subsequently restarts Explorer without being asked, 2000 does it infrequently, and NT4 does it a lot.

So now that we've gotten rid of the shell, we can start debugging. Once the process is up, you can then right-click on the appropriate shell item and you'll hit any breakpoints you've set up. I usually have one on the entry of `IShellExtInit::Initialize()` and on the handler-specific interface methods, in this case the three methods of `IContextMenu`. You can then debug as you would any other DLL/COM component.

For context menu handlers, breaking within `IShellExtInit::Initialize()` and `IContextMenu::QueryContextMenu()` will move the focus to the debugger, so the menu will actually be cancelled—don't be misled by this into thinking that your shell extension is not working. Once you're satisfied that everything is OK with these two methods, it's best to disable the breakpoints therein, so that you can get on with the `GetCommandString()` and `InvokeCommand()` methods. One last tip: When debugging within `GetCommandString()` you can get the shell, indeed the whole system, in a weird state whereby your debugger can be hung. This is no doubt due to Windows' fundamental menu-handling logic—I've experienced similar behavior when debugging other menu functionality—but worry not. On NT-family systems you only need to hit Ctrl-Alt-Del and then hit cancel, and it all gets nicely cleared up (most of the time).

Don't forget to change the path for Explorer if you're testing on multiple boots on the same system. I can assure you that NT4's Explorer.exe will not execute on 2000, XP, and so on, and you may experience a few panicked moments, imagining you've trashed your system or the shell extension, before you realize your oversight.

The advice I've given about debugging has been all NT-family based. Alas, it is too many years since I did any debugging of any kind on 95-family operating systems, and I cannot remember whether I ever did shell extension debugging on them. I suspect I probably made do with `OutputDebugString_printf()`-style debugging on them. I'm not sure how many shell extension developers will be disenfranchised by this lack of advice, but judging from the hit-counts on http://shellext.com/, it is clear that the vast majority (>90 percent) of shell extension users are running NT-family machines.

Conclusion

I hope you've learned a little about shell extensions in general, and a lot about context menu handlers in particular. I also hope that I've sparked your interest in WTL and STLSoft (COMSTL, WinSTL, WTLSTL, and all the other little STLs), and I invite you to try both out, especially when developing small lightweight components. There's a lot more mileage in C++ as the primary development language for the Win32 platform than some quarters would have us believe, and there are still powerful and effort-reducing libraries being created that will support its position for a long time to come.

Acknowledgments

I'd like to thank Scott Patterson (http://www.gameframework.com/) for providing his usual constructive criticism while recovering from a nasty bout of the flu: above and beyond! I'd also like to thank the many users of the Synesis Software Shell Extensions for all your kind words, offers to buy, useful bug reports, and intriguing feature requests over the last couple of years. And, yes, they're going to continue to be free. Honest! w::d

| Download code › **windevnet.com/wdn/code/** |

Microsoft®

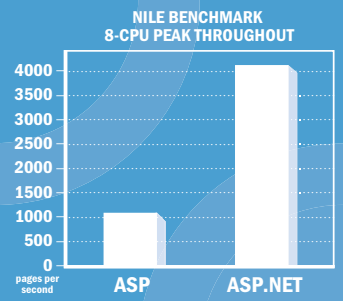
It can make your Web apps three times faster.

Visual Studio .NET can help you with (nearly) every part of your job. Your Web applications just got faster. ASP.NET, the Web application environment in Visual Studio® .NET, offers dramatically improved performance over classic ASP. Here's how: 1 **Compiled Page Execution** ASP.NET pages are compiled once and cached in memory instead of being interpreted each time the page is requested. 2 **Rich Output Caching** ASP.NET's caching features quickly retrieve database queries, full pages (or parts of pages), and objects from memory for improved app performance. 3 **Crash Protection** Web applications can't be fast if they're

*Connect time fees may apply. The Nile Application Benchmark is a Doculabs/Ziff Davis e-commerce benchmark that represents a complete, end-to-end, e-commerce application with realistic, heavy user loads placed on the system. See msdn.microsoft.com/vstudio/tryit for complete details.

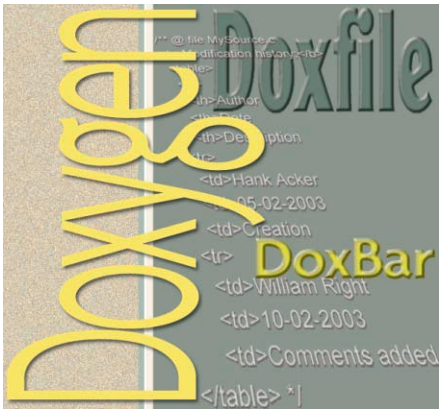
down. Duh. So ASP.NET automatically detects and recovers from errors like deadlocks so your application is always available. And now the newly released Visual Studio .NET 2003 is here for building and deploying even faster and more stable applications. Try it now: log on to a fully featured, free* online hosted session and get more information at **msdn.microsoft.com/vstudio/tryit**

Microsoft Visual Studio .net



Better Docs with Doxygen

Document generation for C, C++, C#, Java, VB, Delphi, and more



ASK A SOFTWARE ENGINEER about source documentation and you are likely to hear one or more of the following observations: sources of other developers are poorly documented; it takes too much time to document my own sources; or source documentation is never up to date. And let's be honest, documenting sources tends to be a lousy job, especially when it has to be done afterwards.

Doxygen is a popular solution to this problem. With Doxygen, you can generate documentation based on the source files themselves. It works for sources written in several languages: C, C++, C#, Java, JavaScript, Visual Basic, Delphi, and PHP.

Doxygen is a freeware tool developed by the Dutchman Dimitri van Heesch. It started as a Linux-based tool but has been ported to Windows. Moreover, other developers have created Windows-based extensions to Doxygen to integrate it with existing development tools, including DoxBar and GraphViz, which I'll discuss in this article.

One of the big advantages of Doxygen is its ability to make the documentation consistent with the source in one step. You have the option of generating the documentation in HTML or RTF format, thus serving both online and offline documentation purposes. In addition, numerous formatting options are provided.

MARTIN KEESEN *is a consultant for Technical Software Engineering with LogicaCMG in The Netherlands. His main working areas are the medical, telecom, and traffic control industries, with a focus on embedded software. He can be contacted at martin.keesen@logicacmg.com.*

In fact, there is only one disadvantage: Software engineers are losing their arguments against delivering thorough, high-quality source documentation!

Installing the Tools

The first thing to do is to get your copy of the latest version of Doxygen and accompanying tools from http://www.doxygen.org/. You have to pick the binary distribution for Windows 95/98/Me/NT/2000/XP. This is a self-installing archive including the HTML and compressed HTML versions of the manual and the GUI front end.

I also advise you to separately download the documentation in either PDF or compressed HTML format. Unlike many other freeware tools, the documentation of Doxygen is superb. Of course, this is not surprising for a documentation tool! In this article, I'll assume you install Doxygen in its default location: Program Files\doxygen.

If you are using Visual C++, you should download DoxBar from http://sourceforge.net/projects/doxbar. DoxBar is an add-in to use Doxygen from within Microsoft's Visual C++ integrated development environment. Use it to run Doxygen on a selected project and to search in Doxygen-generated documentation. The ZIP file you download contains two files, which you should extract into directory Program Files\doxygen\bin. Next, open the file, doxbar.chm, that you just extracted. This compressed HTML file contains detailed instructions on how to install DoxBar in combination with the Visual C++ IDE.

Also interesting to download is the file VC-Macros4Doxygen.zip, which contains a set of macros for the Visual C++ IDE. Extract the file DoxygenMacros.dsm into Program Files\Microsoft Visual Studio\Common\MSDev98\Macros to speed up your source documentation activities.

The last item to download is the graphic support system GraphViz, which Doxygen uses to produce graphics. You should pick the Wise Install Package from http://www.research.att.com/sw/tools/graphviz/download.html. After you have run this installer, it is easy to extend the PATH environment variable to include Program Files\ATT\Graphviz\bin.

Configuring Doxygen

Doxygen uses a file called "Doxyfile" to store its configuration. I advise you to let Doxygen generate this file the first time. You can do this by executing the file Program Files\Doxygen\bin\doxygen.exe from the DOS command prompt while providing command line option *-g*. Use the generated file Doxyfile as a template and copy it to your project's directory.

Of course, you can edit this file by opening it in a text editor, but this is a tedious job to do. Here is where DoxBar comes in. When you click on the DoxBar Edit button, don't be overwhelmed by what appears next (see Figure 1).

Indeed the popup window you see now contains many options. Luckily, if you leave your mouse on an entry zone, you will get a tool tip message explaining the indicated option. This feature guides you surprisingly fast through a wealth of configuration options.

Doxbar derives the contents of the tool tip messages from the comments in the file Doxyfile. This file directly controls even the presence of the tabs. For example, to remove the man page output tab, you simply remove all the lines related to man pages options.

Even without specific tags to elaborate the generated source documentation, Doxygen extracts basic information from the files in your project

You cannot remove the Doxbar-specific tab. This tab lets you choose how DoxBar explores the project collection in the current workspace. Selecting the active project and its dependencies requires that the workspace file (.dsw) and the active project file (.dsp) have the same name. DoxBar issues a warning if this is not the case.

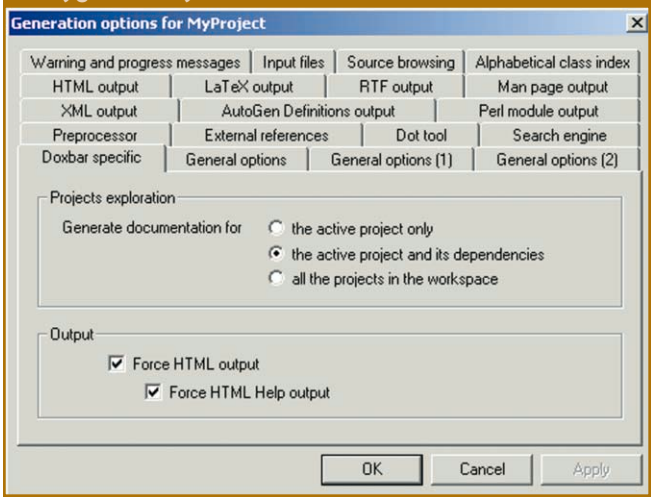
Please note that the changes DoxBar makes are stored in the project's Doxyfile in the project's directory, not in the default template.

Your First Project

Even without specific tags to elaborate the generated source documentation, Doxygen extracts basic information from the files in your project. This information includes a file list, dependency graphs, and inheritance plus collaboration diagrams.

If you generate the HTML version of this information, you have online access to it and can directly see the changes to it when you alter an option.

Figure 1 The DoxBar add-on provides a GUI for Doxygen's many features



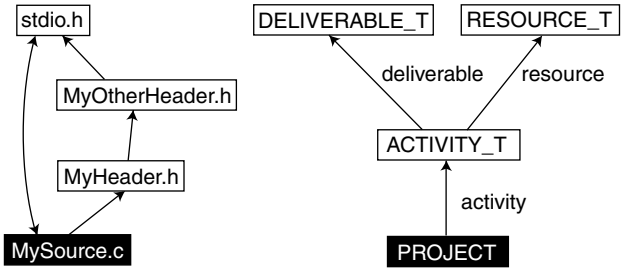
Now use your web browser to navigate through the menu tree included in the HTML output (see Figure 2).

Experiment with the settings to see how they influence the output created. The settings for EXTRACT_ALL (general options), HAVE_DOT (Dot tool), HTML_STYLESHEET (HTML output), and SOURCE_BROWSER (source browsing) are especially worth experimenting with.

While generating source documentation, you will notice that Doxygen reports undocumented items via the Doxygen output pane in the

Figure 2
Navigating the menu tree

Figure 3
Generated graphs clearly show code



IDE. Note that this is only true if EXTRAC_ALL is off and WARN_IF_UNDOCUMENTED is turned on.

Reverse Engineering

Doxygen is very useful in case you have to reverse engineer large quantities of code. One of the goodies of Doxygen is that the HTML output uses syntax highlighting to ease navigation through the code fragments included in the documentation. The graphs also give fast insight into the code structure; for example, the include nesting and structs in Figure 3. Therefore, without even changing anything in the source code, you are able to better understand it by applying Doxygen.

Creating Even Better Source Documentation

Adding extra tags to the source code allows Doxygen to greatly increase the usability of the generated docs. Doxygen uses comments that follow certain conventions to build more detailed and structured documentation. Refer to the manual for an extensive list of all the possible tags. Inserting the required tags is a piece of cake if you use the macros mentioned earlier. You can insert file, attribute, single, or multiline headers with these macros.

Modifying your comments is one of the easiest changes. By changing all applicable /* into /** in C source reveals a lot of extra information included in the documentation. Note that comments in the source precede the item to be documented in Figure 4.

Depending on the position of the source, the text of the comment appears in one of the following summaries: files, namespaces, classes, structs, unions, templates, variables, functions, typedefs, enums, or defines.

In a function header you can explicitly describe the function's goal, use of the parameters,

Figure 4 Modifying your C comments allows Doxygen to generate more info in the docs

Source (in file MyHeader.h):

```
/** Identifiers for all states an activity can be in */
typedef enum
{
   INITIAL,  /* Initial state */
   /** Idle, no activity */
   IDLE,
   ONGOING   /* Work in progress */
} STATE_T;
```

Doxygen Output:

enum STATE_T

Identifiers for all states an activity can be in

Enumeration values:

IDLE Idle, no activity

Definition at line 12 of file MyHeader.h

```
00013 {
00014    INITIAL,      /* Initial state */
00016    IDLE,
00017    ONGOING       /* Work in progress */
00018 }  STATE_T;
```

Figure 5 Parsing parameter comments

Source (in file MySource.c):

```
/** Do some parsing on a given string
  * @param work_string Pointer to string of characters
  * @param max_length  Maximum number of characters to evaluate
  * @return OK if string is parsed correctly, else NOK
  */
RESULT_T MyFunc(char* work_string, int max_length)
{
}
```

Doxygen Output:

RESULT_T MyFunc(char* *work_string,*
int *max_length*
)

Do some parsing on a given string

Parameters:

work_string Pointer to string of characters
max_length Maximum number of characters to evaluate

Returns:

OK if string is parsed correctly, else NOK

Definition at line 26 of file MySource.c

and the return values to be expected, as in Figure 5.

You can also achieve some nice things by applying HTML tags in the comments, as in Figure 6. Use this feature with care because the HTML has to be syntactically correct.

Figure 6 HTML-tagged comments

Source (in file MySource.c):

```
/** @file MySource.c
 <b>Modification history:</b>
   <table>
    <tr>
     <th>Author
     <th>Date
     <th>Description
    <tr>
     <td>Hank Acker
     <td>05-02-2003
     <td>Creation
    <tr>
     <td>William Right
     <td>10-02-2003
     <td>Comments added
   </table> */
```

Doxygen Output:

Detailed Description

Modification history:

Author	Date	Description
Hank Acker	05-02-2003	Creation
William Right	10-02-2003	Comments added

Well-Written Manuals

A big disadvantage of most freeware software is the lack of a good manual to work with. This is where Doxygen clearly stands out by providing a very well-written set of manuals. The set consists of a User Manual, a Reference Manual, and a Developers Manual.

The User Manual deals with the installation of the tool followed by a brief instruction on how to produce your first results fast. Next follows the different options to document your source with lists, diagrams, formulas, and linking with external documentation files.

The Reference Manual contains a description of all available features of Doxygen. In addition, the different output formats are described here. As already stated, Doxygen can be configured to create documentation in almost any thinkable layout. You can find all options to achieve this in the Reference Manual.

Last but not least is the Developers Manual. This text explains the internal structure of the program and gives good insight into how the program does its thing.

Extending Doxygen

Distribution of Doxygen sources and binaries are under the terms of the GNU General Public License. Therefore, anyone can extend the capabilities of Doxygen. On the Doxygen web site, you can find a wish list so long that it would be impossible for the author, Dimitri van Heesch, to implement all of it on his own. That is why he invites others to help him to achieve these goals in the near future.

After you have read the Developers Manual, it will not be difficult to estimate your possible contribution and to decide to join the development crew! w::d

| Download code › **windevnet.com/wdn/code/** |

Four Full Blown Conferences

OCTOBER 12-15, 2003

La Quinta Resort & Club
Palm Springs, CA

This is a unique opportunity to get your cutting edge technology & training direct from the source: Microsoft® Architects and World Renowned Gurus

- *OVER 180 IN-DEPTH SESSIONS TO CHOOSE FROM*
- *70+ SPEAKERS*
- *COMPUTER LAB, EXPO & NETWORKING PARTIES*

REGISTER EARLY and you'll receive FREE access to the sessions of these concurrently run events.

MICROSOFT ASP.NET Connections

Now the largest and most informative ASP.NET event in the industry!

VS Connections
Visual Basic .NET ⟷ C#

Where the who's who of developers meet.

MICROSOFT KEYNOTES

Scott Guthrie
PRODUCT UNIT MANAGER, ASP.NET

David Lazar
GROUP PRODUCT MANAGER, .NET TOOLS AND SERVICES

SPEAKERS

Mike Amundsen
Asli Bilgin
Markus Egger
Dino Esposito
Ken Getz
Scott Guthrie
Alex Homer
Rob Howard
Tom Howe
Rockford Lhotka
Paul Litwin
Jeff Prosise
Doug Seven
Steven Smith
Joshua Trupin
Peter Vogel
and more!

Ari Bixhorn
Mark Dunn
Tom Eberhard
Markus Egger
Dino Esposito
Carl Franklin
Ken Getz
Richard Grimes
Fernando Guerrero
Billy Hollis
Tim Huckaby
Omar Khan
Don Kiely
Nickolas Landry
Rockford Lhotka
Juval Löwy
Ingo Rammer
William R. Vaughn
and more!

SPEAKERS SUBJECT TO CHANGE

REGISTER ONLINE AT www.DevConnections.com, OR CALL THE CONFERENCE HOTLINE AT 800-438-6720 or 203-268-3204.

Microsoft · msdn Microsoft Developer Network · Dr. Dobb's Journal · msdn magazine · windows developer network · UNISYS Imagine it. Done. · SQL Server · TECH Conferences LLC PENTON MEDIA

ONE PLACE. ONE TIME.

Connect with the Source!

DevConnections

SQL Server Magazine CONNECTIONS

The #1 SQL Conference from the #1 SQL Server Magazine!

Stan Sorensen
DIRECTOR, SQL SERVER PRODUCT MANAGEMENT

Joe Chang
Gert Drapers
Fernando Guerrero
Mike Hotek
Steve Kass
Don Kiely
Morris Lewis
Michael Luckevich
Brian Moran
Sanjay Soni
Mark Souza
Kimberly L. Tripp
William R. Vaughn
and more!

Microsoft Office System Conference

Turning Information into Impact

Richard McAniff
CORPORATE VICE PRESIDENT, MICROSOFT OFFICE

Joe Andershak
Asli Bilgin
Mary Chipman
John R. Durant
Mike Fitzmaurice
Ken Getz
Robert Green
Michael Hernandez
Tom Howe
Don Kiely
Chris Kunicki
Paul Litwin
Charles Maxon
Richard McAniff
Bill Ramos
Peter Torr
Peter Vogel
and more!

ONE LUCKY ATTENDEE WILL GO HOME WITH A HARLEY... SEE WEB SITE FOR DETAILS

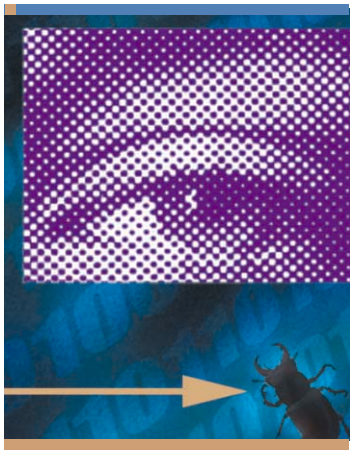
REGISTER Today - www.DevConnections.com

Be careful when using MI in Visual C++ 7.0; plus, more on overloading main

Multiple Inheritance

ONE OF THE FEATURES of C++ that, in my opinion, has "come into its own" over the last couple of years is multiple inheritance. MI has gotten a (somewhat justifiably) bad reputation as leading to far too many complications for the problems that it attempts to solve. You only need to get to the dreaded "diamond" hierarchy once to understand what problems can arise:

```
class Base {};
class Intermediate1 : public Base {}
class Intermediate2 : public Base {}
class Derived : public Intermediate1, public
  Intermediate2 {}
```



This hierarchy is so named because if you draw a diagram of the inheritance tree, it forms a diamond. Much has been written about the perils of such a class structure, so I won't go into it here. The kind of problems and complexity that arise, however, is one of the reasons that many people tend to shy away from multiple inheritance.

With the advent of COM and other interface-based programming methods, MI really started to shine. As you probably know, using COM in C++ involves defining interfaces, which are implemented as abstract base classes that contain only pure virtual functions and no data. For example:

```
class Interface1 : public IUnknown
{
public:
    virtual HRESULT Foo() = 0;
};

class Interface2 : public IUnknown
{
public:
    virtual HRESULT Bar() = 0;
};

class Implementation : public Interface1,
    public Interface2
{
public:
    HRESULT Foo()
    { /* Foo implementation */ }
    HRESULT Bar()
    { /* Bar implementation */ }
};
```

For COM, I'd also have to implement `QueryInterface`, `AddRef`, and `Release`, but I'll leave that out for now. The great thing about a model like this is that it completely separates the interface to the object from the implementation of that object. If I get a pointer to `Interface1`, I can call the `Foo` function, but I have absolutely no idea how it is implemented. All I know is that this object has a `Foo` implementation. This avoids building unnecessary dependencies on the implementation, which happen inadvertently all too often.

All this is great, but complications can still arise. In particular, suppose `InterfaceA` and `InterfaceB` both define the same function (I'll leave out `IUnknown` for simplicity; the same problem exists):

```
class InterfaceA
{
    virtual void Draw();
};

class InterfaceB
{
    virtual void Draw();
};

class MyImplementation : public InterfaceA
    public InterfaceB
{
public:
    // How do I overload InterfaceA::Draw
    // or InterfaceB::Draw?
};
```

If I want to provide a new implementation of the `Draw` function, what do I do? How can I specify which `Draw` I want to implement? As C++ stands now, I can't do it. There is no way to specify that within

JEFF CLAAR *is a software engineer at Nemesis in Southern California, writing drivers for consumer and professional audio products. You can submit your bugs to him at wdletter@cmp.com.*

the class definition. There *is* a way, however, by using some additional classes:

```
class MyImplA : public InterfaceA
{
public:
    virtual void InterfaceADraw() = 0;
    void Draw() { InterfaceADraw(); }
};

class MyImplB : public InterfaceB
{
public:
    virtual void InterfaceBDraw() = 0;
    void Draw() { InterfaceBDraw(); }
};

class MyImplementation : public MyImplA :
  public MyImplB
{
public:
    void InterfaceADraw() {...}
    void InterfaceBDraw() {...}
};
```

In this case, two additional base classes, `MyImplA` and `MyImplB`, are used as a form of indirection so that, essentially, the two base class `Draw` functions are given different names. I can then reimplement them as I see fit. While it would be nice to not have to create the two additional base classes, it's not too much effort; after all, they only have to reimplement the functions that have the same name in the two interface classes.

Listing 1
VC7's language extension for multiple inheritance

```
// Non-standard code that compiles successfully
// under Visual C++ 7.0 with standard options.

#include <iostream>
using namespace std;

class Base1
{
    public:
        virtual void foo()=0;
};

class Base2
{
    public:
        virtual void foo()=0;
};

class Derived : public Base1, public Base2
{
public:
    // This is a non-standard declaration!
    void Base1::foo() { cout << "Base1" << endl; }
    void Base2::foo() { cout << "Base2" << endl; }
};

int main(int, char**)
{
    Derived der;
    Base1* base1 = &der;
    Base2* base2 = &der;

    base1->foo();
    base2->foo();
    return 0;
}
```

The "Bug"

If I'm using Visual C++ 7.0, however, it appears that I don't have to implement the two base classes! Daniel Anderson sent in the code shown in Listing 1. It defines two base classes, similar to those just mentioned, and implements them in the derived class using the following syntax:

```
class Derived
{
...
    void Base1::foo() {...}
    void Base2::foo() {...}
};
```

This is completely nonstandard, but VC7 compiles it with no problems, and the resulting program behaves as you would expect. Each reimplementation is called as expected. All other compilers I tried it on failed to compile it.

Microsoft's Response

This appears to be expected behavior by VC7, but I was curious as to why it was implemented at all. After all, you can use the aforementioned technique with extra base classes to achieve the exact same behavior. With that in mind, I sent the code off to Microsoft for

comment. Jeff Peil responded with the following:

> *This is correctly identified as an extension. This extension was necessary for managed classes because managed classes can only multiply implement interfaces and cannot use multiple inheritance of base classes.*
>
> *—Jeff Peil*

I have to admit a considerable lack of knowledge about using managed classes, but if that's the case, then it certainly makes sense for VC7 to implement that feature. And I have to say it's certainly a convenient feature to have. It doesn't appear to be disabled, however, if I turn on the "Disable Language Extension" option in the project settings. If portability is not a concern, it shouldn't be a problem, but it should be kept in mind otherwise.

Overloading main

In a previous column, I talked a little bit about overloading `main`. In it, I mentioned that Windows may qualify as a freestanding environment, and as such it may not require a `main` function. However, reader Alan Stokes was quick to point out that the C++ Standard is pretty clear as to what qualifies as a freestanding environment, in a paragraph that I simply missed:

> *"A freestanding implementation is one in which execution may take place without the benefit of an operating system, and has an implementation-defined set of libraries that includes certain language-support libraries." (Section 1.4)*

Well, that makes it pretty clear that a C++ compiler for Windows does not qualify as a freestanding environment! (And no wisecracks, please, as to whether or not Windows 9X qualifies as an "operating system"!) It looks like Visual C++ is truly erroneous when it allows the program to overload `main`. I don't really qualify this as a particularly serious bug, though it is certainly incorrect.

Upon further thought, it's pretty clear that a C++ compiler for Windows must be a hosted (nonfreestanding) implementation. If it's not, that would mean compilers for UNIX, Linux, and just about any other operating system on the planet would not be hosted either. I'll have to be a little more careful reading in the future!

Farewell

This is the final installment that I'll be writing for Bug++. My day job is taking more and more time, and with an 18-month-old son and another on the way, I'm finding that I simply don't have the time necessary to devote to a column like this. Someone once said that writing an article is like running a sprint, and writing a monthly column is like running a marathon. After doing this month after month, I'd have to say that's a pretty accurate analogy! I'd like to thank John Dorsey for his support, as well as Amy Stephens for her work in getting all the parts I submit to her into one coherent whole. (And not to mention her understanding when I miss deadlines.) I'd also like to thank Ron Burk, the former editor of *Windows Developer* (back when it was *WDJ*) for giving me the opportunity to write this column in the first place. I'd even like to thank Microsoft (and Borland) for their professional responses, which were always courteous even though they were probably sick of hearing from me. Even when I was mistaken and misidentified a compiler bug, they never took the chance to rub it in! And most of all, I'd like to thank you readers for your submissions and your constructive criticism. I definitely learned a lot from all of you, and I appreciate the time you took to write. I can still be reached at jclaar3@cox.net with any comments and questions on past columns. Thanks again, and I hope you found the column useful! w::d

| Download code › **windevnet.com/wdn/code/** |

Please send us your best tricks and hacks—those clever pieces of code to make things work the way they should! You'll receive at least $50 for each tip that we print. Send your submissions to wdletter@cmp.com with the header "Tech Tip submission."

Installing a USB Filter Driver

ALAN MACINNES
alanmacinnes@earthlink.net

INSTALLING A WDM (WINDOWS Driver Model) filter driver for a USB device requires that just a few lines be added to the .INF file, which is used to install its primary device driver. In the example .INF file provided (Listing 1), FILTER.SYS will be installed as an "upper" filter driver to SESUSB.SYS. Three things need to be accomplished by these additional lines to the .INF file. First, the filter driver image itself, FILTER.SYS, must be copied to the %systemroot%\system32\drivers folder. Second, an entry must be added to the registry at HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Services to define the filter driver as a service. Finally, a registry value must be added to the entry within HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Enum/USB for these particular USB Vendor ID and Product ID values. This is to specify that there is an upper filter driver that must be loaded into the device driver stack for this particular device. (This .INF file was tested with FILTER.SYS, which is the sample USB filter driver provided in the Microsoft DDK).

Accessing IDL refTypes as C++ References

MATTHEW WILSON
matthew@synesis.com.au

THE INTERFACE DEFINITION LANGUAGE can, in common with C, manipulate function arguments as by-value or by-reference, where by-reference parameters are expressed in the form of pointers.

When such by-reference parameters cannot be NULL, the parameter is marked with the `[ref]` attribute, as in the `IInterface::NonNullMethod` in Listing 2.

This means that the marshaling code, and the implementing class's code, can always rely on a valid nonNULL pointer `pss` to transmit and use. However, in C/C++ client code, there is nothing to prevent a NULL from being passed (with the attendant crash following shortly thereafter). (Note that `[ref]` and another pointer attribute `[ptr]` cannot be applied to interface pointers, as they are always assumed to be `[unique]`, and specifies such leads to the MIDL compiler ignoring the attribute and giving the MIDL2034 warning. However, the concepts described here may still be applied usefully to interface pointers, in so far as providing the convenience of the use of references to interface-implementing objects.)

In C++, the reference is a very useful syntactic construct that provides the programmer with the ability to pass-by-reference while still appearing to be using object instances themselves, as opposed to pointers to them. Furthermore, since it is illegal—and in practice usually takes a deliberate effort—to pass a NULL reference, it is a very useful way for programmers to express to client programmers this semantic in their APIs.

Because reference arguments exhibit the same type conversion and, where applicable, polymorphic abilities as do pointers, it would be very useful to be able to pass reference parameters to interface methods that have been defined as being of `[ref]` type. This can be achieved with a simple trick on the part of the MIDL compiler's preprocessor commands.

It would be nice when compiling for C++ if the `pss` and `pii2` parameters to `NonNullMethod()` would be references, without that causing an issue to the MIDL compiler. The technique for this is very simple, and relies on using the MIDL `cpp_quote` keyword to insert preprocessor code for the C/C++ compile, not for the MIDL compile, as in Listing 3.

The use of `cpp_quote` to insert post-MIDL compile-time preprocessor instructions for the C/C++ compiler allows IDL and C/C++ to see different definitions of types. Because a C++ reference is equivalent to a pointer (in terms of what happens at the instruction level), the technique allows one to change the parameter type. It should be noted that great care must be taken to get the respective definitions correct, and to make sure they stay in sync as the IDL source evolves, or nasty things can happen.

Despite this technique having an inherent danger in IDL, it can help increase the safety of interface-using C++ code. It is clear how much more convenient this is, as well as its affording an additional level of type safety by enforcing the use of (C++) references to the interface's method's (IDL) reference parameters. For example, if one had wished to wrap SOMESTRUCT into a class `SomeStruct` and had a class `Class2` implementing `IOther`, the use of the `IInterface` interface with these types is very simple, as in Listing 4.

The only caveat is that one must ensure that the `cpp_quote` code is correct, and current, should the interface method change (though I am sure none of our good readers would ever change an interface except prior to its initial release).

GEORGE FRAZIER *is a software engineer in the System Design and Verification group at Cadence Design Systems Inc. and has been programming for Windows since 1991. He can be reached at georgefrazier@yahoo.com.*

Listing 1
USB filter driver example

```
; This .INF file demonstrates how to install a filter driver
; for a USB device.
;
; The lines marked with the comment "Filter driver install"
; identify those lines that were specifically added in order to install
; the upper filter driver onto the "driver stack" for this USB device
;
; If one were to remove these lines, what remains is the original
; .INF file to install just the one driver for the USB device.
;

[Version]
Signature="$WINDOWS NT$"
Class=USB
ClassGUID={36FC9E60-C465-11CF-8056-444553540000}

[Manufacturer]
%MfgName%=MyDriver

[MyDriver]
%USB\VID_07CC&PID_0003.DeviceDesc%=SESUSB.Dev, USB\VID_07CC&PID_0003

[DestinationDirs]
SESUSB.Files.Ext = 10,System32\Drivers

[SESUSB.Dev.NT]
CopyFiles=SESUSB.Files.Ext

[SESUSB.Files.Ext]
SESUSB.SYS
FILTER.SYS                                    ; "Filter driver install"

[SESUSB.Dev.NT.Services]
Addservice = SESUSB, 0x00000002, SESUSB.AddService
Addservice = FILTER, , SESFILTER.AddService  ; "Filter driver install"

[SESUSB.AddService]
DisplayName    = %SESUSB.SvcDesc%
ServiceType    = 1                  ; SERVICE_KERNEL_DRIVER
StartType      = 3                  ; SERVICE_DEMAND_START
ErrorControl   = 1                  ; SERVICE_ERROR_NORMAL
ServiceBinary  = %12%\SESUSB.SYS
LoadOrderGroup = Base

[SESFILTER.AddService]                        ; "Filter driver install"
DisplayName    = %SESFILTER.SvcDesc%         ; "Filter driver install"
ServiceType    = 1                            ; "Filter driver install"
StartType      = 3                            ; "Filter driver install"
ErrorControl   = 1                            ; "Filter driver install"
ServiceBinary  = %12%\FILTER.SYS             ; "Filter driver install"
LoadOrderGroup = PnP Filter                   ; "Filter driver install"

[SESUSB.Dev.NT.HW]                            ; "Filter driver install"
Addreg=SESFILTER_Filter_Reg                  ; "Filter driver install"

[SESFILTER_Filter_Reg]                        ; "Filter driver install"
HKR,,"UpperFilters",0x00010000,"FILTER"      ; "Filter driver install"

[Strings]
MfgName="Sample Driver"
USB\VID_07CC&PID_0003.DeviceDesc="Sample USB device"
SESUSB.SvcDesc="SESUSB.SYS Sample USB device driver"
SESFILTER.SvcDesc="FILTER.SYS Upper filter driver" ; "Filter driver install"
```

Listing 2
Marking a parameter with [ref]

```
typedef struct SOMESTRUCT
{
    int     i;
    short   s;
} SOMESTRUCT;

interface IOther
{
    HRESULT TellSOMESTRUCT([in] int i, [in] short s);
}

interface IInterface
{
    HRESULT NonNullMethod(  [in, ref] SOMESTRUCT *pss,
                            [in] IOther *pii2);
}
```

Accessing Old List-View Headers

MATTHEW WILSON
matthew@synesis.com.au

THE LIST-VIEW COMMON control, in report mode (window style contains LVS_REPORT), has a header control. This control is accessed via the LVM_GETHEADER message, or the macro `ListView_GetHeader` (which wraps a sending of the LVM_GETHEADER message), which takes no parameters and simply returns the window handle of the header control.

Unfortunately, old versions of the common control library (comctl32.lib) do not handle this message, requiring the following function, `ListView_GetHeaderCtrl()`, which searches for the child header control if the parent list-view does not recognize the LVM_HEADER message.

```
HWND ListView_GetHeaderCtrl(HWND hwnd)
{
#ifndef LVM_GETHEADER
#define LVM_GETHEADER   (LVM_FIRST + 31)
#endif

    /* Attempt the LVM_GETHEADER message */
    HWND    hwndChild = (HWND)SendMessage(hwnd,
      LVM_GETHEADER, 0, 0L);

    if(hwndChild  ==  NULL)
    {
       /* NULL returned so attempt a search */
        HWND    hwndFirst;

        hwndChild = GetWindow(hwnd, GW_CHILD);
        hwndFirst = hwndChild;

        do
        {
            CHAR    szCls[200];

            if( GetClassNameA(hwndChild,
              szCls, sizeof(szCls)) &&
                lstrcmpiA(szCls, WC_HEADERA) == 0)
```

Listing 3
Using the MIDL cpp_quote keyword

```
typedef struct SOMESTRUCT
{
    int     i;
    short   s;
} SOMESTRUCT;

cpp_quote("#ifndef __cplusplus")
typedef SOMESTRUCT    *SOMESTRUCT_ref_t;
cpp_quote("#else")
cpp_quote("typedef SOMESTRUCT   &SOMESTRUCT_ref_t;")
cpp_quote("#endif /* !__cplusplus */")

interface IOther
{
    HRESULT TellSOMESTRUCT([in] int i, [in] short s);
}

cpp_quote("#ifndef __cplusplus")
typedef IOther                    *IOther_ref_t;
cpp_quote("#else")
cpp_quote("typedef IOther  &IOther_ref_t;")
cpp_quote("#endif /* !__cplusplus */")

interface IInterface
{
    HRESULT NonNullMethod(  [in, ref] SOMESTRUCT_ref_t pss,
                            [in] IOther_ref_t pii2);
}
```

```
            {
                /* Found it! */
                break;
            }

        } while((hwndChild =
GetWindow(hwndChild, GW_HWNDNEXT)) != NULL &&
                hwndChild != hwndFirst);
    }

    return hwndChild;
}
```

The function has been compiled with Visual C++ 2.0, 4.0, 4.2, 5.0, and 6.0, and Borland C++ 4.52 and 5.5, and tested on Windows 95, 98, NT 4, and Windows 2000.

Avoiding the MIDL Semantic Analysis Bug

MATTHEW WILSON
matthew@synesis.com.

THE MICROSOFT IDL (MIDL) compiler provides a number of facilities for defining and manipulating types borrowed from C. One of these, `typedef`, is used to create aliases of existing types (or of previously defined aliases), usually for clarity/brevity or for flexibility.

One would declare such types in the following way:

```
typedef ExistingType    NewAliasType;
```

For example, in wtypes.idl the APPID type alias is typedef'd from the type GUID. (In fact GUID is also an alias for the actual type struct _GUID), as in:

```
typedef GUID            APPID;
```

In C/C++ the `ExistingType` may be omitted, in which case the type `int` is assumed. But if the `ExistingType` is an identifier that is unknown to the compiler, then obviously the compilation will fail at that point.

In MIDL, however, the MIDL compiler appears to treat typedefs in the same way as preprocessor symbol definition replacements, since it is possible to have the following compile without errors or warnings (where NOT_DEFINED is not defined):

```
typedef NOT_DEFINED     XYZ_t;
```

This may not seem like a problem, since how often would one define a type in an IDL file and not use it? Well quite often, when building a complex system with a hierarchy of interfaces, types and, consequently, IDL files. In particular, one often defines types in IDL that are only utilized in C/C++ source code. Therefore, this issue can be the source of a number of subtle bugs, causing big headaches when combined with minimal spelling errors. For example:

```
typedef GULD * const    CPGUID;
// Trouble awaits! w::d
```

| Download code › **windevnet.com/wdn/code/** |

Listing 4 Using the IInterface interface

```
// By defining the following classes ...

class SomeStruct
    : public SOMESTRUCT
{
// Construction
    SomeStruct(int i, short s)
    {
        this->i = i;
        this->s = s;
    }

    ...
};

class Class2
    : public IOther
{
    ...
};

// ... one can write validly, from within C++, the following code.

void funcX(IInterface *pii, Class2 &cls2)
{
    ...

    SomeStruct  some(65536, 256);

    pii->NonNullMethod(some, cls2);

    ...
}
```

Map any XML document stored in a database to any .NET class

Mapping SQL Data to Class Properties

AS DISCUSSED IN LAST month's column, the XML serialization process generates an XML representation of the data stored in a .NET Framework class. Only the public fields of the class are taken into account and the whole process aborts if the class holds circular references with other managed classes. The `XmlSerializer` class is the .NET Framework tool that governs the process. You use this class to serialize a living instance of a class to a persistence medium and to recreate an object from a source.

Last month, I briefly hinted at a bunch of events. You can take advantage of them during the deserialization step whenever the input stream contains an XML document that doesn't match the schema of the object being deserialized. The programmer can fix things up and programmatically map the unmatched XML node to a particular combination of fields in the target object using events such as `UnknownNode`. This technique is often used to deserialize across different versions of the class. For example, Version 1.0 of the application saves a class. Next, Version 2.0 of the application that manages a slightly different version of the class needs to deserialize the bytes. Using this approach, the same data can be easily mapped to new, or simply renamed, fields.

Taken to the limit, the feature also proves useful in a more enticing scenario—mapping SQL Server data directly to class instances.

In this article, I'll show how to execute a query that returns XML data and map the various nodes to fields of a predefined class. The process should not be read simply as the deserialization of an instance of the same class. More exactly, the technique discussed here represents a way to map, in total or in part, any XML document stored in a database to any class usable within a .NET application.

Getting XML Data From the Database

Let's prepare a simple query to run against SQL Server that returns XML data. In the latest version, SQL Server supports the FOR XML clause in the SQL's SELECT statement. When used, the clause causes the query processor to pack the result set into an XML document. It goes without saying that any query, and any other database server, can be used as long as it returns text that can be processed as well-formed XML text, or at least as a well-formed XML fragment. (Incidentally, an XML fragment is a well-formed XML document except that it doesn't include a unique root node.)

The following code is at the heart of the example. You call into a method of a worker class and the method executes a SQL XML command. The data flows into the serializer, and an instance of a particular class is returned, as shown here:

```
Employee emp = LoadEmployeeData(empID);
```

Internally, the `LoadEmployeeData` method utilizes the `ExecuteXmlReader` method of ADO.NET's `SqlCommand` class to execute the query and obtain back XML data. In the .NET Framework, XML data is normally worked using a reader that is a cursor-like component, which processes one node at a time. The peculiarity of the `ExecuteXmlReader` method is that it returns the XML data from the query stuffed in a ready-to-use instance of the XML reader class.

```
// cmd here is a SqlCommand object
XmlSerializer ser = PrepareSerializer();
Employee emp = null;
XmlTextReader reader;
reader = (XmlTextReader) cmd.ExecuteXmlReader();
if(ser.CanDeserialize(reader))
     emp = (Employee) ser.Deserialize(reader);
else
     Console.WriteLine("Cannot deserialize");
reader.Close();
```

The XML reader is passed to the `Deserialize` method of the `XmlSerializer` and its content is processed and creates an instance of the specified user class. For the whole mechanism to work, some attributes

DINO ESPOSITO *is Wintellect's ADO.NET and XML expert and is a trainer and consultant based in Rome, Italy. He is a contributing editor to* MSDN Magazine*, writing the "Cutting Edge" column, and is the author of several books for Microsoft Press, including* Building Web Solutions with ASP.NET *and* Applied XML Programming for .NET. *Contact him at dinoe@wintellect.com.*

must be set on the serializer class. These attributes play an essential role to instruct the serializer on how to map nodes in the source XML to fields in the target class. The configuration attributes are defined in the `PrepareSerializer` method.

In general, the attributes of the XML serialization process can be set in two ways—programmatically through an ad hoc set of classes, and declaratively by associating attributes with the serializable members of the class. For example, the declaration to follow states that the `LastName` property must be rendered to XML as a nullable element named `FamilyName`.

```
[XmlElement(Namespace ="urn:dino-e",
  IsNullable=true,
  DataType="string",
  ElementName="FamilyName")]
public string LastName;
```

The same association can be set programmatically by manipulating the corresponding attribute classes such as `XmlElementAttribute`.

A special case of XML serialization is when the data to deserialize is stored in a database. In a similar situation, the document being returned is not necessarily the result of a previous XML serialization that was operated on in the same output class. In the aforementioned code snippet, you deserialize the output of a query to an `Employee` class. However, nothing would guarantee that the contents of the database column is exactly the XML string generated by the `XmlSerializer` for an instance of `Employee`. At the end of the day, XML serialization attributes allow you to deserialize any XML text, no matter the storage medium, to any .NET Framework class. More importantly, the mapping takes place automatically within the `XmlSerializer` class and doesn't even require that you create an instance of target class.

Deserializing From a Database

Let's examine the details of XML data mapping considering the following class, named `Employee`.

```
public class Employee {
          public string FirstName;
          public string LastName;
          public string Position;
          public DateTime Hired;
}
```

You run a SQL query against the Northwind database and make sure you select the fields listed in the following statement:

```
SELECT firstname, lastname, title, hiredate
FROM employees
WHERE employeeid=@empID
FOR XML
```

Since the SELECT statement contains a FOR XML clause, the final XML output will be made of a sequence of XML nodes like the one shown here. You have one of such a node for each record in the result set.

```
<employees firstname="..."
   lastname="..."
   title="..."
   hiredate="..." />
```

As mentioned, the serializer is used only to deserialize the data coming from SQL Server. No previous serialization was explicitly done. The deserializer reads the inbound data and infers an ad hoc class

structure; then, it just matches this inferred structure with the specified type to deserialize to—the Employees class in this case.

The first piece of information you need to pass on to the XML serializer is the name of the class to deserialize to. By default, the serializer assumes that the name of the target class is the root of the XML fragment. In this case, it will be Employees. To change this to Employee, you must define an XmlRoot attribute on the Employee class. This can be done either declaratively or programmatically. You could do it declaratively as shown in the code snippet here:

```
[XmlRoot(ElementName=employees)]
public class Employee {
  :
}
```

If for some reason you don't have access to the source code of the Employee class, then you can proceed programmatically by creating an instance of the XmlRootAttribute class. When you bind attributes programmatically, you must first create an instance of the XmlAttributes class, which represents the whole set of attributes for the XML serializer.

```
XmlAttributes changes = new XmlAttributes();
XmlRootAttribute newRoot = new
XmlRootAttribute();
newRoot.ElementName = "employees";
changes.XmlRoot = newRoot;
```

The XmlRoot property in XmlAttributes is set with an instance of the XmlAttributeRoot class that was previously configured to represent the requested mapping. However, for the whole thing to become effective, the changes must be added to an XmlAttributeOverrides object, which will then be passed to the type-specific serializer's constructor.

```
XmlAttributeOverrides over = new
  XmlAttributeOverrides();
over.Add(typeof(Employee), changes);
XmlSerializer ser = new
  XmlSerializer(typeof(Employee), over);
```

In the original XML source, you have four fields—lastname, firstname, title, and hiredate. The last two have a counterpart in the Employee class with a different name—Position and Hired. The deserializer, though, works in a strictly case-sensitive fashion and considers firstname completely different from FirstName. For this reason, renaming the lastname and firstname properties is absolutely necessary.

```
XmlAttributes changes = new XmlAttributes();
XmlAttributeAttribute fname = new
  XmlAttributeAttribute();
fname.AttributeName = "firstname";
changes.XmlAttribute = fname;
over.Add(typeof(Employee), "FirstName",
  changes);
```

You should note that a distinct XmlAttributes object is required for each element you want to override. The XmlAttributes object collects all the overrides you want to enter on a given element. In this case, after creating a new XmlAttributeAttribute object, you change the attribute name to that of the source SQL field and store the resultant object in the XmlAttribute property of the overriding container. A nearly identical piece of code is needed for the other properties of the Employee class.

It is worth noticing that the root of the XML source must coincide with the name of the class. If this is not the case, then the Deserialize method just fails. If you need to apply some logic during the conversion phase, you can do it using one of the techniques that I discussed in last month's column. In particular, you might want to hook up one of the deserializer's events to be notified of any unknown nodes found. Next, you extract any information from nodes and process it as needed.

Working with Embedded Data

What happens if the query contains embedded data; for example, the results of an INNER JOIN? In this case, the FOR XML clause comes up with data packed as shown here:

```
<employees>
<firstname>Nancy</firstname>
<lastname>Davolio</lastname>
   <orders>
      <orderid>1234</orderid>
      <orderid>5678</orderid>
   </orders>
</employees>
```

How can you import and map the embedded data rooted in the <orders> node? The XML serializer class is always notified of any <orders> elements found along the way through an UnknownElement event. So to import the contents of the <orders> subtree you need to write an event handler. For example, suppose that you want all the orders to populate a collection member (say, Orders) of the Employee class. The following code illustrates how to handle the event.

```
if (e.Element.Name == "orders")
{
         if (emp.Orders == null)
        emp.Orders = new ArrayList();
         int oID = (int) e.Element.InnerText;
         emp.Orders.Add(oID);
}
```

If the collection is NULL, it is created and added to all the various orders. The specific order information is retrieved using the InnerText property of the element. The necessary code is slightly different if the order is expressed as an attribute, as shown here:

```
<orders orderid="1234" />
```

In this case, you rely on the Attributes property of the e.Element object:

```
int oID = (int)
e.Element.Attributes["orderid"].Value;
emp.Orders.Add(oID);
```

The Attributes collection property contains all the attributes on the current node.

More About the XmlSerializer Class

When you run a sample application that uses the XmlSerializer class in Visual Studio .NET, you'll notice that the first time the class is initialized in the session, it takes a while. The reason is that the XmlSerializer needs to compile and then store an ad hoc assembly that it uses to perform the serialization and deserialization of the specified type. When you instantiate the XML serializer, you indicate a type. The initialization of the serializer just consists of the creation of an assembly capable of processing data of that type.

The XmlSerializer class maintains an internal table of type/assembly pairs. If there is no known assembly to handle the type, a new assembly is promptly generated and cached; otherwise, the existing assembly is used to serialize and deserialize. A little known thing, though, is that if you happen to use an XmlSerializer constructor different from the simplest one, the assembly cache is disabled and the assembly is recreated each time the class is instantiated.

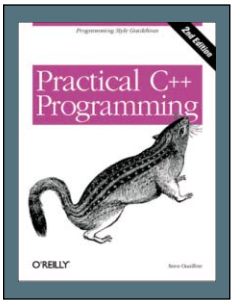
```
Type t = typeof(MyClass);
XmlSerializer ser = new XmlSerializer(t);
```

Of all the constructors available to the serializer, the one just shown is the only one not affected by the aforementioned behavior. If you use another constructor, then be aware of the fact that the internal assembly cache is disabled.

The XML serializer is a double-edged sword. On one hand, it lets you serialize and deserialize even complex .NET Framework classes to and from XML with very few lines of code. To obtain that, though, the serializer creates an assembly on the fly. Unless you use a single global instance of the serializer on a per-type basis, you can easily add hundreds of milliseconds of overhead to each call. Watch your constructor and enjoy serialization! w::d

Practical C++ Programming, 2nd Edition

Steve Oualline
574 pages
O'Reilly & Associates
$39.95
ISBN: 0596004192
www.oreilly.com



PRACTICAL C++ PROGRAMMING, 2ND Edition, by Steve Oualline, updates a classic O'Reilly tutorial. The author sets out quite clearly in the preface that it's designed for people with "no previous programming experience." As a part-time instructor who has taught more than 100 college freshman-level students in this category, my interest in this book was piqued immediately. The book surveys the entire language gamut including control, preprocessor, file I/O, namespaces, pointers, classes, inheritance, templates, and STL.

Practical C++ Programming, 2nd Edition weighs in with only a few more pages than its preceding 1995 edition. A chapter-by-chapter comparison shows that "Portability Problems" was dropped in favor of a new chapter on "Standard Template Library." It also adds a very brief appendix of Internet-based resources.

The author leads off with a chapter on style issues even before introducing any syntax. This seems like an easy way to inculcate readers into the culture of style. Throughout the book, the author broaches various software engineering topics such as structured programming and modular design after an appropriate background.

The pacing is brisk compared to many academic texts I have experienced. For example, Oualline covers arrays (single and multidimensional), strings, C-strings, and prefix/postfix operators altogether in a single brief 25-page chapter. Each chapter poses several questions that are fully answered at the end. Additionally, appropriate programming exercises appear at the end, though solutions to these are unavailable.

After introducing control structures, a similar pacing covers everything related to functions in one chapter: scope, storage class, namespaces, return values, reference/value params, overloading, default args, recursion, and related topics.

Chapter 17 on debugging and optimization brings forth many common C++ aphorisms, such as inline and converting array indices into pointers. On optimization, I tend to agree with Herb Sutter: "Way too many programmers spend way too many hours trying to optimize—but sometimes pessimizing, and always complicating code that doesn't need it." (InformIT.com: "Interview with C++ Expert Herb Sutter," February 1st, 2002.)

The chapters on STL templates are brief and give you mostly the basics to get started with them. Oualline admits that treatment of such areas as partial specialization lies beyond this text. On STL, short examples on set and map containers point out the general direction. For a complete understanding of STL, you'll need another text such as Nicolai Josuttis's *The C++ Standard Library* and/or Scott Meyers's *Effective STL*.

I found it unusual that the do/while loop and the ternary operator were both relegated to the very last chapter, especially since Oualline concedes that most programming is maintenance. Conversely, some less frequently used items such as bit members receive more prominent placement than they deserve.

Online reviewers vary widely in opinions of *Practical C++ Programming*. Novice programmers have proclaimed its excellent readability, syntax explanations, exercises, and pragmatic guidelines. Experts bemoan its lack of emphasis on object-oriented programming through its primary emphasis on procedural techniques. Comparisons to Oualline's *Practical C Programming, 3rd Edition* inevitably follow this line of criticism. Based on my analysis of their respective tables of contents, several chapters are nearly identical between the two texts.

Even so, I've yet to see any book that effectively combines programming for beginners with object-oriented programming. As with most C++ introductions, *Practical C++ Programming* begins discussing objects earnestly around the middle of the book. The author develops a simple stack class from scratch, improving object awareness with each iteration. Along the way, lots of useful tips help the unwary avoid the most common pitfalls, especially for the default member functions (i.e., copy constructor). Some object features, such as friend classes, merely show the usage without enough information for you to make informed decisions about when not to use them.

Chapter 27, "Putting it all Together," provides a complete case-study of design and implementation of a program to provide simple source-code metrics. Since most other examples exist simply to illustrate a language feature in isolation, this project adds more coherence to the overall picture.

Source code for the book is online at www.oreilly.com/catalog/cplus2/. The source archive is quite small but comprehensive. Makefiles for GNU, Borland, and Microsoft appear for every example, even the "Hello World" program. The programs compiled without complaint on VC++ 7.0, though one of the compiler switches is unsupported in VC++ 6.0. The biggest possible improvement would be to include solutions to the Programming Exercises. Last, O'Reilly faithfully reproduces reader-submitted errata lists on its web site. When I submitted a new errata item, the author reviewed and approved it less than a week later. Although the index is detailed, I failed to find an entry for the word "inheritance."

Practical C++ Programming provides a thorough introduction to the basics of C++ including lots of pragmatic advice along the way. I would recommend this book to anyone with a working knowledge of any prior programming language (such as VB, Perl, or JavaScript) who wants to get started quickly on C++. w::d

VICTOR R. VOLKMAN *received a B.S. in Computer Science from Michigan Technological University. He has been a frequent contributor to* Windows Developer Magazine *since 1990. He is the author of C/C++* Treasure Chest*, which includes 300 products on CD-ROM. He can be reached by e-mail at sysop@HAL9K.com or through http://www.HAL9K.com/.*

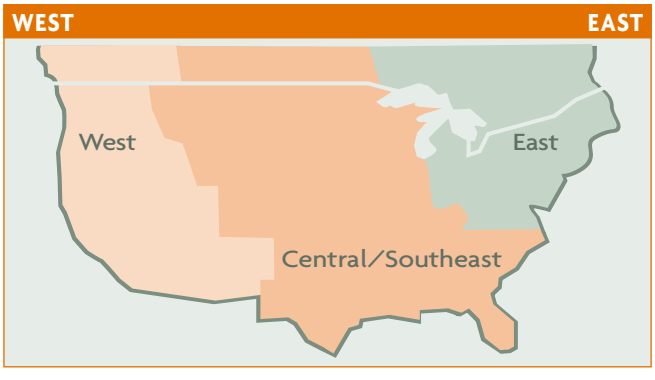
GET ADDITIONAL INFORMATION ABOUT PRODUCTS AND SERVICES YOU SEE ADVERTISED FAST!

PHONE: Contact the vendor directly using the information in the advertisement.

WEB: Go to the development tool page on our web site, www.windevnet.com. From there you can link to the advertisers below.

www.windevnet.com

windows:: developer
APPLICATION DEVELOPMENT FROM WINDOWS TO WEB NETWORK



	Michele Hurabiell	Regional Manager—West
	415-947-6199	mhurabiell@cmp.com
	Ed Day	Regional Manager—Central/Southeast
	785-838-7547	eday@cmp.com
	Jon Hampson	Regional Manager—East
	603-924-8500	jhampson@cmp.com
	Julie Thibault	Account Manager—All Regions
	603-924-8400	jthibault@cmp.com

Submit new product announcements to wdletter@cmp.com.

ThinPrint Announces Free Software Development Kit

ThinPrint has announced the availability of a free Software Development Kit for ThinPrint .print. The new SDK gives developers of web and mobile applications a way to add a complete print system to their environment. ThinPrint .print enables printing over the Internet and high-quality printing from a browser, from any device, including Microsoft Windows Powered Pocket PC and Microsoft Windows Smartphone, which can deliver fully formatted A4 pages to the next available printer. Existing reports can be moved to the mobile Internet without any further modifications. ThinPrint .print SDK is available for download from http://www.thinprint.com/dev (registration required).

ThinPrint GmbH
866.845.7811
www.thinprint.com

SlickEdit Releases Visual SlickEdit v8.0

SlickEdit has released Visual SlickEdit v8.0. Version 8.0 offers increased flexibility with directory-based projects, autoupdated distributed tagging, Secure FTP (SFTP), and Section 508 accessibility for blind and vision-impaired developers. The new three-way merge user interface in v8.0 complements the existing DIFFzilla file and directory tree differencing engine. Version 8.0 provides Borland JBuilder users with advanced editing, debugging, and building capabilities, including automatic setup of build and compile commands. Java developers will benefit from SlickEdit's native Java debugger with its new Edit and Run capabilities and exceptions toolbar. A number of new GNU C/C++ debugger features are provided, including remote debugging, suspend support, and a memory toolbar, along with support for Apache Jakarta Ant for doing builds. And for developers using the CVS open-source, version-control system, v8.0 provides a new graphical user interface to perform the tasks previously done via the command line. Visual SlickEdit pricing starts at $299.00 for a full media kit on Windows and Linux x86 platforms. Single-user Academic pricing is available for students and faculty starting at $99.00.

SlickEdit Inc.
800.934.3348
www.slickedit.com

/n Software Delivers IP*Works! and IP*Works! Zip for the .NET Compact Framework

/n Software has delivered new editions of IP*Works! and IP*Works Zip! for the .NET Compact Framework. IP*Works! is a suite of professional Internet components providing everything needed to write connected applications. IP*Works! contains more than 30 royalty-free components implementing HTTP, FTP, SOAP, SMTP, POP, IMAP, SNMP, NNTP, XML, MIME, Telnet, Multicast, and more. IP*Works! Zip is a suite of development components for adding compression and decompression functionality to web and desktop applications. The current release of IP*Works! Zip allows developers to integrate compression and decompression into applications using the Zip, Tar, Gzip, or Jar standards for compression. The components also feature password support and streaming compression and decompression. IP*Works! and IP*Works! Zip for the .NET Compact Framework are available from the company's web site.

/n Software Inc.
800.225.4190
www.nsoftware.com

Quest Software Presents JClass DesktopViews

Quest Software has presented JClass DesktopViews, a set of components for Java applications and applets. Key features include: 100 percent pure Java JavaBeans; widest breadth and depth for components of Swing development; stable and scalable; and works with the latest Java SDKs and JREs. Included in JClass DesktopViews: JClass Chart—embeds professional charts and graphs in your GUIs; JClass Chart 3D—provides interactive, three-dimensional data presentation; JClass LiveTable—creates professional tables and forms; JClass Field—provides data validation for a range of data types; JClass HiGrid—a front end for database applications; JClass DataSource—a tool for data access; and JClass JarMaster—a way to package and manage your Java classes for rapid download.

Quest Software Inc.
949.754.8000
www.quest.com

Datalight Introduces ROM-DOS 7.1

Datalight has introduced an upgrade to ROM-DOS 7.1, a development kit that allows developers to build Internet-ready embedded systems using as little as a 186 or higher processor, 60K to 90K of ROM or flash, and a packet driver or modem for Internet connectivity. Using the TCP/IP stack within ROM-DOS, applications can have remote system control or alerts delivered via e-mail, pager, or cell phone. ROM-DOS provides a secure web server that allows an embedded system to be viewed and controlled from a web browser. ROM-DOS also provides support for the Internet standard File Transfer Protocol (FTP) with an FTP server. The Remote Console features work with Sun Microsystems' JVM environment to provide remote DOS console access to a remote system. In addition to the publishing of the LFN libraries, Datalight has added support for Berkeley Socket Distribution (BSD). The BSD makes it easy to migrate from any OS to DOS without writing a new program to make the transition. See the company's web site for more details.

NEW PRODUCTS

Datalight Inc.
800.221.6630
www.datalight.com

Celestial Software Releases AbsoluteTelnet

Celestial Software has released AbsoluteTelnet, a software terminal emulator that gives you terminal access to your networked host systems from any Windows PC. AbsoluteTelnet includes the Telnet, SSH, and SSH2 protocols, with port forwarding and X11 forwarding. AbsoluteTelnet is compatible with Linux, Solaris, AIX, HPUX, DCOSx, AT&T, DEC, SGI, Sequent, and FreeBSD. AbsoluteTelnet provides emulations such as VT100, VT220, VT320, XTERM, ANSI, SCO-ANSI, and QNX. File transfers can be accomplished with Zmodem. AbsoluteTelnet also includes SSH capabilities with strong encryption algorithms such as Blowfish, Twofish, AES, Arcfour, 3DES, Cast128, IDEA, and RC4. International character set support including character translation for many Russian, Japanese, Chinese, and Korean character sets is included, in addition to all ISO 8859 character sets and their Windows equivalents. AbsoluteTelnet is available for $29.95.

Celestial Software
www.celestialsoftware.net

DEVELOPERS' MARKETPLACE

Does your Team do more than just track bugs?

Download Your Free Trial at www.alexcorp.com

Alexsys Team does! Alexsys Team 2 is a multi-user Team management system that provides a powerful yet easy way to manage all the members of your team and their tasks - including defect tracking. Use Team right out of the box or tailor it to your needs.

Track all your project tasks ...

- Defects
- New Product Features
- Help Desk Cases
- Action Items
- ISO 9000 Issues and more…

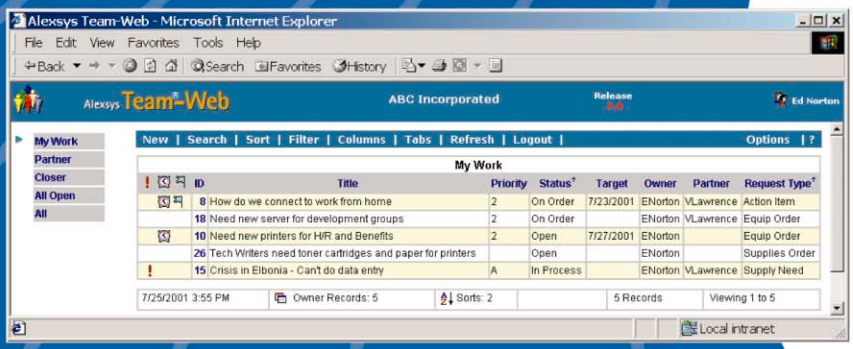
New Version Available
Team v2.5
With Group Access Control

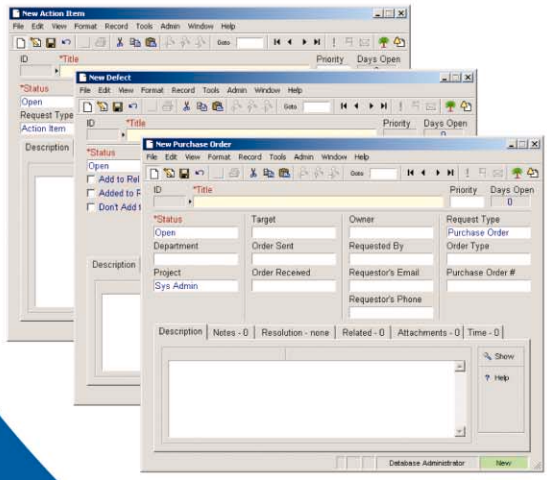
... in one database so you can work together ...

- Keep your project on-track with up-to-date status
- Allow team members to see their assignments and stay on top of the issues
- Find the information you want when you want it
- Run effective meetings using online agendas
- Balance project and team member workloads
- Keep a complete history of your work and more …

... to get projects done.

Alexsys
TEAM 2





Team 2 Features:

- Affordable and scalable
- Multiple work request forms to make data entry a breeze
- Automatic Escalations to stay-on-top of critical tasks
- Knowledge base for you and your customers to find answers fast
- Time recording to manage budgets and improve productivity
- Web Access to work where and when you want
- Secure web forms for customers to get help 24/7

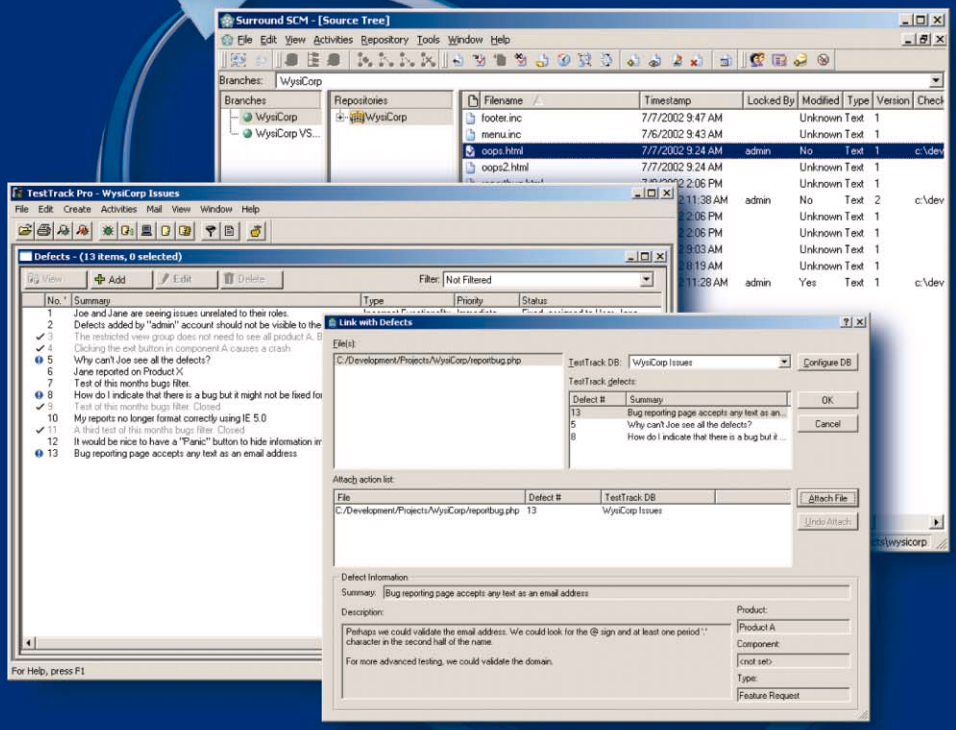
ZDNet Editors' Pick

Download a free, no obligation trial version at www.alexcorp.com.
Need more help, give us a call at 1-888-880-ALEX (2539).

Team 2 works with its own standard database, while Team-SQL works with Microsoft SQL and Oracle Servers.
All versions of Team 2 work on Windows 95/98/Me and Windows NT/2000/XP, Netscape and Microsoft internet browsers.

Manage Change
Manage Defects
Effortlessly

Seapine Software™
changing the world of software development



The ultimate change management solution for busy software development teams

Benefits:

- Comprehensive defect management — track bug reports and change requests, define workflow, customize fields
- Complete source code control with private workspaces, automatic merging, role-based security, and more
- Fast remote access to your source files and defects — work from anywhere
- Advanced branching makes it a breeze to manage multiple versions of your products

Features:

- Scalable and reliable cross-platform, client/server solution supports Windows, Linux, Solaris, and Mac OS X
- Exchange data using XML and ODBC, extend and automate with SOAP support
- Licenses priced to fit your budget
- Seamlessly integrate defect tracking and change management within the Visual Studio IDE

Effective source code control and bug tracking require powerful, flexible, and easy-to-use tools—Surround SCM and TestTrack Pro

Seapine Surround SCM

Seapine TestTrack PRO

Download Surround SCM and TestTrack Pro at www.seapine.com or call 1-888-683-6456

WTL & ATL

WTL IS a windowing framework extension to ATL that attempts (and largely succeeds) to provide an application framework analogous in functionality, but vastly different in bloat, to MFC. It provides many of the challenging-to-write-oneself features such as frames, splitters, command-updates, toolbars, coolbars, and so on. It's very nice stuff (although it has no-more and no-less regard for "proper" C++ practices than does ATL itself), and I strongly recommend you try it out. It's quite hard to actually find out where it lives, but at the time of this writing is was available at http://download.microsoft.com/download/VisualStudioNET/Install/7.0/WXP/EN-US/WTL70.exe.

Although WTL is a Microsoft library, Microsoft does not widely promote it, and the WTL library files take pains to note that Microsoft does not support WTL. I guess this is a commercial decision, since Microsoft is heavily pushing .NET, but it seems a disservice to all those talented C++/COM/ATL/MFC developers out there. Thankfully, a significant core of Win32 developers is using it, and there are resources (such as CodeProject, www.codeproject.com) that are popularizing it. While we must admit (sometimes against our IT-political mores) that C#/.NET does indeed have merit as an application development technology, it is far from a universal solution. A very good example of where one would wish to remain in the tight world of C++ is in shell extensions. All the functionality in the Remote Reboot component is within one 43-KB DLL (and a lot of that is the string resources), with the only dependencies (bar one, which we'll cover shortly) being system DLLs (i.e., that are installed with your system) found on all 95/NT-family systems. The installation program that installs all the Synesis Software Shell Extensions contains only those seven DLLs, and then uses its `DllRegisterServer()` entry points to register them. It could hardly be simpler! (I also use the NSIS installer, http://nsis.sourceforge.net/, which helps keep everything small.)

The one dependency that I mentioned is a run-time one, on ATL.DLL. The shell extensions do not require this DLL to run, but they do utilize it in their registration, since they use the ATL registrar on their embedded .rgs (registry script) files that are created by the ATL wizard. I've never had a user report a failure to install, so ATL.DLL must be pretty widely embedded out there by now, but in testing Remote Reboot on an NT 4 boot on one of my machines, I came across this very problem. ATL.DLL was not installed, so the shell extension would not install. This is something you should be aware of, but you're pretty unlikely to come across it; this is the first time I have since 1999. (Nonetheless, I'm considering whether to remove this and provide my own registration, as other providers, such as Borland, have done in the past.)

The line between ATL and WTL is not a clean one. This presents no serious problem for using the technologies, but it does seem to be a conceptual muddiness. When creating the WTLSTL subproject for the STLSoft libraries, it was kind of arbitrary to select between which components go into WTLSTL and which into ATLSTL. My choice is to have the non-trivial user-interface stuff to go into WTLSTL and all the rest remain within ATLSTL.

In my opinion, Microsoft has made a mistake with its half-hearted promulgation of WTL. I think it should have made WTL a fully supported part of the Visual Studio distribution, on a par with ATL and MFC, including wizards. Perhaps with Visual Studio 8, Microsoft may smell the coffee of all those C++ developers out there who have good reason to eschew .NET for part/all of their developments. These people want development tools and libraries that suit their needs, and I don't think they're going to be going away for a long time to come, no matter how many marketing dollars may get behind the current "let's pretend that writing complex software is trivial, and can be done by people with little training/experience" thrust.

Though not shown, I've used another WTLSTL class, in addition to `SimpleContextMenuHandler<>`, for the help windows on the dialogs. The `SimpleHelpWindow<>` class intercepts the WM_HELP message, and loads a string resource sharing the given control ID, which it then displays in a floating help window, as shown in Figure 7. Using the class is very simple, as can be seen with `COptionsDialog` (Listing 10).

One final note on WTL: Both versions 3.1 and 7.0 of WTL work fine with ATL versions 3.0 and above (Visual C++ 6.0 and 7.0), but do not work with Visual C++ 5.0. For reasons far too weird to go into here, I use Visual C++ 5.0 for the Synesis Software Shell Extensions, and wanted to get WTL 3.1 to work with it also. I'll not go into details here (though I intend to submit a Tech Tip to *WDN* on the subject), but suffice to say that with a very small amount of editing of the WTL headers, it is possible to make it work.

—M.W.

Listing 2 Retrieving a drop handler from IDataObject

```
SYFNCOMDECL DataObject_GetDropHandle(LPDATAOBJECT lpdo, HDROP *phdrop)
{
  HRESULT hr;

  if( lpdo == NULL ||
      phdrop == NULL)
  {
    hr = E_INVALIDARG;
  }
  else
  {
    FORMATETC fe;
    STGMEDIUM sm;

    *phdrop = NULL;

    fe.cfFormat = CF_HDROP;
    fe.ptd      = NULL;
    fe.dwAspect = DVASPECT_CONTENT;
    fe.lindex   = -1;
    fe.tymed    = TYMED_HGLOBAL;

#ifdef __cplusplus
    hr = lpdo->GetData(&fe, &sm);
#else
    hr = lpdo->lpVtbl->GetData(lpdo, &fe, &sm);
#endif /* __cplusplus */

    if(SUCCEEDED(hr))
    {
      *phdrop = (HDROP)sm.hGlobal;

      sm.hGlobal = NULL;

      ReleaseStgMedium(&sm);
    }
  }

  return hr;
}
```

Listing 3 Using WinSTL's drophandle_sequence

```
typedef winstl::basic_drophandle_sequence<TCHAR>
                                    drophandle_sequence_t;


HDROP   hdrop;

. . . // Get the drop handle from the IDataObject

drophandle_sequence_t   paths(hdrop);

if(paths.empty())
{
    ::MessageBox(NULL, _T("Invalid selection" . . .
}
else
{
    std::for_each(paths.begin(), paths.end(),
                  std::back_inserter(m_paths));

    . . .
}
```

Figure 4 The Remote Reboot main dialog

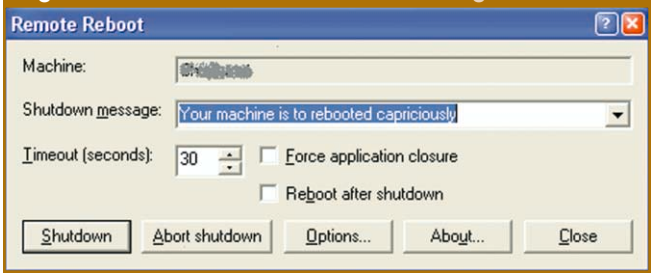


Figure 5 Shutdown failed

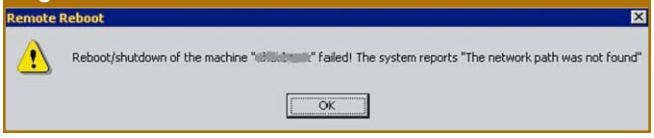


Figure 6 Remote reboot in progress

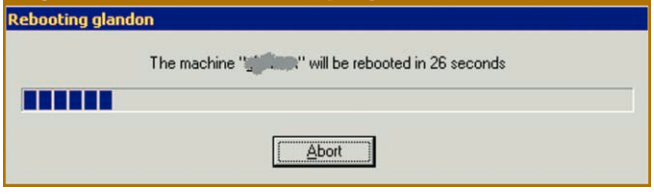
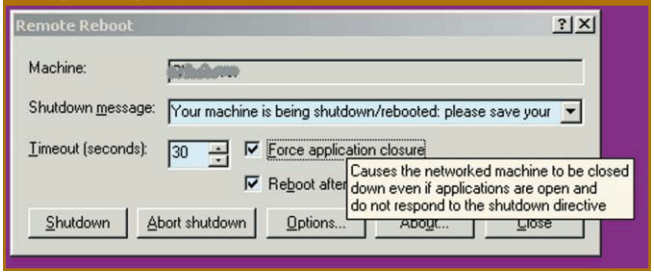


Figure 7 Context help from WTLSTL's SimpleHelpWindow‹›



Listing 4 Tracing clipboard formats with COMSTL's enum_simple_sequence

```
struct trace_FORMATETC
{
public:
  void operator ()(FORMATETC const &fe)
  {
    TCHAR szFmtName[_MAX_PATH + 1];

    if(0 == ::GetClipboardFormatName(fe.cfFormat, szFmtName,
                                     stlsoft_num_elements(szFmtName)))
    {
      if(fe.cfFormat == CF_HDROP)
      {
        // CF_HDROP is built-in, so does not have a registered
        // name
        lstrcpy(szFmtName, _T("Drop handle (HDROP)"));
      }
      else
      {
        lstrcpy(szFmtName, _T("<unknown clipboard format>"));
      }
    }

    ATLTRACE( _T("fmt: %d / 0x%04x / %s, tymed: %d\n")
      , fe.cfFormat
      , fe.cfFormat
      , szFmtName
      , fe.tymed);
  }
};

static void DumpFormats(LPDATAOBJECT lpdobj)
{
  IEnumFORMATETC  *pe;
  HRESULT          hr = lpdobj->EnumFormatEtc(DATADIR_GET, &pe);

  if(SUCCEEDED(hr))
  {
    // Typedef a COMSTL sequence for FORMATETC
    //
    // The template parameters are:
    //
    // a - the COM enumerator interface, in this case
    //         IEnumFORMATETC
    // b - value type, in this case FORMATETC
    // c - value policy type, describes operations to init(),
    //         copy() and clear() instances of FORMATETC
    // d - reference type, how one wishes to access the value
    //         when dereferencing the iterator(s)
    // e - cloning policy, determines whether want forward
    //         iterator or input iterator semantics
    // f - the # of elements retrieved per call to
    //         IEnumFORMATETC::Next()

    typedef comstl::enum_simple_sequence
    /* a */             < IEnumFORMATETC
    /* b */             , FORMATETC
    /* c */             , FORMATETC_policy
    /* d */             , FORMATETC const &
    /* e */             , forward_cloning_policy<IEnumFORMATETC>
    /* f */             , 10
                        >     enum_sequence_t;

    enum_sequence_t   fmts(pe, true);  // true: swallow ref

    std::for_each(fmts.begin(), fmts.end(), trace_FORMATETC());
  }
}
```

Listing 8 Shutdown methods

```
void CRemoteReboot::OnShutdownServer()
{
  ShutdownServer(false, IsKeyPressedAsync(VK_SHIFT), NULL);
}

void CRemoteReboot::OnRebootServer()
{
  ShutdownServer(true, IsKeyPressedAsync(VK_SHIFT), NULL);
}

void CRemoteReboot::OnReboot()
{
  CRebootDialog(m_szHost).DoModal();
}

// Helpers

void CRemoteReboot::ShutdownServer( bool    bReboot,
                                    bool    bForce,
                                    LPCTSTR pcszMessage)
{
  DWORD   dwTimeout;
  TCHAR   szMessage[256];
  LPCTSTR timeoutName = bReboot
                          ? _T("DefaultRebootTimeout")
                          : _T("DefaultShutdownTimeout");

    // Load the appropriate timeout, or use 30s otherwise
  if(ERROR_SUCCESS != Reg_QueryDWordValue(  NULL,
                                            s_szRegKey,
                                            timeoutName,
                                            dwTimeout))
  {
    dwTimeout = 30;
  }

    // Load the appropriate message, or use defaults otherwise
  if( NULL == pcszMessage ||
      0 == *pcszMessage)
  {
    int     cchMessage  = stlsoft_num_elements(szMessage);
    LPCTSTR msgName     = bReboot
                            ? _T("DefaultRebootMessage-tx")
                            : _T("DefaultShutdownMessage-tx");

    if(ERROR_SUCCESS != Reg_QuerySzValue(   NULL,
                                            s_szRegKey,
                                            msgName,
                                            szMessage,
                                            cchMessage))
    {
      if(0 == ::LoadString( _Module.GetResourceInstance(),
                            bReboot
                                ? IDS_DEFAULT_REBOOT
                                : IDS_DEFAULT_SHUTDOWN,
                            szMessage,
                            cchMessage))
      {
        szMessage[0] = _T('\0');
      }
    }

    pcszMessage = szMessage;
  }

  // Initiatate the system shutdown ...
  if(::InitiateSystemShutdown(  m_szHost,
                                const_cast<LPTSTR>(pcszMessage),
                                dwTimeout,
                                bForce,
                                bReboot))
  {
    // ... and show the pending dialog, or
    CPendingDialog(m_szHost, dwTimeout, bForce, bReboot).DoModal();
  }
  else
  {
    // ... explain why it failed
    TCHAR szErr[512];

    FormatMessage(::GetLastError(), szErr, stlsoft_num_elements(szErr));

    MessageBox_printf(_Module.GetResourceInstance(),
                      NULL,
                      MAKEINTRESOURCE(IDS_SHUTDOWN_FAIL),
                      MAKEINTRESOURCE(IDS_PROJNAME),
                      MB_OK | MB_ICONEXCLAMATION,
                      m_szHost,
                      szErr);
  }
}
```

Listing 10 Using WTLSTL's SimpleHelpWindow<>

```
class COptionsDialog
   : public CDialogImpl<COptionsDialog>
   , public SimpleHelpWindow<COptionsDialog, true>
{
   typedef SimpleHelpWindow<COptionsDialog, true>  help_window_class_type;

public:
   enum { IDD = IDD_OPTIONS };

BEGIN_MSG_MAP(COptionsDialog)
   MESSAGE_HANDLER(WM_INITDIALOG, OnInitDialog)
   MESSAGE_HANDLER(WM_SYSCOMMAND, OnSysCommand)
   COMMAND_ID_HANDLER(IDOK, OnCloseCmd)
   COMMAND_ID_HANDLER(IDCANCEL, OnCloseCmd)
   CHAIN_MSG_MAP(help_window_class_type)
END_MSG_MAP()

// Message handlers
private:
   LRESULT OnInitDialog(UINT uMsg, . . .);
   LRESULT OnSysCommand(UINT uMsg, . . .);
   LRESULT OnCloseCmd(WORD wNotifyCode, . . .);

   . . .
};
```