CMP
United Business Media

Distributed Computing

Book Review: **Inside Windows Server 2003**

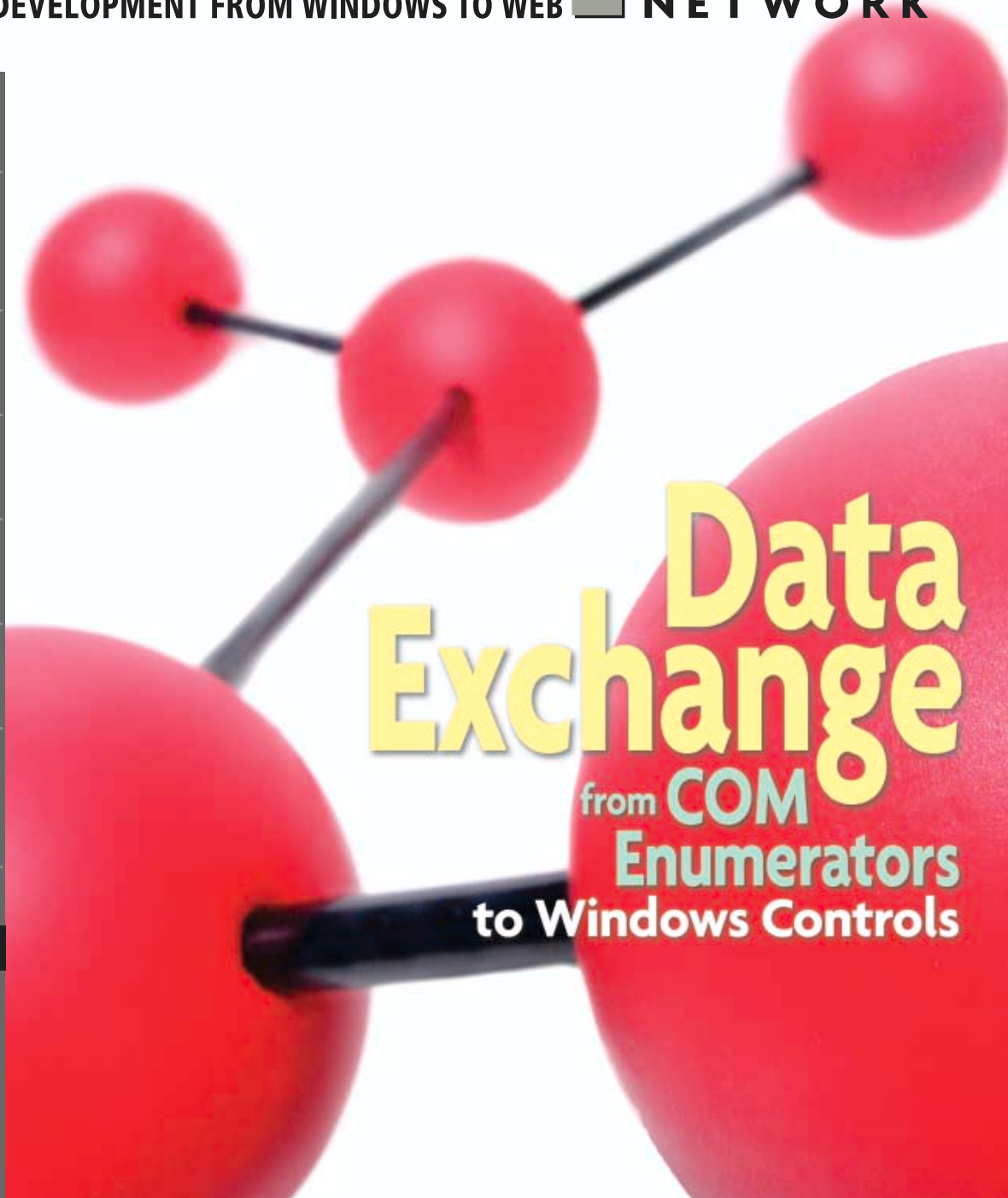
NOV
2003

windows developer

APPLICATION DEVELOPMENT FROM WINDOWS TO WEB NETWORK

Volume 2 / No. 11

www.**windevnet**.com



Data Exchange from COM Enumerators to Windows Controls

Windows Developer Network Special Bonus Issue:
Language Performance Benchmarking

C# has won early praise from some developers for its efficiency and ease-of-use. But is it ready for high-performance software development? Can C# keep up with compiled languages like C, C++, and D or byte-code based Java?

This special issue examines some basic features common to all the languages involved and quantitatively compares the performance of C# and its libraries against the languages and facilities of C, C++, D, and Java.

DOWNLOAD TODAY:

www.windevnet.com/wdn/webextra/2003/0313/

The 2003 Borland® Conference

Accelerate your development.

Register now for the premier event for technical education.

With more than 200 technical sessions focused on Java™/J2EE,™ Microsoft® .NET Framework and Windows,® Web Services, agile processes, and mobile application development — the 2003 Borland Conference is sure to help you advance your mastery of the technologies impacting enterprise-class development today. Borland — helping the whole development team make better software, faster.

Special 50% discount on select products for attendees.*

November 1–5, 2003
McEnery Convention Center
San Jose, California

For complete conference details and session information:

connect.borland.com/borcon03

Borland®
Excellence Endures™

DEFINE····· CaliberRM™

DESIGN···· Together®

DEVELOP····· JBuilder,® Delphi,™ C#Builder,™ C++Builder,® Kylix™

TEST····· Optimizeit™

DEPLOY····· Borland® Enterprise Server, AppServer,™ InterBase,® JDataStore™

MANAGE···· StarTeam®

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, 4601 W 6th St, Ste B, Lawrence, KS 66049, 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/

windows::developer
APPLICATION DEVELOPMENT FROM WINDOWS TO WEB NETWORK

PUBLISHER Kerry Gates
EDITOR IN CHIEF John Dorsey

EDITORIAL

MANAGING EDITOR **Amy Stephens**

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

EDITORIAL ADVISORY BOARD **Mark Baker, 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 785-838-7500, or write *Windows Developer Network*, 4601 West 6th Street, Suite B, Lawrence, KS 66049 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 **Bill Amstutz**
SENIOR V.P., HUMAN RESOURCES **Leah Landro**
VICE PRESIDENT AND GENERAL COUNSEL **Sandra Grayson**

MARKET GROUPS
PRESIDENT, TECHNOLOGY SOLUTIONS **Robert Faletra**
PRESIDENT, HEALTHCARE MEDIA **Vicki Masseria**
V.P., GROUP PUBLISHER APPLIED TECHNOLOGIES **Philip Chapnick**
V.P., GROUP PUBLISHER INFORMATIONWEEK MEDIA NETWORK **Michael Friedenberg**
V.P., GROUP PUBLISHER ELECTRONICS **Paul Miller**
V.P., GROUP PUBLISHER NETWORK COMPUTING MEDIA NETWORK **Fritz Nelson**
V.P., GROUP PUBLISHER SOFTWARE DEVELOPMENT MEDIA **Peter Westerman**
CORPORATE DIRECTOR, AUDIENCE DEVELOPMENT **Shannon Aronson**
CORPORATE DIRECTOR, AUDIENCE DEVELOPMENT **Michael Zane**
CORPORATE DIRECTOR, PUBLISHING SERVICES **Marie Myers**

CMP
United Business Media

CONTENTS

NOV 2003

windows::developer NETWORK VOL 2 • NO. 11

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:

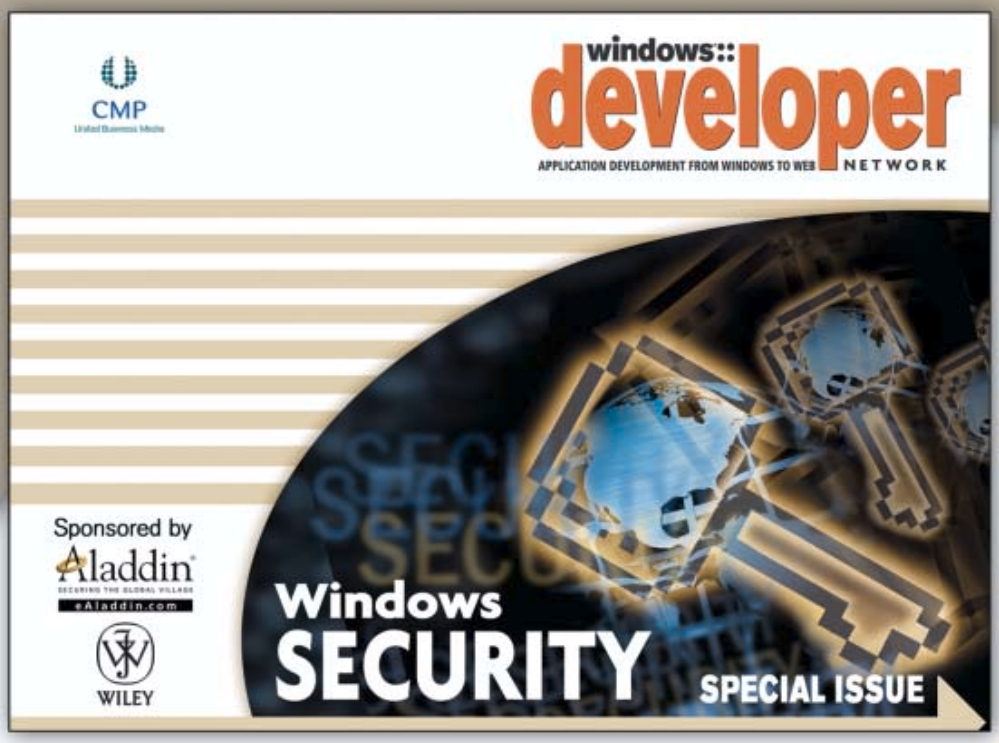
- Sidebar and additional Listing for "Data Transfer, COM Enums, and Windows Controls"
- Additional Listings for "Connect Office Apps to .NET with "Lite" Web Services"

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

COMING NEXT MONTH: **.NET DEVELOPMENT**

Special Security E-Book

Security is an ever-present challenge for developers. To help keep you up to date on security issues and solutions, CMP Media is proud to present a series of articles focusing on security for Windows that have appeared in these leading programming publications: Windows Developer Network, MSDN Magazine, Dr. Dobb's Journal, and C/C++ Users Journal, in one E-book.



Feature articles inside Windows Security E-Book:

Testing for Software Security
Dr. Dobb's Journal Herbert H. Thompson and James A. Whittaker

Defend Your Code with Top Ten Security Tips Every Developer Must Know
MSDN Magazine Michael Howard and Keith Brown

Implementing an SSL/TLS-Enabled Client/Server on Windows Using GSS API
C/C++ Users Journal Alen Talibov

Win32 Security in Managed C++
Windows Developer Network Matthew Wilson

Analyzing the Mescaline Worm / Backdoors Can Damage Trust
Windevnet.com E-Newsletters Jason Coombs

Download today:

www.windevnet.com/wdn/webextra/2003/0314/

DISTRIBUTED COMPUTING HAS BEEN big news of late, but unfortunately for the wrong reasons—throughout August and September, security attacks designed to exploit the Windows DCOM RPC interface were often front-page headlines. The outbreak of the latest W32.Blaster worm was followed closely by the W32.soBig.F worm, which flooded inboxes with so much e-mail that thousands of people simply abandoned them and changed e-mail addresses. As a result of these and similar attacks, August set a record as "the worst month for digital damage," according to mi2g (http://mi2g.com/), a digital risk management firm. mi2g estimates the monthly bill for the combined economic impact of all computer security attacks worldwide to be $32.8 billion.

The W32.Blaster hit unpatched Windows 2000 and Windows XP machines hardest. Unpatched Windows NT and Windows 2003 Server machines were infected, too, but the worm couldn't replicate on those systems. The Windows ME and Win 9x platforms weren't affected by Blaster. (Ironically, the Microsoft web site pages that detail the vulnerabilities of Windows' latest versions also strongly encourage users of Win 9x OSes to upgrade.)

Although Microsoft posted security warnings and systems patches for both Blaster and soBig before widespread infections occurred, a sufficient number of machines remained unpatched

Although Microsoft posted security warnings and systems patches for both Blaster and soBig before widespread infections occurred, a sufficient number of machines remained unpatched. According to security experts, soBig exploited a handful of unpatched Windows 2000 machines running on broadband cable modems whose owners were unaware of the harm caused by their machines. This scenario has Microsoft contemplating default system updates. But automatic system patching has problems, too. Not all machines are unpatched due to inattentive owners—some system administrators are unable to update without causing conflicts with earlier OS fixes or hobbling incompatible, mission-critical software. Catch-22.

Whether the OS updates are distributed automatically or not, the process will always be reactionary, compensating for new exploits of existing systems. Usually, experts interested in improving security find security holes, and fixes can be created before an attack is released in the wild. But the lag time between discovery of a hole and its widespread exploitation may not always be weeks or even days. It's always possible that the bad guys could find the weakness first. And aside the multibillion-dollar estimated cost, the trouble caused in August rated as not more than a major nuisance. A truly malicious attack that cascades through entire networks could have far-reaching economic repercussions. My advice: Keep that copy of Windows ME booted up on your boneyard PC.

If you'd like to learn more about the state of security in Windows development, check out our special issue on Windows Security. This is a collection of articles from some of CMP Media's top technology publications, including *Dr. Dobb's Journal*, *MSDN Magazine*, *C/C++ User's Journal*, and, of course, *Windows Developer Network*. The special issue is available at http://www.windevnet.com/wdn/webextra/2003/0314/.

John Dorsey
Editor in Chief
wdeditor@cmp.com

Table of Contents

MATTHEW WILSON

Data Transfer, COM Enums, and Windows Controls

Two strategies using C and STL for data exchange from COM enumerators to Windows controls

THESE DAYS, SOFTWARE ENGINEERING involves a lot of flexibility. We have different programming paradigms, component object models, co-operating languages, even different binary standards. Many Windows-based developers would probably say that component software development began with COM. Others might say DLLs (and exported functions). A few might even say Windows controls. In a way, they're all right.

One of the challenges of working within even modern frameworks is the cost of development and maintenance time (not to mention run time) in handling all this flexibility. Several years ago, in order to obviate these costs, I wrote a C-API for data exchange between COM enumerators (instances implementing one or more of the `IEnumXXXX` family of interfaces) and Windows controls. I've updated part of that API to present here.

C++ development has recently (and wisely) swung towards the STL model of compile-time development. Whilst STL is no real help in binary interoperability, it is extremely useful for the implementation of binary interoperable components (and for pretty much every other kind of C++ software development that you can think of). STL has become extremely popular because it supports reuse (in a lightweight manner—not more hefty hierarchies), maintainability, and very importantly, efficiency (or at least it does when used correctly). I've embraced STL in a big way: I have created the STLSoft project (http://stlsoft.org/) over the last couple of years in order to apply STL techniques to a broad range of technologies and operating systems (including Win32, UNIX, ATL, MFC, COM, WTL, XML, and .NET) in a robust and efficient manner. Two of the STLSoft subprojects, COMSTL (http://comstl.org/) and WinSTL (http://winstl.org/), contain components for manipulating COM objects and windows controls. I will show how collaboration between these components can provide an interesting alternative to the data exchange C-API.

C-API in C

Let's first look at the C-API. C++ aficionados might well wonder why anyone would bother writing a COM API in C. There are two answers. First, I like to write in C sometimes because it keeps the true understanding of the mechanics of COM to the fore in the brain. (Anyone who thinks that one can just use frameworks and never get into the guts has not written any real COM programs.) The second is that, when creating software that may be reused (in source form as well as binary), it is a good idea not to disenfranchise those developers who still prefer to exist in the C world. (Also, C affords you the opportunity to change the contents of the vtable at run time. It's very useful when you need it, but to be used *very* sparingly, and is not for the faint hearted.)

These observations notwithstanding, I acknowledge the masochism required to create such an API in C, and in your reading the code presented here. I would suggest that you persevere, mainly for the illumination previously alluded to.

Listing 1 shows the API flags, typedefs, structures, and function declarations. Skipping over what looks like a rather complex set of structures for the moment, we see there are two functions. The second one of these, `ListExchange_PutItems()`, has a manageable three arguments. They represent the window handle of the control, the enumerator interface, and flags, respectively. Hence, to populate, say, a list box with the contents of an `IEnumVARIANT`, one would call:

```
ListExchange_PutItems(hwndListBox, penVar,
SYLXF_RESETCONTENT);
```

The `SYLXF_RESETCONTENT` flag causes the list control to be emptied prior to populating with the enumerator contents. Hence, you can populate a list control with the contents of several enumerators, thus:

```
ListExchange_PutItems(hwndComboBox,
    penVar1, SYLXF_RESETCONTENT);
ListExchange_PutItems(hwndComboBox,
    penOleStr, 0);
ListExchange_PutItems(hwndComboBox,
    penBSTR, 0);
```

That seems nice and straightforward. But how does it work? Naturally, the answer relates to the structures and the other function. In fact, if we look at Listing 2, we can see that `ListExchange_PutItems()` is actually implemented in terms of `ListExchange_PutItems_Base()`. Arrays of `EnumeratorHandler` structures and `ControlHandler` structures are defined and passed to `ListExchange_PutItems_Base()`. These structures describe the COM enumerators and windows controls that will be understood by the API. You can see that `ListExchange_PutItems()` understands the `IEnumString`, `IEnumBSTR`, `IEnumVARIANT`, `IEnumGUID`, and `IEnumUnknown` interfaces and the listbox, combo-box, list-view, and (multiline) edit controls that, for most requirements, more than suffices. (Note that there are two `IEnumBSTR`

MATTHEW WILSON *is a software development consultant for Synesis Software, specializing in robustness and performance in C, C++, C#, and Java. Matthew is the author of the STLSoft libraries, and the forthcoming book* Imperfect C++ *(to be published by Addison-Wesley, 2004). He can be contacted via matthew @synesis.com.au or at http://stlsoft.org/.*

and IEnumGUID interfaces handled. This is because this API was defined and used with those Synesis-defined interfaces before the Microsoft interfaces of the same name were released; hence the need to support both. However, this dual support causes no problems since the interfaces are identically defined.)

The handlers are reasonably straightforward. The control handler contains the window class (an identifying type that all windows will provide via the GetClassName() API function), and functions to add an item and to reset the contents. The enumerator handler contains the Interface Identifier (IID) and functions to reset the enumerator, acquire an item's value (which it does via calling the Next() interface method), and to release an item's value. Because the functions operate with the Enum2WndItem item union and have an opaque handle (void*) for the value type (both defined in Listing 1), the API can be expanded to handle any enumerated value type simply by appropriately defining matching handler functions.

You may call ListExchange_PutItems() and use the predefined handlers, or set up your own handler arrays and directly call ListExchange_PutItems_Base(), which is where all the action happens (see Listing 3). The first few lines validate the arguments, followed by the first task of identifying the type of the window. The function iterates through the array looking for a matching control handler. If it fails to find one, then no further processing is done and the function returns a fail code (and writes to the thread's error object). Next is the test for the enumerator. If no enumerator is specified, then the control is simply emptied. It's not exactly orthogonal programming, but I just found it a real convenience (see Listing 6) to be able to clear the contents of any list control, no matter what its type, so this feature

Listing 1 Definitions of the ListExchange structures and functions

```
/* ////////////////////////////////////////////////////////////
 * ...
 *
 * Extract from MOCtlFns.h
 *
 * Copyright (C) 1998-2003, Synesis Software Pty Ltd.
 * (Licensed under the Synesis Software Standard Public License:
 *  http://www.synesis.com.au/licenses/ssspl.html)
 *
 * ...
 * ////////////////////////////////////////////////////////////// */

/* Flags */

#define SYLXF_RESETCONTENT  (0x00000001) /* Empties control first */

/* Typedefs */

typedef union _Enum2WndItem
{
  LPOLESTR    olestr; /* OLE string */
  BSTR        bstr; /* COM BSTR */
  VARIANT     var;  /* COM VARIANT type */
  GUID        guid; /* GUID */
  LPUNKNOWN   punk; /* Object */
  LPDISPATCH  pdisp;  /* Automation object */
  void        *pv;  /* any other type */

} Enum2WndItem;
__SyPtrType1(Enum2WndItem)

typedef HRESULT (*PFnEnum_Reset)(   LPVOID          itf);

typedef HRESULT (*PFnEnum_NextItem)(LPVOID          itf,
                                    LPEnum2WndItem  item,
                                    LPVOID          *pvalue);

typedef void (*PFnEnum_ClearItem)(  LPVOID          itf,
                                    LPEnum2WndItem  item,
                                    LPVOID          value);

typedef struct _EnumeratorHandler
{
  REFIID            iid;
  PFnEnum_Reset     pfnReset;
  PFnEnum_NextItem  pfnNextItem;
  PFnEnum_ClearItem pfnClearItem;

} EnumeratorHandler;

typedef HRESULT (*PFnCtrl_PutItem)( HWND      hwndCtrl,
                                    LPVOID    str,
                                    UInt32    flags,
                                    UInt32    param);

typedef void (*PFnCtrl_Reset)(      HWND  hwndCtrl);

typedef struct _ControlHandler
{
  char const      *className;
  PFnCtrl_PutItem pfnPutItem;
  PFnCtrl_Reset   pfnReset;

} ControlHandler;

/* Functions */

SInt32 ListExchange_PutItems_Base(HWND                      hwndList,
                                  LPUNKNOWN                 punkEnumItems,
                                  UInt32                    flags,
                                  UInt32                    param,
                                  EnumeratorHandler const *enumerators,
                                  UInt32                    cEnumerators,
                                  ControlHandler const    *controls,
                                  UInt32                    cControls);

SInt32 ListExchange_PutItems( HWND      hwndList,
                              LPUNKNOWN punkEnumItems,
                              UInt32    flags);
```

Listing 2 Implemenation of ListExchange_PutItems()

```
/* ////////////////////////////////////////////////////////////
 * ...
 *
 * Extract from MOCtlFns.c
 *
 * Copyright (C) 1998-2003, Synesis Software Pty Ltd.
 * (Licensed under the Synesis Software Standard Public License:
 *  http://www.synesis.com.au/licenses/ssspl.html)
 *
 * ...
 * ////////////////////////////////////////////////////////////// */

SInt32 ListExchange_PutItems( HWND      hwndList,
                              LPUNKNOWN punkEnumItems,
                              UInt32    flags)
{
  static const IID IID_IEnumBSTR_Synesis  = { . . . };
  static const IID IID_IEnumBSTR_MS       = { . . . };
  static const IID IID_IEnumGUID_Synesis  = { . . . };
  static const IID IID_IEnumGUID_MS       = { . . . };

  static const EnumeratorHandler  enumerators[] =
  {
      {   &IID_IEnumString,     EnumXXXX_Reset
        , EnumString_NextItem,  EnumString_ClearItem  }
    , { &IID_IEnumBSTR_Synesis, EnumXXXX_Reset
        , EnumBSTR_NextItem,    EnumBSTR_ClearItem    }
    , { &IID_IEnumBSTR_MS,      EnumXXXX_Reset
        , EnumBSTR_NextItem,    EnumBSTR_ClearItem    }
    , { &IID_IEnumVARIANT,      EnumXXXX_Reset
        , EnumVARIANT_NextItem, EnumVARIANT_ClearItem }
    , { &IID_IEnumGUID_Synesis, EnumXXXX_Reset
        , EnumGUID_NextItem,    EnumGUID_ClearItem    }
    , { &IID_IEnumGUID_MS,      EnumXXXX_Reset
        , EnumGUID_NextItem,    EnumGUID_ClearItem    }
    , { &IID_IEnumUnknown,      EnumXXXX_Reset
        , EnumUnknown_NextItem, EnumUnknown_ClearItem }
  };
  static const ControlHandler   controls[] =
  {
      { "LISTBOX",         ListBox_PutItem,  ListBox_Reset   }
    , { "COMBOBOX",        ComboBox_PutItem, ComboBox_Reset  }
    , { "SysListView32",   ListView_PutItem, ListView_Reset  }
    , { "EDIT",            Edit_PutItem,     Edit_Reset      }
  };

  return ListExchange_PutItems_Base(hwndList, punkEnumItems,
          flags, 0, enumerators, NUM_ELEMENTS(enumerators),
          controls, NUM_ELEMENTS(controls));
}
```

Listing 3 Implemenation of ListExchange_PutItems_Base()

```
/* //////////////////////////////////////////////////////////////
 * ...
 *
 * Extract from MOCtlFns.c
 *
 * Copyright (C) 1998-2003, Synesis Software Pty Ltd.
 * (Licensed under the Synesis Software Standard Public License:
 *  http://www.synesis.com.au/licenses/ssspl.html)
 *
 * ...
 * ////////////////////////////////////////////////////////////// */

SInt32 ListExchange_PutItems_Base(
        HWND                        hwndList,
        LPUNKNOWN                   punkEnumItems,
        UInt32                      flags,
        UInt32                      param,
        EnumeratorHandler const *enumerators,
        UInt32                      cEnumerators,
        ControlHandler const    *controls,
        UInt32                      cControls)
{
  HRESULT hr;

  _ASSERTE(IsWindow(hwndList));

  if( !(flags & SYLXF_RESETCONTENT) &&
      ( NULL == punkEnumItems ||
        NULL == enumerators ||
        0 == cEnumerators))
  {
    hr = E_INVALIDARG;
  }
  else if(!IsWindow(hwndList) ||
          0 == cControls ||
          NULL == controls)
  {
    hr = E_INVALIDARG;
  }
  else
  {
    LPUNKNOWN punkEnum  = NULL;
    UInt32    i;

    /* Find the window class */
    for(hr = E_FAIL, i = 0; FAILED(hr) && i < cControls; ++i)
    {
      if(IsWindowClass(hwndList, controls[i].className))
      {
        hr = S_OK;
        break;
      }
    }

    if(FAILED(hr))
    {
      ErrorInfo_SetDescW(L"Window not of recognised class");

      hr = E_INVALIDARG;
    }
    else
    {
      ControlHandler const  *control  = &controls[i];

      if(NULL == punkEnumItems)
      {
        /* Only want to clear the window */

        _ASSERTE((flags & SYLXF_RESETCONTENT) == SYLXF_RESETCONTENT);

        control->pfnReset(hwndList);
      }
      else
      {
        /* Find the interface */
        for(hr = E_NOINTERFACE, i = 0; i < cEnumerators; ++i)
        {
#ifdef __cplusplus
          hr = punkEnumItems->QueryInterface(
#else
          hr = punkEnumItems->lpVtbl->QueryInterface( punkEnumItems,
#endif /* __cplusplus */
                                      enumerators[i].iid,
                                      (LPPVoid)(&punkEnum));

          if(SUCCEEDED(hr))
          {
            break;
          }
        }

        if(FAILED(hr))
        {
          ErrorInfo_SetDescW(L"Enumerator not of recognised type");

          hr = E_INVALIDARG;
        }
        else
        {
          EnumeratorHandler const *enumerator = &enumerators[i];

          /* Erase the existing content */
          if((flags & SYLXF_RESETCONTENT) == SYLXF_RESETCONTENT)
          {
            control->pfnReset(hwndList);
          }

          /* Now enumerate over the sequence, inserting as we go. */
          for(hr = S_OK, enumerator->pfnReset(punkEnum); SUCCEEDED(hr); )
          {
            Enum2WndItem  item;
            LPVOID      value;

            hr  = enumerator->pfnNextItem(punkEnum, &item, &value);

            if(hr == S_OK)
            {
              /* Add to the control ... */
              hr = control->pfnPutItem(hwndList, value, flags);

              /* ... and release any resources from enumerator */
              enumerator->pfnClearItem(punkEnum, &item, value);
            }
            else if(hr == S_FALSE)
            {
              hr = S_OK;
              break;
            }
          }

#ifdef __cplusplus
          punkEnum->Release();
#else
          punkEnum->lpVtbl->Release(punkEnum);
#endif /* __cplusplus */
        }
      }
    }
  }

  return hr;
}
```

creep was let by. If an enumerator is specified, then it is tested against the enumerator handler information. If recognized, the control is cleared when requested—note that this is not done earlier since a failure in a specified enumerator would change the list without populating it, and it is not generally a good idea to have a failing API effect changes if it can be avoided—and then the enumeration is conducted.

Enumeration begins with resetting the enumeration point. The enumerator handler's `pfnReset` function is passed the enumerator interface pointer, which calls the `IEnumXxxx::Reset()` method on the interface. Since all `IEnumXxxx` enumerator interfaces have the same general definition, and identical `Reset()` and `Skip()` method definitions, we can simply define one handler, `EnumXXXX_Reset()`, to be used for all interfaces, as can be seen in Listing 4; available online (the choice to cast to `IEnumString` is arbitrary). Inside the loop, the enumerator's `Next()` method is called within the enumerator handler's `pfnNextItem` function, passing in the instances of `Enum2WndItem` and `LPVOID`. If the function call returns `S_OK`, then the value is passed to the control handler's `pfnPutItem` function, and is thus added to the control. The value is then cleared (with `pfnClearItem`), so that any allocation in the enumerator handler's `pfnNextItem` function can be released. If the value returned from `Next()` is `S_FALSE`, then there are no more items available (and the return code is changed to `S_OK` before breaking out of the loop because `S_FALSE` is not meaningful to callers of the API). Any error return codes will cause the loop to terminate and will be passed back to the caller.

Programmer's Paradise®

Your best source for software development tools!

iTech Logging 2 for Windows *by iTech Software*

Build, test and debug your Windows, Java and .NET applications faster and more efficiently by adding 2nd generation logging, monitoring and analyzing capabilities. iTech Logging is an integrated solution to trace, watch and analyze your apps at any time and track down errors in your code—even after deployment.

www.programmersparadise.com/itech Paradise # ITA 0100 **$436.99**

XMLSPY 2004 NEW!

by Altova

XMLSPY 2004 is the industry standard XML Development Environment for designing, editing and debugging enterprise-class applications. Now including:

- Visual Studio® .NET Integration
- XML-to-XML Mapping
- XML Differencing

XMLSPY 2004 is the ultimate productivity enhancer for J2EE, .NET and database developers.

Enterprise Edition
Paradise # IOD 017T
$965.99

Professional Edition
Paradise # IOD 017U
$389.99

www.programmersparadise.com/altova

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

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

Paradise # E75 0311
$869.99*

** Price after mfr's mail-in rebate. New US/Can licenses only. Expires 11/30/05.*

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

XML Object Link

by Rogue Wave

Generate C++ Code From XML Schemas!

Many organizations depend on business-critical C++ applications. XML Object Link is a high-performance solution that maps XML documents to C++ and vice versa, quickly enabling C++ applications to communicate using standard XML. XML Object Link increases productivity and reduces development workload for low-level XML programming. Designed for use by developers, support personnel and even end users.

License packs available.

Paradise # R14 0224
$1,136.99

www.programmersparadise.com/roguewave

c-tree Plus®

by FairCom

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.

NEW! SQL Support!

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.
programmersparadise.com

Enterprise Edition
Paradise # T79 0214
$1,495.99

Professional Edition
Paradise # T79 0215
$729.99

www.programmersparadise.com/theimagingsource

SourceOffSite Classic

by SourceGear Corp.

Access SourceSafe™ over the Internet

New Version! 3.5.3

Visual SourceSafe collaboration tool for distributed teams.

- IDE integration
- Over 10 times faster than RAS
- Efficient—uses data compression
- Secure—up to 128-bit data encryption
- Familiar—the look and feel of VSS
- Cross-Platform—Windows and UNIX.

† Classic edition plus Media, 1 user.

Paradise # S0Z 041C
$223.99†

www.programmersparadise.com/sourcegear

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!
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, 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 5, Standard Edition

by Sun Microsystems

Sun ONE Studio 5, Standard Edition is an integrated development environment (IDE) for Java technology. Based on the modular, open source NetBeans Tools Platform, this IDE is ideal for building and deploying Web services across multiple hardware and software platforms, including Windows, Windows NT, Linux, and the Solaris Operating Environment.

www.programmersparadise.com/sunone

Paradise # S69 0U67
$670.99

800-445-7899 • programmersparadise.com

Prices subject to change. Not responsible for typographical errors.

Note that the enumeration loop retrieves and inserts one item at a time. This is necessary because items from an arbitrary enumerator could be of any size. While it is certainly possible to cater for retrieving multiple items at a time, it would complicate the implementation of the handlers to a degree that I determined would not be warranted. (The efficiencies gained from reduced round-trips to COM enumerators are associated with marshalling across apartment/process/machine, and most enumerators from which one would wish to retrieve will likely be in the calling apartment. If not, marshal-by-value is also commonly used.) Also, all of the standard window handlers cause each string item to be added after the previous one, which covers most cases, but may not be what you want.

That's the main function. What remains is to see how the handlers are implemented. Listing 4 (available online) shows a selection of the enumerator handlers, and Listing 5 shows some of the control handlers. I won't go into too much detail as I've already discussed their required semantics. There are, however, a few points of interest that I'll briefly discuss.

First, the standard controls have to do conversions of the text to ANSI if the windows are not Unicode (determined by the calls to `IsWindowUnicode()`); there is no need to do this for the list view because, like all the Common Controls, it can work with both ANSI and Unicode messages. Second, the edit control handler works by moving the selection to the end of the contents, pasting the value text, repeating the selection, and pasting CR-LF. This is done in two parts to remove the need to concatenate the value and CR-LF, but it is conceivably open to race conditions (you'd have to write an atomic version if this was a possibility in your application). The rest of the implementations are fairly obvious, except the `IEnumUnknown` handler, which is discussed in the sidebar titled "DISPID_VALUE" (available online).

C-API in STL

`ListExchange_PutItems()` is a good solution for handling a variety of cases, whether for easy prototyping where one may wish to change list control types, or for production code that needs to handle multiple enumerator interfaces. However, using it constrains you to the enumerator and control types understood by the provided handlers, which may not cover your particular needs, or requires you to define your own, which is a fair amount of effort.

It is also a bit difficult to modify the items as they go into the list controls. This would require using special enumerator and control handler functions to change the value. The API does cater for that—by defining the value type to be `void *` and not `LPCOLESTR`, and by providing a user-supplied parameter into `ListExchange_PutItems_Base()` that is passed through to the control handler's `pfnPutItem`—but it's not a trivial effort to use it in this way.

The C-API was written a long time before STL became popular, and there are now

Listing 5 Implementation of control handlers

```
/* ////////////////////////////////////////////////////////////
 * ...
 *
 * Extract from MOCtlFns.c
 *
 * Copyright (C) 1998-2003, Synesis Software Pty Ltd.
 * (Licensed under the Synesis Software Standard Public License:
 *  http://www.synesis.com.au/licenses/ssspl.html)
 *
 * ...
 * //////////////////////////////////////////////////////////// */

/* Listbox */

HRESULT ListBox_PutItem(HWND hwnd, LPVOID value, UInt32 flags, UInt32 param)
{
  HRESULT hr;

  if(IsWindowUnicode(hwnd))
  {
    hr = (ListBox_AddString(hwnd, (LPCOLESTR)value) < 0)
            ? E_FAIL : S_OK;
  }
  else
  {
    . . . // Do conversion to ANSI and add
  }

  return hr;
}

void ListBox_Reset(HWND hwnd)
{
  ListBox_ResetContent(hwnd);
}

/* Combobox */

HRESULT ComboBox_PutItem(HWND hwnd, LPVOID value, UInt32 flags, UInt32 param)
{
  HRESULT hr;

  if(IsWindowUnicode(hwnd))
  {
    hr = (ComboBox_AddString(hwnd, (LPCOLESTR)value) < 0)
            ? E_FAIL : S_OK;
  }
  else
  {
    . . . // Do conversion to ANSI and add
  }

  return hr;
}

void ComboBox_Reset(HWND hwnd)
{
  ComboBox_ResetContent(hwnd);
}

/* Listview */

HRESULT ListView_PutItem(HWND hwnd, LPVOID value, UInt32 flags, UInt32 param)
{
  LV_ITEMW  item;

  item.mask     = LVIF_TEXT;
  item.iItem    = ListView_GetItemCount(hwnd); /* add to end */
  item.iSubItem = 0;
  item.pszText  = (LPWSTR)value;

  return (int)SendMessage(hwnd, LVM_INSERTITEMW, 0, (LPARAM)&item) < 0
            ? E_FAIL : S_OK;
}

void ListView_Reset(HWND hwnd)
{
  ListView_DeleteAllItems(hwnd);
}

/* Edit */

HRESULT Edit_PutItem(HWND hwnd, LPVOID value, UInt32 flags, UInt32 param)
{
  HRESULT hr;

  SendMessage(hwnd, EM_SETSEL, (WPARAM)-1, (LPARAM)-1);

  if(IsWindowUnicode(hwnd))
  {
    hr = SendMessage(hwnd, EM_REPLACESEL, 0, (LPARAM)(LPCOLESTR)value)
              ? S_OK : E_FAIL;
    if(SUCCEEDED(hr))
    {
      SendMessage(hwnd, EM_SETSEL, (WPARAM)-1, (LPARAM)-1);
      hr = SendMessage(hwnd, EM_REPLACESEL, 0, (LPARAM)L"\r\n")
                ? S_OK : E_FAIL;
    }
  }
  else
  {
    . . . // Do conversion to ANSI and add
  }


  return hr;
}

void Edit_Reset(HWND hwnd)
{
  SetWindowText(hwnd, "");
}
```

alternatives if you prefer to be in the STL world entirely. Perhaps you only need to populate one kind of control and from one kind of enumeration interface, or you want to modify the contents of the retrieved items with adaptor functionals (function objects). Two STLSoft subprojects, COMSTL and WinSTL, provide components that, when used in concert, provide a very simple solution. If you have an `IEnumString` enumerator and you want to populate a list-box, the following code will do that:

```
LPENUMSTRING  penStr      = . . .
// An IEnumString instance
HWND          hwndListBox = . . .
// and a window to put it in

// Declare an enumerator sequence for
//IEnumString
comstl::enum_simple_sequence< IEnumString
    , LPOLESTR
    , comstl::LPOLESTR_policy
    , LPCOLESTR
    , comstl::input_cloning_policy<IEnumString>
    , 5 // Next() five at a time
    > strings(penStr, true);
// borrows the reference

std::for_each(strings.begin(), strings.end(),

winstl::listbox_back_inserter(hwndListBox));
```

Alternatively, if you have `IEnumVARIANT` and you want it to go into a combo box in reverse order:

```
comstl::enum_simple_sequence< IEnumVARIANT
    , VARIANT
    , comstl::VARIANT_policy
    , VARIANT const
    ,
comstl::input_cloning_policy<IEnumVARIANT>
    , 10 // Next() 10 at a time
    > strings(penVar, false);
// eats the reference

std::for_each(strings.begin(), strings.end(),
              winstl::combobox_front_insert
                    er(hwndComboBox));
```

(It looks like a lot of code, but almost all of it is the parameterization of the enumerator sequence template. In real code, these are usually typedef'd.) As well as not needing to compile and link in the C-API, this has some other advantages. The sixth parameter to the template is a number, which specifies the number of items to acquire in each call to the enumerator's `Next()` method. Where this is important (e.g., when marshalling between apartments or between hosts), it allows you to increase efficiency by reducing round-trips by parameterizing the template appropriately.

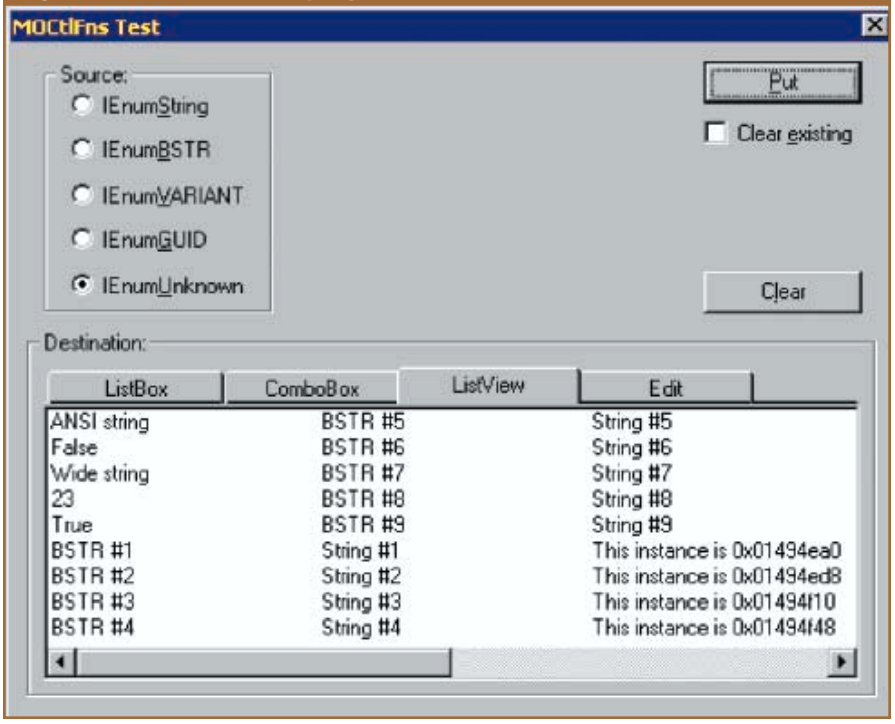
You could also provide modification of the items before they go into the list control by applying an adaptor function (e.g., to set to uppercase) to the control inserter functional, as in:

```
template <typename F>
struct upper_adaptor;
template <typename F>
upper_adaptor<F> make_upper_adaptor(F );

for_each( strings.begin(), strings.end(),

make_upper_adaptor(combobox_front_inserter(hwnd
                    ComboBox)));
```

Figure 1 Enumerator test program



Test client

Included in the online archive is the test program, shown in Figure 1. The dialog allows you

Listing 6 Use of ListExchange_PutItems() in test program

```
static HWND MOCtlFns_Test_GetListWindow(HWND hwnd)
{
  HWND  hwndTab   = ::GetDlgItem(hwnd, IDC_TAB);
  int   iCurSel   = TabCtrl_GetCurSel(hwndTab);
  int   idList    = Tab_GetItemData(hwndTab, iCurSel);
  HWND  hwndList  = ::GetDlgItem(hwnd, idList);

  _ASSERTE(idList != 0);
  _ASSERTE(hwndList != 0);

  return hwndList;
}

static void MOCtlFns_Test_OnClear(HWND hwnd)
{
  HWND  hwndList  = MOCtlFns_Test_GetListWindow(hwnd);

  ListExchange_PutItems(hwndList, NULL, SYLXF_RESETCONTENT);
}

static void MOCtlFns_Test_OnPut(HWND hwnd)
{
  LPUNKNOWN penItems;
  HRESULT   hr        = MOCtlFns_Test_GetEnumerator(hwnd,
                                IID_IUnknown,
                                reinterpret_cast<void**>(&penItems));
  HWND      hwndList  = MOCtlFns_Test_GetListWindow(hwnd);

  if(SUCCEEDED(hr))
  {
    UInt32 flags = IsDlgButtonChecked(hwnd, IDC_CLEAR_EXISTING)
                                          ? SYLXF_RESETCONTENT
                                          : 0;

    /* Simply pass the window, enumerator (as IUnknown) and
     * flags to the function.
     */
    ListExchange_PutItems(hwndList, penItems, flags);

    penItems->Release();
  }
}
```

to select one of five enumerator types (each of which are filled with sample data when they're created) and to select one of four list windows. Listing 6 shows some functions from the test program. `MOCtlFns_Test_GetListWindow()` is a helper function that retrieves the ID of the list control that is associated with a tab control item, and returns the corresponding window handle. It is used by `MOCtlFns_Test_OnClear()`, which demonstrates the use of `ListExchange_PutItems()` to clear out any list control, and by `MOCtlFns_Test_OnPut()`, which demonstrates the list population. `MOCtlFns_Test_OnPut()` obtains the appropriate COM enumerator from another function in the test program, `MOCtlFns_Test_GetEnumerator()`, which synthesizes an enumerator of the appropriate type (as determined by the "Source" radio group). In real code, you'd have received the enumerator from a source and would likely obtain the control from `GetDlgItem()`, so your code would be very succinct.

Listing 7 shows a solution based around the COMSTL `enum_simple_sequence<>` sequence class and the WinSTL control (standard and common) inserter functionals. Clearly, there is a lot of code to write, but this is just the demonstration program. In real-world scenarios, you'd use the C-API if you need this level of complexity, and the COMSTL/WinSTL components if you didn't.

Iterator concepts and IEnumXXXX::Clone()

A full description of the iterator concept is given at http://sgi.com/tech/stl/Iterators.html, and an in-depth treatment of the issues involved in writing iterators and sequences is given in my March 2003 *WDN* article (see References), so I'll try and be very brief here. An input iterator is something from which one may acquire its value only once. When copied, the ownership of the "range" is passed to the copy, making subsequent retrieval from the original invalid. A forward iterator is something from which one may take a copy and retrieve meaningful results from both copies.

COM programmers will recognize that the COM enumerator model describes a forward iterator semantic, by the `Clone()` method provided by all `IEnumXxxx` enumerator interfaces, which is defined to return a copy of the source enumerator that is at the same point in its enumeration as the source. However, it is legal for the `Clone()` method to fail, so were we to implement `enum_simple_sequence`'s iterators as forward iterators, that could lead to trouble when it did so. Conversely, most enumerators succeed a call to `Clone()`, so were we to implement the iterators as input iterators, we would be restricting functionality in a large number of cases where that would not be necessary. Hence, the COMSTL `enum_simple_sequence<>` template takes a cloning policy so that users can select the desired iterator concept support based upon their needs.

Listing 7
Permutations of enum_simple_sequence<> and functionals in test program

```
using comstl::enum_simple_sequence;
using comstl::input_cloning_policy;
using comstl::LPOLESTR_policy;
using comstl::BSTR_policy;
using winstl::listbox_back_inserter;
using winstl::combobox_back_inserter;
using winstl::listview_back_inserter;

static void MOCtlFns_Test_Put(HWND hwnd)
{
  LPUNKNOWN punk;
  HRESULT   hr        = MOCtlFns_Test_GetEnumerator(hwnd,
                          IID_IUnknown,
                          reinterpret_cast<void**>(&punk));
  HWND      hwndList  = MOCtlFns_Test_GetListWindow(hwnd);

  if(SUCCEEDED(hr))
  {
    HWND  hwndTab   = ::GetDlgItem(hwnd, IDC_TAB);
    int   iCurSel   = TabCtrl_GetCurSel(hwndTab);
    int   idList    = Tab_GetItemData(hwndTab, iCurSel);

    if(::IsDlgButtonChecked(hwnd, IDC_STRING))
    {
      LPENUMSTRING  penum;

      hr = punk->QueryInterface(IID_IEnumString,
                        reinterpret_cast<void**>(&penum));

      if(SUCCEEDED(hr))
      {
typedef enum_simple_sequence
                    < IEnumString
                    , LPOLESTR
                    , LPOLESTR_policy
                    , LPCOLESTR
                    , input_cloning_policy<IEnumString>
                    , 5
                    >   enum_sequence_t;

        // Declare the sequence (and eat the reference)
        enum_sequence_t   values(penum, false);

        switch(idList)
        {
          case  IDC_LISTBOX:
            std::for_each(values.begin(), values.end(),
                          listbox_back_inserter(hwndList));
            ListBox_ResetContent(hwndList);
            break;
          case  IDC_COMBOBOX:
            std::for_each(values.begin(), values.end(),
                          combobox_back_inserter(hwndList));
            ComboBox_ResetContent(hwndList);
            break;
          case  IDC_LISTVIEW:
            std::for_each(values.begin(), values.end(),
                          listview_back_inserter(hwndList));
            ListView_DeleteAllItems(hwndList);
            break;
        }
      }
    }
    else if(::IsDlgButtonChecked(hwnd, IDC_BSTR))
    {
      LPENUMBSTR penum;

      hr = punk->QueryInterface(IID_IEnumBSTR,
                          reinterpret_cast<void**>(&penum));

      if(SUCCEEDED(hr))
      {
typedef enum_simple_sequence
                    < IEnumBSTR
                    , BSTR
                    , BSTR_policy
                    , BSTR const
                    , input_cloning_policy<IEnumBSTR>
                    , 5
                    >   enum_sequence_t;

        // Declare the sequence (and eat the reference)
        enum_sequence_t   values(penum, false);

        switch(idList)
        {
          case  IDC_LISTBOX:
            std::for_each(values.begin(), values.end(),
                          listbox_back_inserter(hwndList));
            ListBox_ResetContent(hwndList);
            break;
          case  IDC_COMBOBOX:
            std::for_each(values.begin(), values.end(),
                          combobox_back_inserter(hwndList));
            ComboBox_ResetContent(hwndList);
            break;
          case  IDC_LISTVIEW:
            std::for_each(values.begin(), values.end(),
                          listview_back_inserter(hwndList));
            ListView_DeleteAllItems(hwndList);
            break;
        }
      }
    }
    else
    {
      . . . // etc. etc.
    }

    punk->Release();
  }
}
```

COMSTL provides several cloning policies, including:

- `input_cloning_policy<>` assumes Input Iterator semantics, and transfers ownership of the enumerator from the source iterator to the destination iterator.
- `forward_cloning_policy<>` assumes Forward Iterator semantics, and calls `Clone()` on the source iterator's enumerator to obtain one for the destination iterator. If this fails, the destination iterator is set to null, and the destination range [begin, end) is empty.

In our code, we have used the `input_cloning_policy<>` since in all cases, we only need one byte at the cherry of the enumerator ranges.

Summary

This article has examined the common but programmatically complex situation of transferring information from COM enumerators to windows controls. In a sense, we've looked at the two ends of the spectrum of modernity in our two methods. The old way, a C-API written in C, sacrifices some run-time efficiency for succinct client code and flexibility in the things that it can handle at run time. It has the slight disadvantage that it must work with one enumerated item at a time. It has a nice by-product that the list control can be cleared by a single function call without needing to worry about the window class. This is a piece of flexibility that can only be achieved at run time, since all windows are identified at compile time by their `HWND` only, and are distinguished by their window class that is obtained by a function call (`GetClassName()`).

The new way, using STL sequences and function objects, is more appealing to those (of us) who revel in intellectual elegance. It's also likely to be more efficient, since there are no run-time checks carried out on enumerator and list-control types, and it can retrieve multiple items in each call to `IEnumXxxx::Next()`. As is the case with generic (template) programming, you do have polymorphism, but it is compile time (i.e., in the type resolution and template parameterization). If this suits your needs, this is great. If you need to deal with multiple potential interfaces at run time, then you should stick with the C-API.

As with all things in software engineering, there is no absolute best approach. Each of these two approaches reflects an optimal solution to a particular set of circumstances. In one set of circumstances, you would use one; in a different set, the other. It's just a case of filling your bag of tricks with enough tricks.

Finally, you may be wondering whether I was going to discuss data exchange in the other direction? Well I'm sure it won't surprise you to learn that I've got both C-API and STL techniques for doing this, but that's another (longer) article.

References

Generic Programming and the STL, Matt Austern, Addison-Wesley, 1998.

STLSoft is an open-source organization applying STL techniques to a variety of technologies and operating systems. It is located at http://stlsoft.org/. COMSTL and WinSTL are subprojects, and are located at http://comstl.org/ and http://winstl.org/, respectively.

"Adapting Win32 Enumeration APIs to STL Iterator Concepts," Matthew Wilson, *Windows Developer Magazine*, March 2003.

"Inserter Function Objects for Windows Controls," Matthew Wilson, *Windows Developer Network*, November 2003. w::d

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

Connect Office Apps to .NET with "Lite" Web Services

By skipping the web services toolkit and client setups, you won't leave legacy Office users behind

MANY OF US WOULD like to be able to take advantage of web services and access .NET data from Microsoft Office applications such as Excel using VBA. Imagine your users opening an Excel spreadsheet and connecting it to your ASP.NET application in order to view data in a rich environment with equations and charts, and even being able to upload data, as in Figure 1. You could create web services using the Office XP Web Services Toolkit 2.0, but then you will need a deployment package that includes Microsoft XML (MSXML) and Microsoft SOAP (MSSOAP). At times we can't require client installations because, for one reason or another, the workstations and laptops that will be running office applications are out of our control. Although Office 2003 will provide even greater support for interacting with .NET and web services, it may be a while before many of our clients are upgraded. Luckily, it is possible to use existing technology to communicate between .NET and versions of Office products from the 2000, XP, and 2003 family. It might even work with Office 97, although I haven't tried it.

Before I get into the details, I should mention Visual Studio Tools for Office (VSTO). If you perform a complete install of Office 2003 on a machine where the .NET Framework 1.1 is installed, then the Office 2003 Interop assemblies will also be installed into the Global Assembly Cache. You may then download and install VSTO. The tools will allow you to use Visual Studio .NET 2003 along with C# or VB.NET to write code behind Word 2003 and Excel 2003 documents. You create, run, and debug these projects from VS.NET as you would any other .NET project, and you have the Excel and Word object models available as you would in VBA. Your code behind compiles into a DLL that can either reside locally with the Excel or Word document, or be placed in a location such as a network share or web site where the document will look for it (as well as updates to it). VSTO gives you all of the benefits of using .NET—the security of managed code, ADO.NET, XML, no-touch deployment, a feature-rich IDE, etc.—and allows you to use it to program Word and Excel documents, which also has new built-in functionality for working with XML data. Read more at http://msdn.microsoft.com/vstudio/office/officetools.aspx.

With VSTO, you can easily call web services just as you would from any other C# or VB.NET application, and you can write your entire Word or Excel (2003) application without using VBA at all. You don't need to do the extra plumbing in .NET and VBA that I will show you how to do in this article—so if your user base will soon have Office 2003, then you should be looking at VSTO. But if you still need to support several versions of Office, this technique is for you.

In this article, I will show you step-by-step how to create a "lite" web service by using ASP.NET, and how to call it from VBA. I will take advantage of a class in MSXML called "XMLHTTPRequest." Aptly named, this class handles all of the plumbing to make an HTTP request and receive an XML response. Fortunately, MSXML began shipping with Internet Explorer version 4.01 so, unlike MSSOAP, we can assume our clients already have MSXML installed. To package the data up into XML format, I will create a disconnected ADO Recordset on the fly. The Recordset's `Save` method has an option to persist the Recordset in its own XML format. ADO can be used from both VBA and .NET (via COM Interop), but since many of us are already working with ADO.NET, I will show you how to convert DataSets to and from Recordsets. This article assumes that you have access to Visual Studio .NET 2002 or 2003, a SQL Server with the pubs database, and Excel. The listings are written in C# and VBA, and are available for download.

LUTHER MILLER *is a software architect at Softagon Corp. in San Francisco. He is an MCSD for .NET and, when he is not hiking or watching Futurama, creates solutions for the financial industry using .NET, SQL Server, and Office. Questions/comments are welcomed at luther@anandus.com.*

Convert a DataTable to a Recordset

Let's assume that you already have some code that uses ADO.NET and returns important data in the format of a DataSet. It could simply be querying data from a database or an XML file, or it could be the result of some complex business logic. But you can't access ADO.NET objects in VBA, so we need to convert the data into a format that we can easily work with in VBA—Recordsets. A DataSet may actually contain several DataTables, so we will convert a DataSet to a collection of Recordsets. To get started, launch Visual Studio .NET and create a new C# ASP.NET web project called "LiteWebServicesCS." Remove WebForm1.aspx—you won't need it. Then add a new code file to the project called "ADOUtility.cs." You will also need to add a reference to adodb (under the .NET tab) to the project's references. This is a .NET COM-Interop assembly that lets us work with ADO. It is not installed with the .NET Framework, but it comes with Visual Studio .NET, so you'll need to deploy it when you deploy your ASP.NET application.

As you can see in Listing 1, `ConvertDataTableToRecordset` is a static method that takes a DataTable as a parameter and returns a Recordset. This is done by creating a new instance of a Recordset. You may only be familiar with receiving Recordsets as output from ADO. In this case, I am making use of an ADO

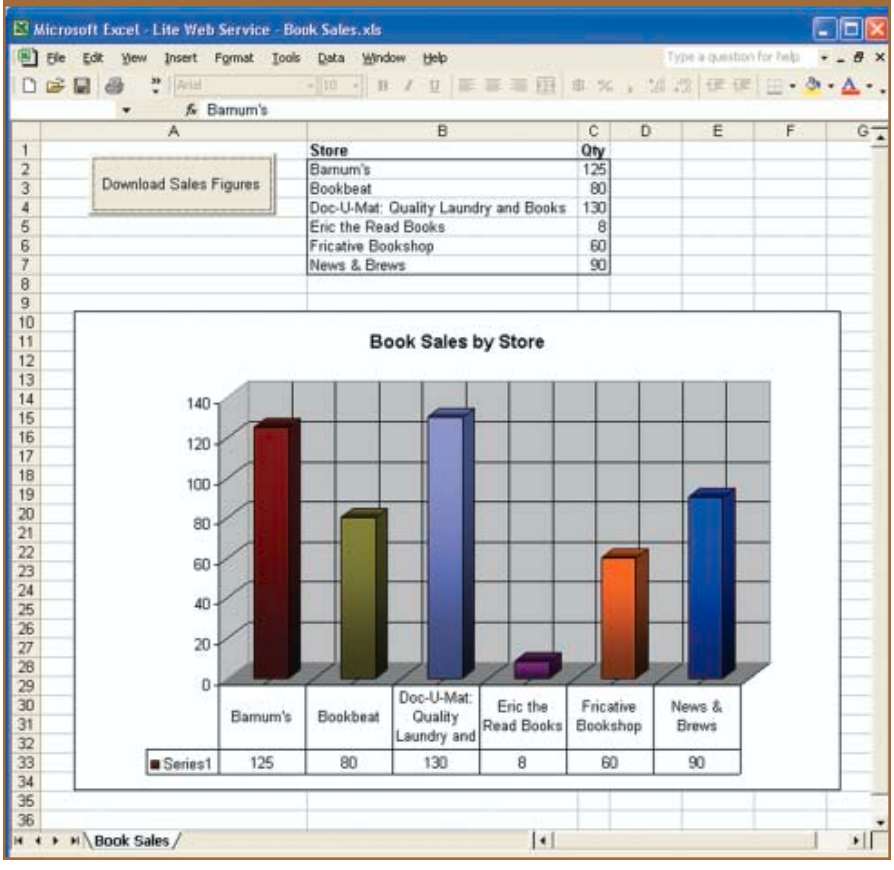
feature that allows the creation of disconnected Recordsets from scratch. The next step is to loop through the columns in the DataTable and create corresponding fields in the Recordset. This is done by looking up the .NET data type and retrieving an equivalent ADO data type and maximum size. Finally, I loop through the Rows collection in the DataTable, retrieve the values for each row, and add them to the Recordset. You might have noticed that I do some special processing for decimal data types when I create the Recordset's fields—you will need this code if you use the decimal data type, otherwise you might lose precision. I also do special processing for globally unique identifiers (GUIDs); otherwise ADO will not recognize them as such.

Listing 2 contains the lookup method to determine the proper ADO data type and size for a given .NET type. The ADO.NET `Column` object's DataType property returns a `System.Type` object, which is passed into `GetADOType`. I chose to create a two-dimensional object array to hold the lookup data. I use the C# `typeof` keyword to get an instance of the correct object; `typeof(System.Boolean)` creates a `System.Type` object corresponding to `System.Boolean`, for example. I loop through the array looking for the specified type, and then return the equivalent ADO data type and maximum length as output parameters. I went through the available ADO data types and included each one that could easily be converted, and this will cover all but the rarest situations. There is an exhaustive list in the ADO API Reference at http://msdn.microsoft.com/library/en-us/ado270/htm/mdaenumdm.asp.

Persist Recordsets as XML

The Recordset's `Save` method can be used to persist the Recordset in its very own XML format. Likewise, the `Open` method of the Recordset will recreate the Recordset from that persisted XML. The XML itself is not very useful to us, but just by being XML, it can easily be manipulated. At this point, my goal is to create a method to return the Recordset's XML as a string, but the `Save` method saves it to a

Figure 1 Accessing Data in Excel with a "Lite" Web Service



Listing 1 ConvertDataTableToRecordset

```
// Converts the specified DataTable into an ADO Recordset.
static public ADODB.Recordset ConvertDataTableToRecordset(DataTable table)
{
  // Create a new (disconnected) Recordset
  ADODB.Recordset recordset = new ADODB.Recordset();

  // Create an array for the names of the colums to use
  // later with the AddNew method of the Recordset.
  object [] fieldList = new object[table.Columns.Count];

  // Loop through all of the columns in the DataTable
  // and define each as a new field in the Recordset;
  // also add the name to the fieldList array.
  for (int i=0; i < table.Columns.Count; i++)
  {
    // get the field name, or create one if necessary
    string fieldName = table.Columns[i].ColumnName;
    if ( fieldName == null || fieldName.Length == 0 )
      fieldName = string.Format("Column{0}", i);
    fieldList[i] = fieldName;

    // Lookup the field's equivalent ADO type and maximum size
    ADODB.DataTypeEnum adoType;
    int adoSize;
    GetADOType(table.Columns[i].DataType, out adoType, out adoSize);

    // add the field to the ADO Recordset
    recordset.Fields.Append(fieldName, adoType, adoSize,
      ADODB.FieldAttributeEnum.adFldIsNullable, null);

    // These generic values appear to work well with any size
    // decimal/numeric; setting them for other types does not
    // cause harm, but they are necessary for decimals.
    recordset.Fields[i].Precision = 0;
    recordset.Fields[i].NumericScale = 25; //the maximum
  } //for

  // Now that the fields are defined, open the Recordset so we may
  // add the data. Notice that because C# does not work like VB with
  // optional parameters, we use the special Missing object which
  // is defined in System.Reflection.
  recordset.Open(Missing.Value, Missing.Value,
    ADODB.CursorTypeEnum.adOpenUnspecified,
    ADODB.LockTypeEnum.adLockUnspecified, -1);

  for (int i=0; i < table.Rows.Count; i++ )
  {
    // get the current record's values into an array
    object [] values = table.Rows[i].ItemArray;

    // ADO does not recognize GUID, or unique identifiers,
    // unless they are wrapped in curly braces, we check for
    // any occurences of them here and convert them to a
    // string in the format "{00000000-0000-0000-0000-000000000000}"
    for (int j=0; j < values.Length; j++ )
      if ( values[j] is System.Guid )
        values[j] = '{' + values[i].ToString() + '}';

    // add the current record to the ADO Recordset
    recordset.AddNew(fieldList, values);
  }

  // Done - Recordset complete!
  return recordset;
}
```

Listing 2 GetADOType

```
// Based on the .NET data type, return the appropriate ADO data type and
// size to be used when defining the Recordset.
static public void GetADOType(Type dotNetType,
  out ADODB.DataTypeEnum adoType, out int adoSize)
{
  adoType = DataTypeEnum.adEmpty;
  adoSize = -1;

  object [][] lookup = new object [][]
  {
    new object [] {typeof(System.Boolean), DataTypeEnum.adBoolean, -1},
    new object [] {typeof(System.Byte), DataTypeEnum.adUnsignedTinyInt, -1},
    new object [] {typeof(System.Byte[]), DataTypeEnum.adBinary, 32767},
    new object [] {typeof(System.DateTime), DataTypeEnum.adDate, -1},
    new object [] {typeof(System.Decimal), DataTypeEnum.adDecimal, -1},
    new object [] {typeof(System.Double), DataTypeEnum.adDouble, -1},
    new object [] {typeof(System.Guid), DataTypeEnum.adGUID, -1},
    new object [] {typeof(System.Int16), DataTypeEnum.adSmallInt, -1},
    new object [] {typeof(System.Int32), DataTypeEnum.adInteger, -1},
    new object [] {typeof(System.Int64), DataTypeEnum.adBigInt, -1},
    new object [] {typeof(System.SByte), DataTypeEnum.adTinyInt, -1},
    new object [] {typeof(System.Single), DataTypeEnum.adSingle, -1},
    new object [] {typeof(System.String), DataTypeEnum.adVarWChar, 32767},
    new object [] {typeof(System.UInt16),
                   DataTypeEnum.adUnsignedSmallInt, -1},
    new object [] {typeof(System.UInt32), DataTypeEnum.adUnsignedInt, -1},
    new object [] {typeof(System.UInt64), DataTypeEnum.adUnsignedBigInt, -1},
  };

  // find the .NET type in the lookup array and retrieve the
  // corresponding ADO Type and Size.
  foreach (object [] type in lookup)
    if ( dotNetType == (Type) type[0] )
    {
      adoType = (DataTypeEnum) type[1];
      adoSize = (int) type[2];
      break;
    }
}
```

file, or, starting with ADO 2.5, to an ADO `Stream` object. A few short lines of code perform this task easily in the `ConvertRecordsetToXmlText` method (see Listing 3).

Recalling that a DataSet contains a collection of DataTables, we need a way to turn several Recordsets into one XML document that can be returned to the client. The `System.Xml` namespace allows us to manipulate XML in a variety of ways, and now we can reap the benefits of having our Recordset in XML format. Since XML is hierarchical, I create a new XML document with an element called "results" to represent multiple Recordsets. I then loop through the tables in the DataSet and add each one's Recordset XML as a child of the results element. Since the Recordset's XML is an XML document in its own right, I use the `ImportNode` method to bring a Recordset as a child element into the new document. The `ConvertDataSetToXmlDocument` method appears in Listing 4.

Creating a Lite Web Service with IHttpHandler

Now that we have a way to represent our data in XML format, we need a way to make it available to our client applications. We can build our service as an Http Handler and let ASP.NET take care of the rest. I call this a lite web service because it only has a few of the characteristics of a real web service: It uses XML and HTTP, but does not use SOAP or conform to any other standards. An ASPX page, or web form, is really just a specialized Http Handler. To build an Http Handler, start out by opening the Class View window, right-clicking on the project, and selecting Add Class. In the wizard, set the class name to `GetAuthors` and then select the Inheritance tab. Set the current namespace to `System.Web` and then add `IHttpHandler`, followed by Finish. In the class view, find `GetAuthors` in the class hierarchy, open Bases and Interfaces, right-click IHttpHandler, and select Add / Implement Interface. This will automatically create the properties and methods of the interface, so all you will need to do is fill in the code. Of course, you could have done all of this from scratch, without using the tools in the Class View, but it is useful to know about the Implement Interface command.

Listing 3 ConvertRecordsetToXmlText

```
// This function accepts a Recordset and converts to an XML string.
static public string ConvertRecordsetToXmlText(ADODB.Recordset recordset)
{
  ADODB.Stream stream = new ADODB.Stream();

  stream.Open(Missing.Value, ADODB.ConnectModeEnum.adModeUnknown,
    ADODB.StreamOpenOptionsEnum.adOpenStreamUnspecified, null, null);
  recordset.Save(stream, ADODB.PersistFormatEnum.adPersistXML);
  stream.Position = 0;

  return stream.ReadText(-1);
}
```

The `ProcessRequest` method is all that we really need to deal with. I deleted the constructor that the wizard created, and I didn't touch the code that returns True for `IsReusable`, because I will implement the class such that a given instance could be invoked multiple times. The only parameter to `ProcessRequest` is an `HttpContext` object. The `HttpContext` object exposes everything we will need including the `Request` and `Response` objects. This means that we can look at query parameters, get and set cookies, and so on, just as with a Web Form. In fact, all we really need to do is write our XML to the response stream.

I added a method to the `GetAuthors` class called `GetAuthorsData`, which returns a DataSet containing the results of "select * from authors" from the database. The data access code looks for the connection string to be defined in the Web.Config file, so add it now just below the <configuration> tag. It should look something like Listing 5. The `ProcessRequest` method simply retrieves the data, calls `ConvertDataSetToXmlDocument`, and then writes the documents XML as the response. I also set the response's Content-Type header to "text/xml," and I set the `CacheControl` property to "no-cache" since I don't want clients to cache the results. The complete `GetAuthors` class is in Listing 6.

Listing 4 ConvertDataSetToXmlDocument

```
// Creates an XML document from a DataSet containing multiple Recordsets.
static public XmlDocument ConvertDataSetToXmlDocument(DataSet dataSet)
{
  // create a new XML document to hold more than one Resultset
  XmlDocument xmlDoc = new XmlDocument();
  // create a "results" element to contain the Recordsets
  XmlElement xmlResults = xmlDoc.CreateElement("results");
  xmlDoc.AppendChild(xmlResults);

  // loop through all the tables in the DataSet
  foreach (DataTable dataTable in dataSet.Tables)
  {
    // Convert the Recordset's XML into an XML Document
    XmlDocument currentRecordsetDocument = new XmlDocument();
    currentRecordsetDocument.LoadXml(
      ConvertRecordsetToXmlText( ConvertDataTableToRecordset(dataTable) )
      );

    // Append the Recordset's XML document as a child node
    xmlResults.AppendChild( xmlDoc.ImportNode(
      currentRecordsetDocument.DocumentElement, true) );
  }

  // Done! Return the new XML document representing the DataSet
  return xmlDoc;
}
```

Listing 5 Connection string definition

```
<appSettings>
  <add key="ConnStr" value=
  "integrated security=sspi;initial catalog=pubs;data source=localhost;" />
</appSettings>
```

There is one last step before our service can be accessed. We need to map a specific URL to our Http Handler. This is done in the Web.Config file by adding an entry to the `<httpHandlers>` section, which you may need to add under `<system.web>`. We need to tell ASP.NET what type of HTTP "verbs" we will handle (GET, POST, etc.), what "path" to look for in the URL, and the Http Handler "type" (class) that will handle the request. See Listing 7 for my mapping of `GetAuthors`. Note that I used "asmx" as the extension in the path. I also could have used "aspx," since both are mapped to ASP.NET in IIS when .NET is installed. Choosing another extension may require additional configuration in IIS in order to get things working. The verb of "*" means that it will handle all types of requests. See "Registering Http Handlers" in the .NET Framework documentation for more information.

At this point, you should be able to build your solution and do a quick test. Don't try to run it from Visual Studio because there is no startup page in your project. However, after you have built it, open up Internet Explorer and enter "http://localhost/LiteWebServicesCS/GetAuthors.asmx" into the address bar (change the "LiteWebServicesCS" part if your web project is configured differently). You should see the raw XML returned by the `GetAuthors` Http Handler. If you do want to run your service from Visual Studio—and you will want to in order to set breakpoints and debug it—then you can simply add a blank HTML page to the project and set it as the startup page.

Creating the VBA Client

Now that the lite web service is in place, the next step is to write the client code in VBA

Listing 6 GetAuthors class

```
// Returns a Recordset listing authors.
public class GetAuthors : System.Web.IHttpHandler
{
  protected DataSet GetAuthorsData()
  {
    SqlConnection conn = new SqlConnection(
      ConfigurationSettings.AppSettings["connStr"]
        );
    SqlDataAdapter adap = new SqlDataAdapter(
      "select * from authors",
      conn);
    DataSet dataSet = new DataSet();
    adap.Fill(dataSet);
    return dataSet;
  }

#region Implementation of IHttpHandler
  public void ProcessRequest(System.Web.HttpContext
          context)
  {
    DataSet dataSet = GetAuthorsData();
    XmlDocument xmlDoc =
      ADOUtility.ConvertDataSetToXmlDocument(
        dataSet );
    context.Response.ContentType = "text/xml";
    context.Response.CacheControl = "no-cache";
    context.Response.Write( xmlDoc.OuterXml );
  }

  public bool IsReusable { get { return true; } }
#endregion
}
```

Listing 7 GetAuthors mapping

```
<httpHandlers>
  <add verb="*" path="GetAuthors.asmx"
       type="LiteWebServicesCS.GetAuthors, LiteWebServicesCS" />
</httpHandlers>
```

Listing 8 InvokeLiteWebService

```
' This function is specifically designed to make a request to a
'   "Lite" web service implemented via an ASP.NET IHttpHandler.
' The Url parameter should include all query strings etc., e.g.:
'   http://server/MyLiteWSProject/MyService1.asmx?p1=val1&p2=val2
Public Function InvokeLiteWebService( _
    ByVal vsUrl As String, _
    Optional ByVal vvRequest As Variant, _
    Optional ByVal vsVerb As String = "GET" _
) As MSXML.DOMDocument

    Dim loXmlHttpRequest As New MSXML.XMLHTTPRequest
    Dim loXmlDocument As MSXML.DOMDocument

    'Connect to the ASP.NET URL and send the request
    loXmlHttpRequest.Open vsVerb, vsUrl, varAsync:=False
    If IsMissing(vvRequest) Then
        loXmlHttpRequest.send
    Else
        loXmlHttpRequest.setRequestHeader "Content-Type", _
            "application/x-www-form-urlencoded"
        loXmlHttpRequest.send vvRequest
    End If

    'Check for an <html> response - that would indicate an error
    'probably occured in ASP.NET and an HTML page was returned with
    'an error message.
    If 0 = StrComp("<html>", Mid$(loXmlHttpRequest.responseText, 1, 6), _
    vbTextCompare) Then
        'Note, this could be enhanced to save the HTML to a file and
        ' open in a browser, or to parse out the text to display nicely...
        Err.Raise vbObjectError + 1000, vsUrl, "An error occurred." _
            & vbCrLf & loXmlHttpRequest.responseText
    End If

    'Retrieve the response (an XML document object) and return it
    Set loXmlDocument = loXmlHttpRequest.responseXML
    Set InvokeLiteWebService = loXmlDocument
End Function
```

Listing 9 ConvertXmlDocumentToRecordsetsCollection

```
' Converts the XML document that was created on the server
' into a collection of Recordsets, representing a DataSet.
Public Function ConvertXmlDocumentToRecordsetsCollection( _
    ByVal voXmlDocument As MSXML.DOMDocument _
) As Collection
    Dim loRecordsetNodes As MSXML.IXMLDOMNodeList
    Dim loRecordsetNode As MSXML.IXMLDOMNode
    Dim loRecordsets As Collection

    Set loRecordsets = New Collection

    'loop through recordset nodes
    Set loRecordsetNodes = voXmlDocument.documentElement.childNodes
    For Each loRecordsetNode In loRecordsetNodes
        'loop through recordset nodes
        loRecordsets.Add ConvertXmlToRecordset(loRecordsetNode.XML)
    Next loRecordsetNode

    'return the collection of (disconnected) Recordsets
    Set ConvertXmlDocumentToRecordsetsCollection = loRecordsets
End Function
```

Listing 10 Converting Recordset's XML

```
' Converts the XML for a single Recordset back into a
' Recordset.
Public Function ConvertXmlToRecordset(ByVal vsXml As String) As ADODB.Recordset
    Dim loRecordset As New ADODB.Recordset
    Dim loStream As New ADODB.Stream

    loStream.Open
    loStream.WriteText vsXml
    loStream.Position = 0
    loRecordset.Open Source:=loStream, Options:=ADODB.CommandTypeEnum.adCmdFile

    Set ConvertXmlToRecordset = loRecordset
End Function
```

to access it. I will use Excel as a client, so open Excel and draw a new Command Button somewhere on the sheet. You might need to select the Control Toolbox by right-clicking the toolbar first. Double-click the Command Button to create the `CommandButton1_Click` event code in VBA, but don't add any code to it yet. Instead, add references to the VBA project for MSXML and ADO, since we will be using those objects. Go to Tools/References and check both "Microsoft ActiveX Data Objects 2.5 Library" (or later) and "Microsoft XML, version 2.0" (or later).

Next, create a function called `InvokeLiteWebService`, as in Listing 8. This method takes three parameters: the URL of the lite web service, an optional request, and an optional verb. The request is a variant containing data that should be sent to the service being invoked. So far, I have only discussed receiving data from the service—we can send data, too, as I will show you later in this article. The third parameter is the verb to be used for the HTTP request. For retrieving data, "GET" is fine. If we are going to send request data as well, then "POST" should be used.

To invoke the service, I create an instance of `MSXML.XMLHTTPRequest`. I use the `Open` method of this object, passing in the verb and the URL. I also specify that I want the call to be synchronous. In the `Open` method you may optionally specify a username and password, which could be used in conjunction with IIS and/or ASP.NET security. Next, I check to see if request data has been passed into `InvokeLiteWebService` that I need to send to the service. If not, I invoke the `Send` method of the `XMLHTTPRequest` without any parameters. If there is a request, then I first set the Content-Type header for the request, and then I call the `Send` method, passing the data to be posted.

The `Send` method invokes the service and receives the response. I'm expecting an XML response, but trial and error has taught me that if something goes wrong on the server side (such as an exception being thrown), then I might end up with an HTML response with an error message, so I check the response text to see if it starts with an `<html>` tag, and I raise an error if it does. If not, then everything went well and I retrieve the response as an XML document and return it. `XMLHTTPRequest` takes care of converting the XML text to an XML document for us. Note that either ASP.NET or IIS might return an HTML page if something goes wrong. You could write code to handle ASP.NET errors and package the error into the XML response with the recordsets, and look for it in the VBA code. You could also try and parse the text of the HTML response, using MSHTML for example, to show a text message to the user. I'll leave this as an exercise for the reader.

Converting the XML Back Into Recordsets

The `ConvertXmlDocumentToRecordsetsCollection` function in Listing 9 will reverse engineer the XML document. On the server, I added each Recordset's XML as a child node of the document I created. Now, I loop through the child nodes and call a function to convert the XML from each node into a single Recordset, and add each Recordset to a Collection. The code to convert a single Recordset's XML back to a Recordset, in Listing 10, writes the XML text to an

Listing 11 Copying the data into Excel

```
' Loads the results of GetAuthors into the first sheet
Private Sub CommandButton1_Click()
    Dim loRecordsets As Collection
    Dim lsUrl As String

    lsUrl = "http://localhost/LiteWebServicesCS/GetAuthors.asmx"

    'get the authors
    Set loRecordsets = ConvertXmlDocumentToRecordsetsCollection( _
        InvokeLiteWebService(lsUrl))

    'copy the authors Recordset into the Excel sheet
    Sheet1.Range("A1").CopyFromRecordset loRecordsets(1)
End Sub
```

ADO Stream object and then uses the Recordset's Open method to load the data from the stream—the opposite of how the XML was first created on the server.

Now we can write the code for the click event of the Command Button. I call the InvokeLiteWebService function and pass the results to ConvertXmlDocumentToRecordsetsCollection. The GetAuthors service only returns one result set, so the collection will only contain one Recordset. I use the CopyFromRecordset method of the Excel Range object to copy the data into Sheet1; see Listing 11. Go back into Sheet1 in Excel, and click the command button (make sure you are no longer in design mode by toggling the toolbar button, if necessary). The data from the pubs table is loaded into Sheet1.

Sending Data to the Server

At this point, we have a back-to-front solution for exposing the data from a .NET application as a lite web service that can be called from a VBA application. Now, I'll show you two ways to send data from a VBA application to a lite web service. The first is by simply adding a query string to the service's URL. The second is by sending an XML document representing multiple Recordsets—the same format that we are receiving data in.

I may want to create a lite web service to return titles for a given author, for example. This is a good case for using a query string to send data to the server. By simply putting the author's ID into the query string, the service can return the titles that correspond to that author. Since some characters in a query string need to be URL-encoded, I created a simple function in VBA to do URL encoding (see Listing 12). The basic idea is to append something like "?au_id=486-29-1786" to the URL. Test this out by adding another command button to the Excel sheet and adding the same code to the event as with the first command button, but this time append the query parameter to specify an author, as in Listing 13. Copy the GetAuthors handler in the ASP.NET application to create GetTitles. Don't forget to add another entry to the Web.Config file for GetTitles. Add some code to the GetTitles handler to retrieve the author ID and then use it in the SQL query, as in Listing 14.

Passing query parameters and accessing them from a lite web service is an easy way to send a couple of scalar parameters to the service.

Listing 12 UrlEncode

```
' Encode parameter name or value that will go into the query string of the
        URL.
Public Function UrlEncode(ByVal vsVal) As String
    Const lsOK_CHARS$ = _
        "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789-_@.*"
    Dim llIdx As Long
    Dim lsChar As String
    llIdx = 1
    Do
        lsChar = Mid$(vsVal, llIdx, 1)
        'is the current character is a space, change it to a "+"
        If lsChar = " " Then
            Mid$(vsVal, llIdx, 1) = "+"
        'is the current character OK, or does it need to be encoded?
        ElseIf InStr(1, lsOK_CHARS, lsChar) <= 0 Then
            vsVal = Mid$(vsVal, 1, llIdx - 1) & _
                "%" & Right$("0" + Hex(Asc(lsChar)), 2) & Mid$(vsVal, llIdx +
                    1)
            llIdx = llIdx + 2
        End If
        llIdx = llIdx + 1
    Loop While (llIdx <= Len(vsVal))

    UrlEncode = vsVal
End Function
```

Listing 13 Retrieving Titles by Author

```
' Retreive titles for a specific author. You'd make this a function
' that takes the Author ID as a parameter and returns the data.
Private Sub CommandButton2_Click()
    Dim loRecordsets As Collection
    Dim lsUrl As String

    lsUrl = "http://localhost/LiteWebServicesCS/GetTitles.asmx" _
        & "?" & UrlEncode("au_id") & "=" & UrlEncode("486-29-1786")

    'get the titles
    Set loRecordsets = ConvertXmlDocumentToRecordsetsCollection( _
        InvokeLiteWebService(lsUrl))

    'copy the titles Recordset into the Excel sheet
    Sheet1.Range("A26").CopyFromRecordset loRecordsets(1)
End Sub
```

Listing 14 GetTitles class

```
// Returns a Recordset listing authors.
public class GetTitles : System.Web.IHttpHandler
{
  protected DataSet GetTitlesData(string authorId)
  {
    SqlConnection conn = new SqlConnection(
      ConfigurationSettings.AppSettings["connStr"] );
    SqlCommand cmd = new SqlCommand(
      @"select t.*
        from pubs..titles t
        inner join pubs..titleauthor ta
          on t.title_id = ta.title_id
        where ta.au_id = @AuthorId",
      conn);
    cmd.Parameters.Add("@AuthorId", SqlDbType.VarChar);
    cmd.Parameters["@AuthorId"].Value = authorId;
    SqlDataAdapter adap = new SqlDataAdapter(cmd);
    DataSet dataSet = new DataSet();
    adap.Fill(dataSet);
    return dataSet;
  }

  #region Implementation of IHttpHandler
  public void ProcessRequest(System.Web.HttpContext context)
  {
    string authorId = context.Request.QueryString["au_id"];
    DataSet dataSet = GetTitlesData(authorId);
    XmlDocument xmlDoc =
      ADOUtility.ConvertDataSetToXmlDocument( dataSet );
    context.Response.ContentType = "text/xml";
    context.Response.CacheControl = "no-cache";
    context.Response.Write( xmlDoc.OuterXml );
  }

  public bool IsReusable { get { return true; } }
  #endregion
}
```

Listing 15 CreateOrderRecordsets

```
' Create a Recordset with title_ids and quantities from Sheet2.
Public Function CreateOrderRecordsets() As Collection
    Dim loRecordsets As Collection
    Dim loRecordset As New ADODB.Recordset
    Dim lvArray As Variant
    Dim llRecCtr As Long

    Set loRecordsets = New Collection

    loRecordset.Fields.Append "title_id", ADODB.DataTypeEnum.adVarChar, 50, _
        Attrib:=adFldIsNullable
    loRecordset.Fields.Append "order_qty", ADODB.DataTypeEnum.adInteger, _
        Attrib:=adFldIsNullable
    loRecordset.Open

    lvArray = Sheet2.Range("A1", "B2")

    For llRecCtr = 1 To UBound(lvArray, 1)
        loRecordset.AddNew
        loRecordset.Fields("title_id").Value = lvArray(llRecCtr, 1)
        loRecordset.Fields("order_qty").Value = lvArray(llRecCtr, 2)
    Next llRecCtr

    loRecordsets.Add loRecordset
    Set CreateOrderRecordsets = loRecordsets
End Function
```

However, you may need to create a service to which you can upload more data. The GetAuthors and GetTitles examples that we have discussed already both return data. This isn't a requirement, though. You could create a service to accept nothing and return data (as in GetAuthors), to accept data and return nothing, or to accept data and return data. In any case, I use the same model to upload data from the VBA code as I do to return it from the .NET service. This is done by putting the data to be uploaded into a collection of one or more Recordsets, converting that into an XML document (this time with VBA code), and posting it to the service. The service then converts the XML document to a DataSet, making it ready for your .NET code to work with. The process is really the same as the code that we have already covered, except that this time the code to create the XML document is in VBA, and the code to convert from it is in .NET.

Creating a Recordset in VBA code is similar to the .NET code. Listing 15 shows how you might create a Recordset from some data in an Excel spreadsheet. Notice that I specify the data type for each field in the Recordset. In the .NET method ConvertDataTableToRecordset, I also find the corresponding data type for each Recordset field. In both cases, the Recordset is strongly typed—that is, you will only be able to put values into the records that match the defined types. There is a special data type in ADO called "adVariant." We could simply define each field as an adVariant data type, avoiding the additional code to set each field's type, or convert to or from .NET data types. This would allow us to create a Recordset easily from any range of data in Excel, as in Listing 16. Inspection of the Recordset's XML shows that ADO apparently persists variant data as strings. Listing 17 (available online) shows the code to convert a collection of Recordsets into an XML document. Listing 18 (available online) shows how this data could then be sent to the server. Notice that I pass the XML document object itself as the request. The Send method of XMLHTTPRequest will post the XML document as the HTTP request.

Listing 16 CreateRecordsetFrom2DArray

```
' Convert a two dimensional array into a Recordset - you could pass Range.Value
' for the array, allowing a Recordset to easily be created from any range of
' data. The Recordset will not be stronly typed, however.
Private Function CreateRecordsetFrom2DArray(ByVal vvArray)
    Dim llColCtr As Long
    Dim llRecCtr As Long
    Dim loRecordset As New ADODB.Recordset
    Dim lvColumns() As Variant
    Dim lvData() As Variant
    ReDim lvColumns(LBound(vvArray, 2) To UBound(vvArray, 2))
    ReDim lvData(LBound(vvArray, 2) To UBound(vvArray, 2))

    For llColCtr = LBound(vvArray, 2) To UBound(vvArray, 2)
        loRecordset.Fields.Append "Column" & llColCtr, _
            ADODB.DataTypeEnum.adVariant, Attrib:=adFldIsNullable
        lvColumns(llColCtr) = llColCtr - LBound(vvArray, 2)
    Next llColCtr

    loRecordset.Open

    For llRecCtr = LBound(vvArray, 1) To UBound(vvArray, 1)
        For llColCtr = LBound(vvArray, 2) To UBound(vvArray, 2)
            lvData(llColCtr) = vvArray(llRecCtr, llColCtr)
        Next llColCtr
        loRecordset.AddNew lvColumns, lvData
    Next llRecCtr

    Set CreateRecordsetFrom2DArray = loRecordset
End Function
```

Listing 19 Receiving the order

```
// Receive a the Orders data
public class PlaceOrder : System.Web.IHttpHandler
{
  #region Implementation of IHttpHandler
  public void ProcessRequest(System.Web.HttpContext context)
  {
    XmlDocument xmlDocRecordsets = new XmlDocument();
    xmlDocRecordsets.Load(new XmlTextReader(context.Request.InputStream));

    DataSet orderDataSet =
ADOUtility.ConvertXmlDocumentToDataSet(xmlDocRecordsets);

    // process the orders DataSet here
    // ...
    // optionally, write a new response document to send back
  }

  public bool IsReusable { get { return true; } }
  #endregion
}
```

Listing 20 ConvertXmlTextToRecordset

```
// Converts ADO's XML back into a Reocrdset.
static public ADODB.Recordset ConvertXmlTextToRecordset(string xmlText)
{
  ADODB.Stream stream = new ADODB.StreamClass();
  stream.Open(Missing.Value, ADODB.ConnectModeEnum.adModeUnknown,
    ADODB.StreamOpenOptionsEnum.adOpenStreamUnspecified,
    "", "");
  stream.WriteText(xmlText,
    ADODB.StreamWriteEnum.adWriteChar);
  stream.Position = 0;

  ADODB.Recordset recordset = new ADODB.RecordsetClass();
  recordset.Open(stream, Missing.Value,
    ADODB.CursorTypeEnum.adOpenUnspecified,
    ADODB.LockTypeEnum.adLockUnspecified, -1);

  return recordset;
}
```

Listing 21 ConvertXmlDocumentToDataSet

```
// Converts our XML document with multiple Recordsets into a DataSet.
static public DataSet ConvertXmlDocumentToDataSet(
  XmlDocument xmlRecordsetsDoc)
{
  DataSet dataSet = new DataSet();

          foreach(XmlNode recordsetNode in xmlRecordsetsDoc.ChildNodes)
          {
    DataTable dataTable = new DataTable();
    ADODB.Recordset recordset =
      ConvertXmlTextToRecordset(recordsetNode.OuterXml);

    // Instead of converting the Recordset manually, we can take
    // advantage of a built in method of the OleDbDataAdapter
    // which will convert a Recordset into a DataTable.
    new System.Data.OleDb.OleDbDataAdapter().Fill(
      dataTable, (object) recordset );
    dataSet.Tables.Add(dataTable);
          }

  return dataSet;
}
```

Receiving the Posted Data

The service needs to parse the HTTP request back into an XML document. Listing 19 shows how this is done by passing the `HttpContext`'s `Request` object's `InputStream` into an `XmlReader`, which is used to load an `XmlDocument`. Listing 20 shows how to convert the XML back into a Recordset, and Listing 21 shows how to convert the Recordsets into a DataSet. The main point to note in Listing 21, `ConvertXmlDocumentToDataSet`, is that I take advantage of the `OleDbDataAdapter`'s `Fill` method. An overload of the `OleDbDataAdapter`'s `Fill` method actually takes an ADO Recordset and fills a DataTable from it, so I didn't need to write code to do a manual conversion. Once the data is in a DataSet, it is ready to be used by your .NET functionality. This completes the circle of communications between .NET and a VBA application—any data that you can express as a DataSet or Recordsets can be passed to the lite web services and/or received from them.

Security & Deployment

Because lite web services are hosted by IIS and ASP.NET, you can take advantage of your existing security infrastructure. You can set up IIS to use SSL and then make the request URL use HTTPS instead of HTTP. You can also use authentication—recall that the `Open` method of `XMLHTTPRequest` optionally takes a user and password. Even integrated windows authentication will work—if the IIS site and Web.Config files are configured for it, leave out the user and password in the `Open` method. The credentials of the user who is logged on to the workstation will be used for the call, or the user will be prompted by Internet Explorer to provide credentials. But in either case, you can make use of integrated windows authentication without any extra coding.

One note on deployment—there is an issue with deploying the ADO Interop assembly adodb.dll. The "Copy Local" option for deploying web server applications copies a particular assembly locally with the rest of the web application. The other option is to copy that assembly to the GAC (global assembly cache). If you have multiple ASP.NET web applications installed on the same machine, then adodb.dll must be installed in the GAC. You will get errors if more than one site tries to use local copies. See the KB article 321688 for more information (although it doesn't specifically address this issue, it discusses deploying adodb.dll).

Conclusion

In this article, I've shown how to convert data from ADO.NET into a form that can be used easily from VBA applications, and how to create lite web services to provide an interface between your Office VBA applications and your .NET functionality. You won't have to worry about providing the users with a potentially problematic setup program because this works with DLLs that are already installed on their machines. I successfully implemented this architecture in a hedge fund application that included a complex Excel spreadsheet for reporting purposes. The rest of the application was written in C# and ASP.NET, and I wanted the Excel component to access services in the .NET application, without any knowledge of the underlying database. I was able to use lite web services to easily create a secure way for the Excel application to communicate with the business layer in .NET. With lite web services, you now have the power to connect your Office applications to your .NET development efforts. **w::d**

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

Next Article
Table of Contents

Create Binary Behaviors for IE with .NET

Binary behaviors are an example of how .NET makes working with COM easier

STARTING WITH VERSION 5.0, Internet Explorer (IE) has provided an ability to customize the way HTML elements behave and display. In fact, it is possible to define completely custom elements with their own unique rendering behavior. For example, if you desire 2D drawings or 3D graphics in your HTML page, writing your own binary behavior gives you the power of GDI+ or DirectX for controlling the rendering of your elements. IE exposes its own device context and lets you draw directly on it.

IE exposes this capability through a set of COM interfaces, and developers of binary behavior have become familiar with such interfaces as `IElementBehavior` and `IHTMLPainter` for hooking up their behavior objects to IE. With the advent of .NET and COM Interop, it is now possible to implement binary behavior entirely in .NET—and with much greater ease. .NET reduces development hassle particularly through the transparency of COM Interop, which reduces the number of interfaces that a developer must explicitly implement. (Event handling is made especially intuitive.) Compared with the ATL-based sample in the Platform SDK, writing a .NET binary behavior is a snap.

In the new world of .NET web development, why would anyone feel compelled to write IE binary behaviors? There are at least two reasons. First, there is legacy code. In a typical example, an HTML application is already deployed, but then requires 2D drawings. Second, HTML is not going away, and IE provides a powerful and easy-to-use rendering engine. IE (MSHTML) may continue to be hosted in other applications, especially if element rendering is easily customizable through binary behaviors. .NET has so simplified the writing of binary behaviors that we may actually see an increased use of this IE feature.

SCOBIE P. SMITH *is a software consultant for Infinitiv, a research and development firm providing custom development services and offering specialties in .NET, linguistic processing, and data mining. He is currently completing a doctorate from Harvard in ancient Near Eastern languages. Contact Scobie at devcom@infinitiv.com.*

The Interfaces

In terms of COM, there are two components involved in writing a binary behavior: the behavior's class factory and the element behavior itself. The factory involves implementing two interfaces: `IElementBehaviorFactory` and `IElementNamespaceFactory`. `IElementBehaviorFactory` provides the `FindBehavior()` method, which hands IE an instantiation of the element behavior. `IElementNamespaceFactory` allows the factory to introduce new element names into the namespace of your HTML page through the `create()` method. That makes only two methods to implement for the factory.

The behavior component must also implement just two interfaces: `IElementBehavior` and `IHTMLPainter`. `IElementBehavior` has three methods to implement, providing the basic functionality of a binary behavior (without rendering): `Init()`, `Detach()`, and `Notify()`. `Init()` is called during initialization of the behavior object and is generally used to cache references back to interfaces implemented by IE (such as `IElementBehaviorSite`). The method `Notify()` is called when the element is ready for use (i.e., when the document is loaded). At this point, a reference to the element can be obtained, event-handling can be attached, and rendering can be triggered.

The second essential interface, `IHTMLPainter`, controls element rendering through four methods, though only these two are significant: `GetPainterInfo()` and `Draw()`. `GetPainterInfo()` allows the behavior to specify how rendering will be performed. This is where the drawing engine (e.g., GDI+, DirectX) is specified, as well as various rendering flags (e.g., transparency, clipping, or transformation). The real work of rendering the element's behavior is done in `Draw()`. `Draw()` is handed an `HDC` (device context), which can be used, for example, by GDI+ to define a `Graphics` object for vector drawing. The two remaining methods, `onresize()` and `HitTestPoint()`, are trivial for most basic rendering behaviors. `onresize()` should just invalidate the element's region, triggering repainting. `HitTestPoint()` is used only if you desire to identify where specifically in your element an event (e.g., mouse click) has occurred—in case some areas of your element should be distinguished from others. (You must set the `HTMLPAINTER_HITTEST` in `GetPainterInfo()` for this method to be called.)

So, all in all there are only four interfaces with five or six methods to implement, most of which are boiler-plate code. Well, there is one last interface, which is peripheral to the functionality of the behavior: `IObjectSafety`. The behavior's class factory should implement `IObjectSafety` in order to declare to IE that this component is safe to run. It has but two methods, which at most set some flags: `GetInterfaceSafetyOptions()` and `SetInterfaceSafetyOptions()`. So much is well known to writers of binary behaviors.

Defining the Interfaces for .NET

Most of the interfaces we need are available in the primary interop assembly for MSHTML, which comes with Visual Studio .NET. Just add an assembly reference to the `Microsoft.mshtml` assembly, and the many COM interfaces of MSHTML are accessible as .NET interfaces. At this point, the above factory interfaces, as well as `IElementBehavior` and `IHTMLPainter`, are ready to use.

`IObjectSafety`, on the other hand, is not supplied in the PIA and must be manually included for the sake of your behavior factory.

Listing 1
Implementing IElementBehavior

```
namespace BinBehaviors
{
  [
  ComVisible(true),

    IElementBehaviorSiteOM bsiteOM;
    IHTMLPaintSite paintsite;

    IHTMLDocument2 document;
    IHTMLWindow2 parwindow;

    IHTMLElement element;
    HTMLElementEvents_Event events;

    Point point1, point2;
    float penwidth;
    Color pencolor;

    int xoffset, yoffset;
    Bitmap bitmap;

    /// <summary>
    /// A default constructor (no parameters) is necessary for COM instantiation
    /// </summary>
    public Line()
    {
      point1 = new Point(0, 0);
      point2= new Point(0, 0);
      penwidth = 1;
      pencolor = Color.Black;

      xoffset = yoffset = (int) this.penwidth + 1;
    }

    ~Line()
    {
    }

    /// <summary>
    /// Gets or sets the end points of the line as a single string.
    /// </summary>
    public string coords
    {
      get
      {
        return String.Format("{0} {1} {2} {3}",
                             point1.X, point1.Y, point2.X, point2.Y);
      }
      set
      {
        UpdateCoords(value);
        DoLineDraw();
      }
    }

    // IElementBehavior method.
    void mshtml2.IElementBehavior.Init(IElementBehaviorSite behavsite)
    {
      try
      {
        this.bsite = behavsite;
        if (behavsite is IElementBehaviorSiteOM)
          this.bsiteOM = (IElementBehaviorSiteOM) behavsite;

        if (behavsite is mshtml.IHTMLPaintSite)
          this.paintsite = (mshtml.IHTMLPaintSite) behavsite;
      }
      catch (System.ExecutionEngineException e)
      {
        MessageBox.Show(e.Message);
      }
    }

    // IElementBehavior method.
    void mshtml2.IElementBehavior.Notify(int lEvent, System.IntPtr pVar)
    {
      try
      {
        switch ((_BEHAVIOR_EVENT)lEvent)
        {
          case _BEHAVIOR_EVENT.BEHAVIOREVENT_DOCUMENTREADY:
            if (this.bsite == null)
              return;

            this.element = this.bsite.GetElement(); // Cache the HTML element.
            this.document = (IHTMLDocument2) this.element.document;
            this.parwindow = (IHTMLWindow2) document.parentWindow;

            // Drawing to absolute point locations
            // requires absolute position style.
            element.style.setAttribute("position", "absolute", 0);

            // Hook up event handlers.
            if (element is mshtml.HTMLElementEvents_Event)
            {
              this.events = (HTMLElementEvents_Event) this.element;
              events.onpropertychange +=
                new mshtml.HTMLElementEvents_onpropertychangeEventHandler(
                  HTMLElement_OnPropertyChange);
            }

            // Initial attribute values.

            // All HTML attributes come in as strings (or null, if not present).
            // "coords" property is ONLY a property, not an attribute.
            string x1 = element.getAttribute("x1", 0) as string;
            string y1 = element.getAttribute("y1", 0) as string;
            string x2 = element.getAttribute("x2", 0) as string;
            string y2 = element.getAttribute("y2", 0) as string;

            if (x1 != null)
              point1.X = Convert.ToInt32(x1);
            if (y1 != null)
              point1.Y = Convert.ToInt32(y1);
            if (x2 != null)
              point2.X = Convert.ToInt32(x2);
            if (y2 != null)
              point2.Y = Convert.ToInt32(y2);

            // ARGB must come in hexadecimal form.
            string argbstring = element.getAttribute("argb", 0) as string;
            if (argbstring != null)
              this.pencolor = Color.FromArgb(
                                (int) Convert.ToUInt32(argbstring, 16));

            // Width must be base 10.
            string widthstring =
              (string) element.getAttribute("width", 0) as string;
            if (widthstring != null)
            {
              this.penwidth = Convert.ToSingle(widthstring);
              xoffset = yoffset = (int) this.penwidth + 1;
            }

            DoLineDraw();

            break;

          case _BEHAVIOR_EVENT.BEHAVIOREVENT_FIRST:
            //MessageBox.Show("event: first");
            break;
          case _BEHAVIOR_EVENT.BEHAVIOREVENT_LAST:
            MessageBox.Show("event: last");
            break;
          case _BEHAVIOR_EVENT.BEHAVIOREVENT_APPLYSTYLE:
            MessageBox.Show("event: apply style");
            break;

          default:
            MessageBox.Show("event: (default)");
            break;
        }
      }
      catch (System.ExecutionEngineException e)
      {
        MessageBox.Show(e.Message);
      }
    }

    // IElementBehavior method.
    void mshtml2.IElementBehavior.Detach()
    {
    }


    /// <summary>
    /// Handle the element's OnPropertyChange event.
    /// Update the drawing to reflect the new property values.
    /// </summary>
    public void HTMLElement_OnPropertyChange()
    {
      IHTMLEventObj2 evo = (IHTMLEventObj2) parwindow.@event;

      string name = evo.propertyName.ToLower();
      object o = element.getAttribute(evo.propertyName, 0);

      switch (name)
      {
        case "x1":
          if (o is System.Int32)
```

(The behavior object itself does not need this.) The interface looks like this:

```
// IObjectSafety interface.
[
ComImport,
Guid("CB5BDC81-93C1-11CF-8F20-00805F2CD064"),
InterfaceType(
    ComInterfaceType.InterfaceIsIUnknown)
]
public interface IObjectSafety
{
    int GetInterfaceSafetyOptions(
         ref Guid riid,
         out int pdwSupportedOptions,
         out int pdwEnabledOptions);
    int SetInterfaceSafetyOptions(
          ref Guid riid,
          int dwOptionsSetMaks,
          int dwEnabledOptions);
}
```

Notice that the methods are defined so as to return their HRESULT, rather than transforming errors into exceptions. As a result, we will need the PreserveSig attribute on our methods; for example:

```
// IObjectSafety method.
[return:MarshalAs(UnmanagedType.Error)]
[PreserveSig]
public int GetInterfaceSafetyOptions(
    ref Guid riid,
    out int pdwSupportedOptions,
    out int pdwEnabledOptions)
{
    // INTERFACESAFE_FOR_UNTRUSTED_CALLER
    // and _DATA.
    pdwSupportedOptions = 0x00000001 | 0x00000002;
    pdwEnabledOptions = 0x00000001 | 0x00000002;
    return 0; // S_OK.
}
```

So far, so good. At this point, it would appear that we have all the interfaces in hand, but this is not quite so. It turns out that two interfaces defined in the Microsoft.mshtml PIA need to be manually adjusted so as to marshal method parameters properly. For some method parameters, we need to preserve the COM-based semantics, which might not translate well into the expected .NET type. For example, when the semantics of the COM type are overloaded by using NULL as a meaningful value, we will have trouble if we marshal this as an Object reference. This is where the transparency of Interop can be tricky and requires special attention.

In particular, the IElementBehavior interface is faulty in the Notify() method. The default signature is:

```
void Notify(System.Int32 lEvent, ref
          System.Object pVar);
```

The second parameter, though, corresponds to a pointer to VARIANT and so may be NULL. When Notify() is called with NULL, an exception will occur. To adjust for these semantics, the methods must be defined this way:

```
void Notify(
    System.Int32 lEvent,
    System.IntPtr pVar);
          // Not: System.Object pVar.
```

This is a classic scenario in which we need to use IntPtr in order to allow for the original NULL semantics of the pointer. (See *.NET and COM*, by Adam Nathan, Pearson Education 2002, p. 264.)

The corrected IElementBehavior interface is, therefore, as follows:

```
// Corrected IElementBehavior.
[
ComImport,
Guid("3050F425-98B5-11CF-BB82-00AA00BDCE0B"),
InterfaceType(
    ComInterfaceType.InterfaceIsIUnknown)
]
public interface IElementBehavior
{
    void Init(
        mshtml.IElementBehaviorSite
        pBehaviorSite);
    void Notify(
        System.Int32 lEvent,
        System.IntPtr pVar); // CORRECTED.
    void Detach();
}
```

We need to place this and other corrected interfaces in a new namespace (such as mshtml2) in order to distinguish them from their synonyms in Microsoft.mshtml. Since

Listing 1
Continued

```
            point1.X = (int) o;
          break;
        case "y1":
          if (o is System.Int32)
            point1.Y = (int) o;
          break;
        case "x2":
          if (o is System.Int32)
            point2.X = (int) o;
          break;
        case "y2":
          if (o is System.Int32)
            point2.Y = (int) o;
          break;
        case "coords":  // NOT CALLED, because of our coords property (above)
          UpdateCoords((string) o); // Object is a string (BSTR).
          break;
        case "width":
          if (o is System.Int32)
            penwidth = (int) o;
          break;
        case "argb":
          // ARGB will come as double if too big to fit as Int32.
          if (o is System.Double)
            pencolor = Color.FromArgb((int) Convert.ToUInt32((double) o));
          else if (o is System.Int32)
            pencolor = Color.FromArgb((int) o);
          break;
        default:
          break;
      }

      DoLineDraw();
    }

    /// <summary>
    /// Update the coordinates members (x1, y1, x2, y2)
    /// given a string representation.
    /// </summary>
    /// <param name="coords">String representation of x1, y1, x2, y2.</param>
    private void UpdateCoords(string coords)
    {
      // May have: coords="  x1 ;y1 , x2 y2  ", etc.
      string s = coords.Trim();
      Regex re = new Regex("\\s*[,;]\\s*|\\s+"); // Separate at: space | punct.
                                                 // Order of alts is important.
      string[] segs = re.Split(s, 5);
      if (segs.Length >= 4)
      {
        point1.X = Convert.ToInt32(segs[0]);
        point1.Y = Convert.ToInt32(segs[1]);
        point2.X = Convert.ToInt32(segs[2]);
        point2.Y = Convert.ToInt32(segs[3]);
        UpdateAttributes();
      }
    }

    /// <summary>
    /// Update the HTML attributes to reflect actual property values.
    /// </summary>
    private void UpdateAttributes()
    {
      this.element.setAttribute("x1", point1.X.ToString(), 0);
      this.element.setAttribute("y1", point1.Y.ToString(), 0);
      this.element.setAttribute("x2", point2.X.ToString(), 0);
      this.element.setAttribute("y2", point2.Y.ToString(), 0);
    }

    //   .
  }
}
```

we will be using interfaces from both mshtml and mshtml2 namespaces, we will need to qualify our interfaces with the new namespace explicitly.

This correction now affects IElementBehaviorFactory, whose FindBehavior() method returns an IElementBehavior. Since we want this to return a corrected IElementBehavior and not the mshtml.IElementBehavior, we need to adjust this interface as well (trivially):

```
// Corrected IElementBehaviorFactory.
[
ComImport,
Guid("3050F429-98B5-11CF-BB82-00AA00BDCE0B"),
InterfaceType(
    ComInterfaceType.InterfaceIsIUnknown)
]
public interface IElementBehaviorFactory
{
    // CORRECTED METHOD: Return corrected
    // interface.
    mshtml2.IElementBehavior
        FindBehavior(System.String bstrBehavior,
                     System.String bstrBehaviorUrl,
                     mshtml.IElement
                         BehaviorSitepSite);
}
```

Finally, we need to make a similar adjustment in the IHTMLPainter interface. The Draw() method is given to us by mshtml as:

```
void Draw(mshtml.tagRECT rcBounds,
        mshtml.tagRECT rcUpdate,
        System.Int32 lDrawFlags,
        ref mshtml._RemotableHandle hdc,
        System.IntPtr pvDrawObject);
```

However, the handle to device context (hdc) provided by IE (and as used in GDI+) is ultimately just a pointer. We use the HDC to create a Graphics object using the static method Graphics.FromHdc(System.IntPtr hdc). This fact gives away the answer for the correction to make. We simply need to revise this parameter type to IntPtr. The newly defined interface now appears thus:

```
// Corrected IHTMLPainter.
[
ComImport,
Guid("3050F6A6-98B5-11CF-BB82-00AA00BDCE0B"),
InterfaceType(
    ComInterfaceType.InterfaceIsIUnknown)
]
public interface IHTMLPainter
{
    // CORRECTED METHOD: Use IntPtr,
    // not ref _RemotableHandle hdc.
    void Draw(mshtml.tagRECT rcBounds,
              mshtml.tagRECT rcUpdate,
              System.Int32 lDrawFlags,
              System.IntPtr hdc,
              System.IntPtr pvDrawObject);
    void onresize(mshtml.tagSIZE size);
    void GetPainterInfo(
        out _HTML_PAINTER_INFO pInfo);
    void HitTestPoint(
        mshtml.tagPOINT pt,
        out System.Int32 pbHit,
        out System.Int32 plPartID);
}
```

As this interface is not referred to in other mshtml interfaces we will need, this is the last correction we need. We are now ready to implement.

For all these interfaces that we have just defined, the Guid attribute provides the COM interface ID (IID) expected for the given interface. That is, these GUIDs are not arbitrary, but are the IIDs used when IE calls QueryInterface().

Implementing the Behavior

As an example, let us implement a simple line-drawing element, which uses the vector graphics of GDI+ to draw a line segment between two points given by left/top pixel locations. Making the line element a binary behavior means that the drawing takes place when IE renders the HTML document, giving us control to draw on a transparent background and to preserve z-index ordering. To do this, we define a Line class that implements our two mshtml2 interfaces, IElementBehavior and IHTMLPainter. Let us design our line element to have four attributes (x1, y1, x2, y2) to describe the two end points of the line. We will also provide a single, string-based property (coords) by which we can specify all four coordinates in one string. For variety, let us also provide stroke width and stroke color attributes. Our Line class, therefore, will have to maintain state information for all these attributes, in addition to caching some housekeeping information for drawing and communicating with IE. The class starts out as in Listing 1.

After the element constructor is called, Init() is executed, which caches the behavior site interface, allowing us to call IE for more information. At various points during the loading of the HTML document, IE calls Notify(), each time with a different lEvent flag. (Actually, only calls with the FIRST and DOCUMENTREADY event flags occur in this example.) The crucial call for us is when the document has been loaded, when lEvent equals BEHAVIOREVENT_DOCUMENTREADY. At this point we know that the document, element, and parent windows are available, and references to these can be saved for future use. Now is also the time we can initialize our element with any default attribute values of our choosing. In our case, we opt to force our line element to have absolute positioning style, which will save us the trouble of setting this style in our HTML to enable drawing to absolute pixel locations.

Most importantly, we can now hook up event handlers for all the events to which we wish our element to respond. For more interactive elements, one might wish to respond to onclick, onmouseenter, or onmouseleave—note the many events offered by HTMLElementEvents_Event. In our case, we only need onpropertychange, which will allow us to accept DHTML-based changes to the attribute values of our line. Since the COM-callable wrapper (CCW) provides default implementations for IDispatch and IEventSink, we are saved this hassle and can simply hook up event handlers in the C# way.

Keep in mind the difference between HTML attributes and object properties. Element attributes are maintained in the HTML DOM. When script alters the state of attribute data in the DOM, our underlying object is also informed via the onpropertychange event. Properties, on the other hand, are maintained directly by the object. Hence, attributes are available in HTML tags, while properties are available only in script. In our example, Line captures the initial attribute values (from the HTML tag) in Notify() and further changes in HTMLElement_OnPropertyChange(). The property coords, however, is simply implemented as a C# property. (When end points are modified through the coords property, the new coordinates must be copied into the x1, y1, x2, and y2 attributes.) COM Interop handles

Listing 2 Implementing IHTMLPainter

```
0123456789012345678901234567890123456789012345678901234567890123456789012345678
namespace BinBehaviors
{
  [
  ComVisible(true),
  ClassInterface(ClassInterfaceType.AutoDispatch),
  Guid("70F8ECDA-3869-4586-958E-0914547E9984"),
  ProgId("BinBehaviors.line")
  ]
  public class Line : mshtml2.IElementBehavior, mshtml2.IHTMLPainter
  {

    //   .


    // IHTMLPainter method.
    void mshtml2.IHTMLPainter.Draw(tagRECT rcBounds, tagRECT rcUpdate,
                    int lDrawFlags, System.IntPtr hdc, System.IntPtr
                       pvDrawObject)
    {
      if (this.bitmap != null)
      {
        Graphics g = Graphics.FromHdc(hdc);
        g.CompositingMode = CompositingMode.SourceOver;

        // Apply any scaling, etc. to the output.
        mshtml._HTML_PAINT_DRAW_INFO info;
        this.paintsite.GetDrawInfo(
          (int) _HTML_PAINT_DRAW_INFO_FLAGS.HTMLPAINT_DRAWINFO_XFORM |
          (int) _HTML_PAINT_DRAW_INFO_FLAGS.HTMLPAINT_DRAWINFO_UPDATEREGION,
          out info);
        Matrix xform = new System.Drawing.Drawing2D.Matrix(
          info.xform.eM11, info.xform.eM12,
          info.xform.eM21, info.xform.eM22,
          info.xform.eDx - rcBounds.left, info.xform.eDy - rcBounds.top);
        g.Transform = xform;

        // Update clipping region.
        Region clip = new Region();
        if (info.hrgnUpdate != System.IntPtr.Zero)
        {
          Region updateclip = Region.FromHrgn(info.hrgnUpdate);
          clip.Intersect(updateclip);
          clip.Translate(rcBounds.left, rcBounds.top);
        }
        g.SetClip(clip, CombineMode.Replace);

        g.DrawImage(this.bitmap,
          rcUpdate.left, rcUpdate.top,
          new Rectangle(rcUpdate.left - rcBounds.left,
          rcUpdate.top - rcBounds.top, rcUpdate.right - rcUpdate.left,
          rcUpdate.bottom - rcUpdate.top),
          GraphicsUnit.Pixel);

        g.Dispose();
      }

    }

    // IHTMLPainter method.
    void mshtml2.IHTMLPainter.onresize(tagSIZE size)
    {
      DoLineDraw();
    }

    // IHTMLPainter method.
    void mshtml2.IHTMLPainter.GetPainterInfo(out _HTML_PAINTER_INFO pInfo)
    {
      pInfo.lFlags = (int) (_HTML_PAINTER.HTMLPAINTER_TRANSPARENT |
        _HTML_PAINTER.HTMLPAINTER_NOPHYSICALCLIP
        | _HTML_PAINTER.HTMLPAINTER_SUPPORTS_XFORM);
          // Possibly also: | _HTML_PAINTER.HTMLPAINTER_HITTEST.

      pInfo.lZOrder = (int) _HTML_PAINT_ZORDER.HTMLPAINT_ZORDER_REPLACE_ALL;
      pInfo.iidDrawObject = Guid.Empty; // No drawing object; using GDI+.
      pInfo.rcExpand.left = pInfo.rcExpand.right =
      pInfo.rcExpand.top = pInfo.rcExpand.bottom = 0;
    }

    // IHTMLPainter method.
    // Not called unless pInfo.lFlags
    // included _HTML_PAINTER.HTMLPAINTER_HITTEST.
    void mshtml2.IHTMLPainter.HitTestPoint(
      tagPOINT pt, out int pbHit, out int plPartID)
    {
      pbHit = 0;
      plPartID = 0;
    }

    /// <summary>
    /// Prepare bitmap image to be drawn.
    /// </summary>
    private void Compose()
    {
      // The element origin is at top left always.
      // This bitmap drawing is relative to the origin.
      int width = Math.Abs(point2.X - point1.X);
      int height = Math.Abs(point2.Y - point1.Y);

      // The end points relative to the rectangle enclosing the line:
      int x1, y1, x2, y2;
      if (point2.X < point1.X)
      {
        x1 = point1.X - point2.X;
        x2 = 0;
      }
      else
      {
        x1 = 0;
        x2 = point2.X - point1.X;
      }
      if (point2.Y < point1.Y)
      {
        y1 = point1.Y - point2.Y;
        y2 = 0;
      }
      else
      {
        y1 = 0;
        y2 = point2.Y - point1.Y;
      }

      // Element/bitmap is offset left by penwidth to give room
      // for line, so line x1, x2 start at +penwidth.
      x1 += this.xoffset;
      x2 += this.xoffset;

      y1 += this.yoffset;
      y2 += this.yoffset;

      // IMPORTANT: Free the bitmap by force or
      // else large bitmaps remain allocated.
      // GC permits increasing amounts of
      // memory to be left allocated between each
      // successive collection, until physical RAM is consumed.
      if (this.bitmap != null)
        this.bitmap.Dispose();

      if (width == 0 && height == 0) // Skip bitmap if zero; invalid size.
        return;

      // Add room to bitmap to correspond to element size and to avoid clipping.
      width += 2*this.xoffset;
      height += 2*this.yoffset;

      this.bitmap = new Bitmap(width, height, PixelFormat.Format32bppArgb);

      Graphics g = Graphics.FromImage(this.bitmap);

      g.CompositingMode = CompositingMode.SourceOver;

      g.DrawLine(new Pen(pencolor, penwidth), x1, y1, x2, y2);
      g.Dispose();
    }

    /// <summary>
    /// Construct the drawing on its bitmap and adjust the containing element.
    /// </summary>
    private void DoLineDraw()
    {
      Compose();
      UpdateElement();
      // Invalidate so entire element must be redrawn.
      this.paintsite.InvalidateRegion(IntPtr.Zero);
    }

    /// <summary>
    /// Update element position and dimensions to fit the drawing.
    /// </summary>
    private void UpdateElement()
    {
      // Position line element at min of point1 and point2 coords.
      this.element.style.left = Math.Min(point1.X, point2.X) - this.xoffset;
      this.element.style.top = Math.Min(point1.Y, point2.Y) - this.yoffset;

      this.element.style.width =
        Math.Abs(point2.X - point1.X) + 2*this.xoffset;
      this.element.style.height =
        Math.Abs(point2.Y - point1.Y) + 2*this.yoffset;
    }

  }

}
```

the work of implementing automation (IDispatch) through its CCW. This difference presents a design decision: Which qualities of your element should be available in the HTML tag, and which only in script?

Once the element has been notified that the document is loaded, it is time to initiate drawing by informing IE that the element is in need of repainting. Calling InvalidateRegion() in the paint site interface provided by IE will do just this. (It would also be possible to call InvalidateRect(), but this method expects a ref to mshtml.tagRECT, which cannot be null. Yet, we need to pass in NULL to invalidate the entire element. We could redefine this interface to provide the InvalidateRect() signature we need, but InvalidateRegion() will also work.) Hence, we call this method every time we change the line drawing, in DoLineDraw(), as seen in Listing 2. When the element region is invalidated, IE will begin to make calls to Draw() on mshtml2.IHTMLPainter.

IE will, in fact, call Draw() more than once in drawing the element, as it paints the element rectangle in successive horizontal bands. For this reason, it is best to prepare an offscreen bitmap image before element invalidation, so that successive calls to Draw() deal with an unchanging graphic. When IE calls Draw(), the drawing requirements dictated by IE are obtained through GetDrawInfo(), and any necessary image transformation and clipping can then be applied before rendering with Graphics.DrawImage().

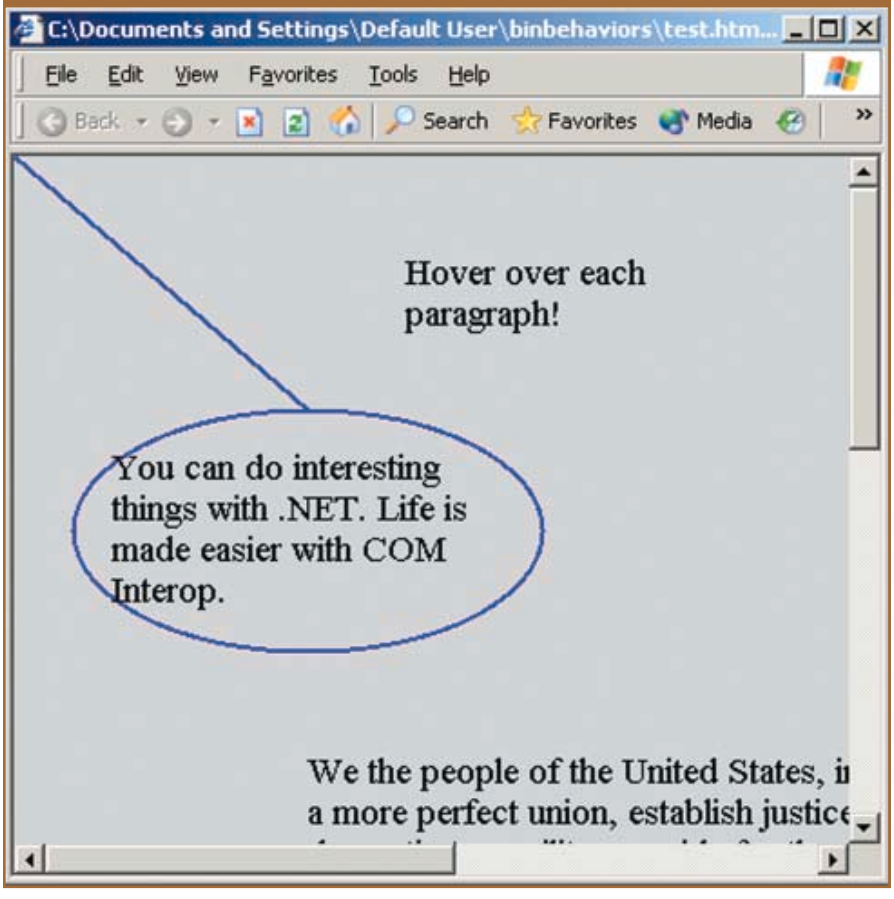
The interesting work of actually constructing the line segment is done in our Compose() and UpdateElement() methods. Compose() creates the bitmap (System.Drawing.Bitmap) representation of the line, where the end points are made relative to the bounds of the bitmap rectangle. UpdateElement() then sets the element dimensions to suit this bitmap and locates the element at the correct pixel location. The element and the bitmap are made slightly wider (by xoffset) and taller (by yoffset) than required by the minimum-size rectangle enclosing the line, because we must provide sufficient room for the stroke width of a vertical or horizontal line (right on the edge).

Of particular importance is the need to free the bitmap on each draw using the Dispose() method. Otherwise, the garbage collector permits large, unused bitmaps to accumulate, and with each garbage collection even larger accumulations are allowed.

Adding More Behaviors

It is easy to add behaviors in this component. First, a behavior class must be defined that implements mshtml2.IElementBehavior and mshtml2.IHTMLPainter. A new GUID is given this class to identify it for COM. Second, the factory class must supply the element namespace with a tag string naming the custom element and return a reference to an instantiation of the behavior. The tag string (element name) is provided via IElementNamespaceFactory.create(), which informs an HTML document how to refer to the element (in the namespace to which the factory will be assigned). In order to return the behavior reference, the factory method FindBehavior() merely constructs a behavior of the requested name and returns its reference. The code sample illustrates these steps in its second behavior, which can draw an ellipse.

Figure 1 Colored ellipse generated with binary behaviors



Sample HTML Page

Included with the source code is a sample HTML file, which employs our binary behaviors to draw a tethered ellipse around any chosen paragraph (see Figure 1). The <object> element instantiates the binary behavior class factory, using the factory's class ID as defined in our C# code. The <?import> tag directs HTML processing to query the behavior class factory for the implementation of any element whose name is in the specified namespace (behaviors). We can now introduce our custom elements by the names we specified in C#, provided we prefix them with the namespace designation (e.g., <behaviors:line>). Two custom elements are given, drawing a colored ellipse with a line segment. Each paragraph in the document responds to its onmouseover event (when the mouse hovers over the paragraph) and calls OnMouseOver(). The binary behaviors are then invoked to shape the colored ellipse around the given paragraph, with the colored line segment tethering the ellipse to the origin of the client window.

Conclusion

.NET COM Interop takes the headache out of getting binary behaviors to work in Internet Explorer. Several standard interfaces are automatically implemented by the default CCW. Event handling and properties are transparently handled in the normal C# way. Some interfaces supplied with the MSHTML primary interop assembly needs manual adjustment, but the guts of the behavior can be implemented easily in C#. Binary behaviors are another case where .NET makes working with COM easier. w::d

Next Article Table of Contents

| 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."

PopulateHelper: A Templated Solution for Managing Strings in Control

BY SHEHRZAD QURESHI
squreshi@picoliterinc.com

RECENTLY, I WORKED ON a project where, depending on the scenario, I either had to populate a list-box (`CListBox`) or a combobox (`CComboBox`) control with a list of strings acquired via a database connection. Of course, I also needed the ability to extract the string at a given index within the control. I developed a class that, given an STL vector of strings, would fill a control with these strings. `CListBox` and `CComboBox` are MFC objects that both derive from `CWnd`, and not a super-class that offers an `AddString()` method, for example. Thus, if I wanted to reduce the amount of methods in my class, the traditional polymorphic manner of collapsing related functionality into a single entity was not available to me. I could have implemented overloaded forms of my methods, as shown in Listing 1.

While the code shown in Listing 1 works, this solution simply cries out for a template-based solution, as shown in Listing 2. Note that in Visual C++ .NET 2003, template member functions in nontemplate classes are now fully supported.

This is simple enough and elegant. However, recall that I also wanted a means to extract the string given an index into the control. In reading the MSDN documentation, I was dismayed to learn that, while `PopulateHelper::getData()` worked when specialized for a `CComboBox` control, it would not compile when used with a `CListBox`. Why, you ask? The reason involves the call to `GetLBText()`, which is not provided in `CListBox` and, hence, the compilation error; see Listing 3. It's unclear why Microsoft chose the names `GetLBText()` for `CComboBox` and `GetText()` for `CListBox`, when the input arguments for the methods are identical (as one would suspect, given the fact that the two controls are so similar in functionality).

So now my conundrum was that if I wanted to implement a truly generic utility class (generic in the sense that it worked across both `CComboBox` and `CListBox` controls), I would have to fall back on C++ overloaded methods for the latter method, due to the fact that Microsoft's interfaces were not "orthogonal," so to speak. Inspired by Andrei Alexandrescu's *Modern C++ Design* (Addison-Wesley, 2001), where he describes how to architect extremely useful template classes or structs he deems "policies" (see Chapter 1), I implemented `PopulateHelper` as shown in Listing 4.

Granted, Listing 4 might contain more code than if I had overloaded `PopulateHelper::getData()` on both `CComboBox` and `CListBox`. However, in the interest of clarity, one could argue that `PopulateHelper::fill()` should be overloaded as well. Moreover, there are numerous other instances in MFC and other third-party APIs where this lack of orthogonality can cause problems. What I've shown here is how policy-driven class design can help steer the developer around these issues by using the template support of Visual C++ .NET 2003 to its fullest extent.

Safer Node Browsing with Microsoft's XML DOM

BY MATTHEW WILSON
matthew@synesis.com.au

WHEN WORKING WITH THE Microsoft XML DOM component, I encountered a rather nasty bug. Basically, when browsing child nodes—using the node-list object returned from `IXMLDOMNode::get_childNodes()`—it is sometimes useful to call the `IXMLDOMNodeList::reset()` method in order to ensure that any previous iteration is cancelled, and that iteration starts from the beginning of the node sequence.

However, when calling this method on a comment node (`nodeName: "#comment"`), it crashes somewhere (at address offset of 0x13447) inside the call to `reset()`. Other nodes with empty child sequences do not exhibit the same behavior.

There are a couple of workarounds. You could remember to refrain from enumerating the child nodes of comment nodes within your application code, but a much better solution is to make sure you always call the `IXMLDOMNodeList::get_length()` method and then only call `reset()` when the length is greater than 0. This is, presumably, pretty efficient, compared to calling, say, `IXMLDOMNodeList::get_item()` or `IXMLDOMNodeList::get__newEnum()` and having to release the objects returned. This technique can be codified in the following simple free function:

```
inline void reset_node_list(IXMLDOMNodeList *nodelist)
{
   long listLength;

   if( SUCCEEDED(nodelist->get_length(&listLength)) &&
       listLength > 0)
   {
       nodelist->reset();
   }
}
```

or within your wrapper class(es). In the `child_node_sequence` class in the XMLSTL DOM library (http://xmlstl.org/), I use the technique in the `begin()` method when starting a node sequence iteration.

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 First version of the PopulateHelper class

```
class PopulateHelper {

  // one for CComboBox
  void fill(CComboBox &ctrl, std::vector<std::string> &strs) {
          ctrl.ResetContent();
          for (int ii=0; ii<strs.size(); ++ii)
            ctrl.AddString(strs[ii]);
  }

  // body identical to above, just the method signature differs
  void fill(CListBox &ctrl, std::vector<std::string> &strs) {
          ctrl.ResetContent();
          for (int ii=0; ii<strs.size(); ++ii)
            ctrl.AddString(strs[ii]);
  }

};
```

The XML DOM component I have encountered this bug with is msxml3.dll, version 8.30.9926.0. It may or may not be present in other versions, but that is not really the point. In many deployment scenarios, you may not wish (or be able) to change the installed version of the XML DOM component, so you should code defensively and do the check.

Beware Null ListViewSubItems in .NET

BY MATTHEW WILSON
matthew@synesis.com.au

IN A RECENT .NET application—a simple network monitor using ICMP—I came across a potentially dangerous artifact of the behavior of the `System.Windows.Forms.ListViewItem` type. The `ListViewItem` type has a property, `SubItems`, that provides access to the subitems (which includes the main item itself) for the item as a collection of type `System.Windows.Forms.ListViewItem.ListViewSubItemCollection`. In order to add subitems to an existing item, you simply call `Add()` on the collection instance.

In my application, I wanted to be able to change the subitems in response to various events after the creation of the item. Further, I wanted to be able to change the order of "address" and "status" subitems at compile time by using the manifest constants ADDRESS_INDEX and STATUS_INDEX. This was to enable me to easily change the column order before the final version of the program.

Understandably, using the index operator of the collection to update a subitem resulted in a `System.OutOfRangeException` being thrown if the subitem did not already exist. This is the case even when you are specifying the "end" index (e.g., index 3 for an array with three items at 0, 1, and 2).

The solution to this is to create the subitems with appropriate initial values at the time of the item, and adding them to the collection, as in:

```
ListViewItem item = m_items.Items.Add(host);

... // sometime later

item.SubItems.Add("");
item.SubItems.Add("");

... // sometime later

subItems[ADDRESS_INDEX] = new ListViewItem.ListViewSubItem(...);
subItems[STATUS_INDEX]  = new ListViewItem.ListViewSubItem(...);
```

Being a C++ kind of guy, I didn't like the idea of creating subitems—in the call to `Add("")`—which were going to be thrown away. So I tried changing the calls to `Add()`:

Listing 2 A template-based version of PopulateHelper

```
class PopulateHelper {

  template <typename T>
  void fill(T &ctrl, std::vector<std::string> &strs) {
          ctrl.ResetContent();
          for (int ii=0; ii<strs.size(); ++ii)
            ctrl.AddString(strs[ii]);
  }

};
```

Listing 3 Compilation error with GetLBText()

```
class PopulateHelper {

  template <typename T>
  void fill(T &ctrl, std::vector<std::string> &strs) {
          ctrl.ResetContent();
          for (int ii=0; ii<strs.size(); ++ii)
            ctrl.AddString(strs[ii]);
  }

  template <typename T>
  CString getData(T &ctrl, int iIndex) {
          CString data;
          ctrl.GetLBText(iIndex, data); // oops, this won't work!
          return data;
  }

};
```

Listing 4 Policy-driven design of PopulateHelper

```
class PopulateHelper {

  // simple, because the interfaces for adding strings between
  // the two controls are orthogonal
  template <typename T>
  void fill(T &ctrl, std::vector<std::string> &strs) {
          ctrl.ResetContent();
          for (int ii=0; ii<strs.size(); ++ii)
            ctrl.AddString(strs[ii]);
  }

  // a place-holder
  template <typename T>
  struct GetTextFromCtrlPolicy {};

  // specialization for the CComboBox control
  template <>
  struct GetTextFromCtrlPolicy<CComboBox> {
static CString GetText(CComboBox &ctrl, int iIndex) {
                      CString retStr;
                      ctrl.GetLBText(iIndex, retStr);
                      return retStr;
          }
  };

  // specialization for the CListBox control
  template <>
  struct GetTextFromCtrlPolicy<CListBox> {
          static CString GetText(CListBox &ctrl, int iIndex) {
                      CString retStr;
                      ctrl.GetText(iIndex, retStr);
                      return retStr;
          }
  };

  // now it doesn't matter if this method is specialized on CComboBox
  // or CListBox, the correct call is resolved through the
  // GetTextFromCtrlPolicy struct.
  template <typename T>
  CString getData(T &ctrl, int iIndex) {
          CString data;
          data = GetTextFromCtrlPolicy<T>::GetText(ctrl, iIndex);
          return data;
  }

};
```

Listing 5
Listbox_front_inserter defined

```
struct listbox_front_inserter
{
public:
  ws_explicit_k listbox_front_inserter(HWND hwndListbox)
    : m_hwndListbox(hwndListbox)
  {}

#ifdef __STLSOFT_CF_MEMBER_TEMPLATE_FUNCTION
  template <ws_typename_param_k T>
  void operator ()(T &t)
  {
    insert(s);
  }
#else
  void operator ()(ws_char_a_t const *s)
  {
    insert(s);
  }
  void operator ()(ws_char_w_t const *s)
  {
    insert(s);
  }
#endif // __STLSOFT_CF_MEMBER_TEMPLATE_FUNCTION

// Implementation
protected:
  void insert(ws_char_a_t const *s)
  {
    ::SendMessage(m_hwndListbox, LB_INSERTSTRING,
                  0, reinterpret_cast<LPARAM>(s));
  }
  void insert(ws_char_w_t const *s)
  {
    ::SendMessage(m_hwndListbox, LB_INSERTSTRING,
                  0, reinterpret_cast<LPARAM>(s));
  }

protected:
  HWND  m_hwndListbox;
};
```



```
item.SubItems.Add((ListViewItem.ListViewSubItem)null);
item.SubItems.Add((ListViewItem.ListViewSubItem)null);
```

This works fine in creating the array (within the enclosing form's constructor), but when the form is `"Run()"`, the program terminates. The exception is, bizarrely, a `System.OutOfMemoryException`, though the additional information does give `"Error creating window handle"`.

From my point of view, this is an error. The call to `Add` with null should throw a meaningful exception—`System.NullReferenceException`—rather than the program dying later in what could, in a more complex application, seem to be an unrelated manner.

Solution: Don't use null subitems!

Inserter Function Objects for Windows Controls

BY MATTHEW WILSON
matthew@synesis.com.au

STL IS RECEIVING EVER-increasing recognition in the C++ development community as more STL-compliant software—in the form of containers, algorithms, adaptors, and function objects—becomes available for our use.

As part of the WinSTL project (http://winstl.org/), I've written function objects that provide back, front, and indexed insertion for list and combo boxes.

The `listbox_front_inserter` is defined in Listing 5. It can be used with any sequence that contains items that can be converted to `CStrings`, as in:

```
vector<CString> strings( . . . );
HWND            hwndListbox = . . .;

. . .

for_each(strings.begin(), strings.end(),
                  winstl::listbox_front_inserter(hwndListBox));
```

Each item will be inserted into the beginning of the list.

The `listbox_back_inserter`, which inserts items at the end of the list, has the same definition, but for the details of the sent message, as in:

```
::SendMessage(m_hwndListbox, LB_ADDSTRING,
          static_cast<WPARAM>(-1), reinterpret_cast<LPARAM>(s));
```

The `listbox_add_inserter` adds items into the list according to its sorted position, and uses:

```
::SendMessage(m_hwndListbox, LB_ADDSTRING,
               0, reinterpret_cast<LPARAM>(s));
```

There are corresponding function objects for working with combo boxes: `combobox_front_inserter`, `combobox_back_inserter`, and `combobox_add_inserter`.

Included in the source code archive is a program that demonstrates all six function objects by adding the items in the PATH environment variable to the list and combo boxes in front, back, and add forms. **w::d**

Next Article Table of Contents

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

How you plan to use a serviced component will affect its design

Design and Write Serviced .NET Components

IN THE .NET FRAMEWORK jargon, a serviced component is a managed class that can be hosted in a COM+ application in order to consume COM+ services. A serviced component can be written in any .NET-compliant language, but needs to inherit the `ServicedComponent` class in the `System.EnterpriseServices` namespace, either directly or indirectly.

The binding between the managed class and one or more COM+ services, such as automatic transactions, just-in-time activation, or asynchronous message queuing, is configured in a declarative manner. At design-time, you give the class all the service-related attributes you need and the .NET Framework enterprise infrastructure guarantees that instances of that class will be using those services at run time. Services can be further configured by calling proper methods on related classes and interfaces. The interaction between serviced objects and services suffers from no architectural limitation. COM+ services can also span over one or more objects. A typical example of this situation is when a transacted object extends the transaction to a fellow object that also supports (or requires) transactions.

Configuration information that links together services and objects is stored in the COM+ catalog. Based on that data, the COM+ runtime environment creates a context for the managed object whose execution, though, is still controlled by the .NET Common Language Runtime (CLR). In summary, each environment controls the implementation and the execution of respective interacting elements—the serviced object and the COM+ service. The context for the object is provided by COM+. Let's review the steps involved with the authoring of a serviced class in the .NET Framework.

Creating a Managed COM+ Class

To create a serviced component, you start by defining a class that inherits the `ServicedComponent` class. The inheritance doesn't have to be direct; you can also derive your final class from a class, which in turn inherits `ServicedComponent`. This scenario is quite common in all those cases in which the class must inherit from a base business class where the COM+ integration represents only a subset of the required features. To create a serviced component, you import the `System.EnterpriseServices` namespace and link the corresponding assembly. The following code snippet declares a class that can only operate within a transaction:

```
using System.EnterpriseServices;
[Transaction(TransactionOption.Required)]
public class Account : ServicedComponent
{
   [AutoComplete]
   public void Refresh()
   {
      :
   }
}
```

The same assembly can contain one or more serviced components. You compile the class into an assembly and deploy the serviced component either dynamically or manually. Dynamic registration consists of simply copying the assembly to the COM+ application's directory. No information is entered in the COM+ catalog at this time. It is important to notice that dynamic registration doesn't require that the assembly is copied to the Global Assembly Cache (GAC). Actually, managed clients (e.g., ASP.NET and Windows Forms applications) issue their first call to serviced components when they are not yet registered with COM+. The first time a client tries to create an instance of a serviced component, the .NET CLR extracts the type library out of the assembly and configures the COM+ catalog. The COM+ registration step takes place only once for each version of the assembly. After the registration step is completed, serviced and

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.*

nonserviced components are completely undistinguishable and managed clients treat them in the same way.

It is worth noting that on Windows 2000, the COM+ runtime always loads the newest CLR that is available on the machine. This means that if you have installed both the .NET Framework version 1.0 and 1.1, the latter is always loaded. The behavior is by design. It cannot be configured, but a workaround is easy to find. You can create a COM+ configuration file and use it to lock all applications to a specific version of the .NET Framework. The name of the configuration file is "dllhost.exe.configuration," where "dllhost.exe" is the process name for COM+ applications. Both Windows XP and Windows Server 2003 let you specify a configuration file on a per-application basis.

Manual registration entails using a command-line tool—regsvcs.exe—to load the assembly, generate a type library, and register the type library with the COM+ application. Manual registration is also possible to achieve programmatically. You use the `RegistrationHelper` class in the `System.EnterpriseServices` namespace and invoke the `InstallAssembly` method.

The location of the assemblies containing serviced components depends on the characteristics of the COM+ application you're setting up. If you create a server COM+ application, the assembly must be copied to the GAC. You can use a Windows Installer setup to do the job. The assembly and all of its dependency assemblies must be added to the GAC before the application is used.

Serviced components destined to a COM+ library can be deployed to the current directory. In this case, the CLR and the COM+ runtime can resolve any reference. Finally, serviced components invoked by pieces of an ASP.NET application can be copied in the Bin directory of the web application.

Essential Programming Guidelines

If you're going to write a serviced component that will only be called by other managed components, then no special consideration and limitation apply to your coding style. Instead, if your serviced component will have a COM object among its clients, it must adhere to a number of limitations. Here are the main ones.

The managed class must have only a basic, parameterless constructor. Parameterized constructors are not acceptable because there would be no way for the COM+ runtime to figure them out in detail and issue a proper call. Static methods are forbidden for a similar reason—managed static methods have no counterpart in COM. All the interfaces that the serviced component implement must be defined in the same component, and COM-specific HRESULT values must be included in user-defined exceptions.

A serviced component makes use of attributes to connect to COM+ services as well as COM+ related facilities provided by the .NET Framework. For example, a transacted component can use the `AutoComplete` attribute to automatically vote for the transaction end. The attribute causes the component to call into `SetComplete` if the method call returns normally. If the method call throws an exception, then the transaction is aborted using an automatic call to `SetAbort`. In any case, you don't have to insert explicit calls to both methods.

> On Windows 2000, the COM+ runtime always loads the newest CLR that is available on the machine

Automatic completion is effective for coding and component authoring, but it's not necessarily better for performance. Using automatic, implicit voting might result in a lengthier operation. To explicitly vote for a transaction, you can use the `ContextUtil` class in the `System.EnterpriseServices` namespace. The class exposes the well-known methods `SetComplete` and `SetAbort`, which you can use to explicitly commit or abort a transaction. `SetComplete` indicates that your component agrees to commit the work, whereas `SetAbort` indicates that your component has found a problem and needs to kill the current transaction.

Notice, though, that the effect of both `SetAbort` and `SetComplete` may be limited. A transaction is neither committed nor aborted until the root object of the transaction gets involved. Even if the component votes for committing the transaction, a single abort vote from any other participants will cause the entire transaction to fail. The following code snippet shows how a serviced component can use the `SetAbort` and `SetComplete` methods:

```
// Critical step for the transaction
If (!DoSomething())
{
   // Something went wrong
   ContextUtil.SetAbort();
}
else
{
   // Everything's OK
   ContextUtil.SetComplete();
}
```

COM+ Services can be activated at various levels—assembly, class, and method. For example, the aforementioned `AutoComplete` attribute works on a per-method basis. Other attributes apply to the class definition. Some of them are: `JustInTimeActivation`, `ObjectPooling`, `LoadBalancingSupported`, and `Transaction`. The `ApplicationQueuing` attribute works on a per-assembly basis and enables queuing support for the specified assembly. In light of this, the application can read messages from Message Queuing queues. Other attributes can be specified at the component level, per assembly, and per method. An example of these attributes is `SecurityRole`. You can use the `SecurityRole` attribute to add roles to an application and to components. Applied to the whole assembly, the attribute guarantees that the role is added to the COM+ catalog. Applied to a single component, the attribute ensures that the role exists in the application configuration.

Strong Names

So far we have repeatedly mentioned the registration step that each serviced component needs to accomplish in order to be used by the COM+ application. However, what does it mean to register, either dynamically or manually, a serviced component? The assembly that contains the serviced component must fulfill a number of criteria. First and foremost, the assembly must be strong-named. To sign an assembly with a strong name, you must have a cryptographic key pair made of a private and public key. The private key is used to sign the assembly, whereas the public key is used to read the signature. The .NET Framework comes with a command-line utility to generate a pair of such keys. The command line to use is the following syntax:

```
sn.exe -k mypair.snk
```

The `-k` switch instructs the tool to generate a new pair; the file name indicates the name of the binary file in which the keys will be persisted. Once you have a pair of keys, you can proceed and sign the assembly in either of two ways: You can use another command-line tool—al.exe—or declaratively bind the keys to the assembly being created.

The al.exe tool (the Assembly Linker tool) takes the name of the assembly and the keys and creates an assembly with a strong name. You can also sign the assembly during compilation by inserting the following line in the assemblyinfo.cs (or .vb) file that the Visual Studio .NET IDE creates for you:

```
[assembly:AssemblyKeyFileAttribute("mypair.snk")]
```

A strongly named file is a file with a unique name made of the public key, a digital signature, the assembly's text name, version number, and culture information, if any.

Aside from being strong named, the assembly that contains a serviced component must be registered in the Windows registry. In addition, type library definitions must be registered and installed into a specific COM+ application.

To successfully perform registration, some information is needed and includes the following: COM+ application identity, type of activation required, and a description. The application identity is a name or a GUID that identifies an existing COM+ application. You can specify the target application of a component by using the `ApplicationName` attribute.

```
[assembly: ApplicationName("BankComponent")]
public class Account : ServicedComponent
{
    :
}
```

The activation type determines whether the serviced component is created in-process or hosted in a new process. You use the `ApplicationActivation` attribute to specify the activation type:

```
[assembly: ApplicationName("BankComponent")]
```

The description is optional, but is often helpful to discern similar assemblies. Here's how to apply the `Description` attribute:

```
using System.EnterpriseServices;
[assembly: Description("BankComponent assembly")]
public class Account : ServicedComponent
{
      :
}
```

Other Serviced Components

With respect to transactions, along with classes, web services and ASP.NET pages can also be configured as participant objects. An attribute determines the transactional behavior of the instantiated object. They can support ongoing transactions, require one, always start a new transaction, or never participate in transactions.

For ASP.NET pages, the transaction support is disabled by default and is turned on by setting the @Page's `Transaction` attribute to any of the following values: `Supported`, `Required`, and `RequiredNew`. The `Supported` attribute indicates that the page participates in an ongoing transaction, if any. The `Required` attribute indicates that the page always requires a transaction and a new one is created if there's no ongoing transaction. Finally, the `RequiredNew` attribute instructs the run time to always create a new transaction in which the page acts as the root.

You can also run a `Web Service` method within the scope of an automatic transaction. To do so, you set the `TransactionOption` property of the `[WebMethod]` attribute. The attribute accepts the same values that are acceptable to an ASP.NET page. In spite of this, it's worth noticing that a `Web Service` method can only participate in a transaction as the root of a new transaction. This means that `Required` and `RequiredNew` are the same and the behavior is always that of creating a new transaction. Likewise, `Supported` and `None` are the same and no transaction is ever supported. If a `Web Service` method is participating in a transaction and an exception occurs, ASP.NET automatically aborts the transaction. If no error occurs, the transaction automatically commits. w::d

Choosing between a wrapper or a full port for your legacy code

Porting Your C++ Code to .NET

I AM NOT A fan of porting code because it produces, well, ported code. When you design your code, you generally have a target system in mind, and you use this system when you test your code. Your code is designed and tested on one system, and when you port it to another system, you attempt to make the code work in a situation for which it was not designed. However, there are situations when porting is required. .NET is clearly the future for Microsoft operating systems, but few software houses are likely to want to start afresh with their code: If you have hundreds of thousands of lines of code, it makes sense to try to reuse it. This is where managed C++ excels: It is designed to allow you to compile unmanaged code to Microsoft Intermediate Language (MSIL) so that it runs under the .NET runtime. In this article, I will outline the basic issues involved in porting your native C++ to .NET.

What Does Porting in .NET Mean?

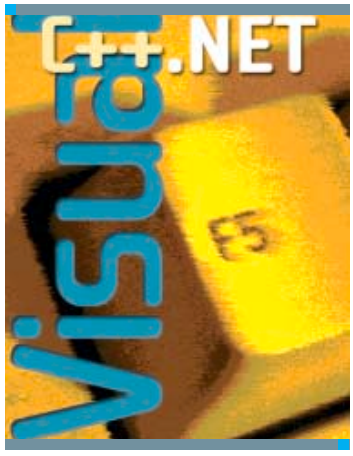
The first point to make about porting C++ code is that, in general, you are porting the code and not the data to .NET. What do I mean by this? When you compile C++ code with the /clr compiler switch, the compiler will generate MSIL for the code (there are some exceptions, but this statement is generally true). This means that all code, whether it be global functions or class methods, will run under the .NET runtime. However, if your classes are unmanaged (they are not marked with the `__gc` modifier), then instances will be created on the stack or in the C++ free store. Only instances of .NET types will be created on the managed heap and managed by the .NET Garbage Collector. Compiling native C++ with /clr is not a solution to fix poorly written C++ code that does not call delete on free store allocated objects!

Porting means giving .NET code access to native C++ routines. If you only have one managed C++ client to your code, then porting is unnecessary because such code should be accessed directly and the managed C++ compiler provides a technology called "It Just Works" (IJW) to facilitate this. On the other hand, if you want to use those routines in other .NET languages, or reuse them in several managed C++ projects, then the code needs to be ported to a managed library. Here, there are two choices: Do you provide a wrapper around the native code or do you port the entire library to managed code?

Porting Libraries

Typically, a C++ library is offered in one of three ways: a source library, for example, a template library; a static library; or a DLL. If you have a library provided in source form, the entire library can be compiled as MSIL; however, this will mean that only code within the current assembly will have access to the classes because to export a class out of an assembly, the class has to be a public .NET class. Furthermore, the method parameters and public data members of your class may be unmanaged types and this means that they will not be accessible by .NET code written in a language other than managed C++. To address this, you must provide a wrapper class that provides a managed interface to your class.

If your library is supplied as a static library, then you'll have to supply a wrapper class in managed C++ to enable other .NET languages to access the library. If the static library is class based, then you will have most of the work done for you because you do not have to identify the classes—you merely have to provide suitable wrappers.



If the library is DLL based, then in most cases it will be a collection of exported C functions. In such cases, you will have to do some work to identify the objects involved and design wrapper classes. In some cases, there may not be any objects and you can simply expose these functions through static members of your managed class. In other cases, the C API will represent a "flattening" of an object API. For example, in last month's column, I described the GDI+ unmanaged API, which is exported from the gdiplus.dll library using C functions. These functions use opaque handles to represent the `this` pointer of the object. The managed classes in the `System::Drawing` namespace are merely wrappers around this unmanaged library.

The major difference between a source code library and static or DLL libraries is that most of the code in a source code library will be compiled as MSIL and executed by the .NET runtime, whereas most of the code in a static library or a DLL will be x86 code running outside of the tentacles of the .NET runtime. This means that the code in a static library or DLL will be run in the environment where it was designed to run, so you don't have the issue of ported code being squeezed into an environment where it was never intended to run. Note also that Microsoft says that the thunking calls from the managed to the unmanaged world used in IJW or Platform Invoke add between 10 to 30 extra x86 instructions to the JITed code. A source code library may make many calls to other libraries (like the C Runtime library or the Win32 library), so compiling the entire code to MSIL will add the extra machine cycles to every unmanaged library call. On the other hand, when you call a static library, you only have those extra machine cycles when the static library methods are called from the wrapper class, and not when the library methods make CRT or Win32 calls.

Common Language Specification

The best library is one that can be used by the largest number of users. In .NET terms, this means that a library has to be Common Language Specification (CLS) compliant. The CLS rules of types and names should apply to all types that are publicly accessible, so your private types can be noncompliant. A compliant class exposes language features that are considered the baseline for all languages, so for example, since some languages cannot handle unsigned integer types, a compliant class should only use signed integers for method parameters, public fields, and properties.

RICHARD GRIMES *is an author and speaker on .NET. His latest book,* Programming with Managed Extensions for Microsoft Visual C++ .NET, *updated for Visual C++.NET 2003, is available now from Microsoft Press. He can be contacted at richard@richardgrimes.com.*

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…

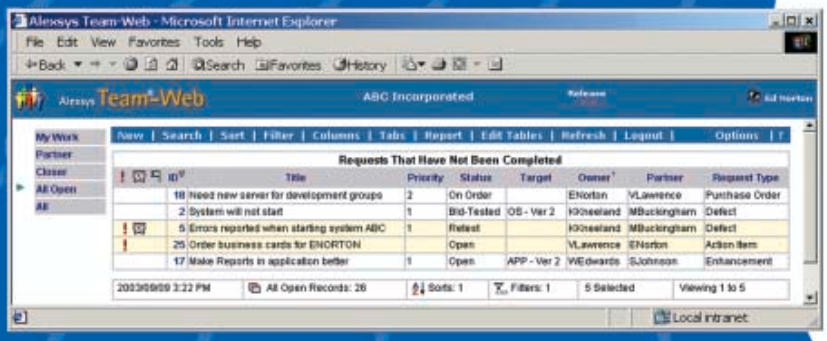
New Version Available
Team v2.5
With Group Access Control

... 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.

A CLS-compliant library should be declared as such with the assembly attribute `[CLSCompliant(true)]`; types and members of types that are not compliant should also be marked with this attribute but with False as the constructor parameter. The Tools Developers Guide installed as part of the .NET SDK provides a list of the CLS rules in the first part of the Common Language Infrastructure specification, so I won't go into details about them here. Instead, I will make a few general comments.

First, different languages have different rules about naming items. The CLS specification says that items should have unique names except when overloading is intended, so a single name cannot be used for a method and a field and two names must differ by more than just their case. Second, you can write a compliant class that has members that are targeted at a specific language (for example, methods that have unsigned integer parameters targeted at C# or managed C++) as long as it provides alternative members that are compliant. Providing CLS-compliant members of your wrapper class will involve extra work, but in the long run, it will be worthwhile because it will widen the potential market for the library.

Porting C++

If you have an existing C++ class and you want to expose this as a .NET class, the temptation is simply to write a wrapper class that uses composition to hold an instance of the native class and provide a managed implementation of each of the object's methods that do the necessary conversions between managed and unmanaged types before delegating the call to the unmanaged object. Although this approach is straightforward, there are problems with it because native C++ has some fundamental differences to managed C++.

Perhaps the most important difference is the lifetime issue: A native C++ object is explicitly destroyed, which results in a call to the object's destructor and frees the memory used by the object. The native C++ developer knows that, when the object is destroyed, the destructor will be called. This is not the case with .NET objects. The managed C++ compiler generates a method called `Finalize`, which calls the code in the C++ classes destructor. The `Finalize` method is called by the garbage collector at some time in the future when a garbage collection occurs. For objects that hold on to scant resources needed by other code, `Finalize` is unsatisfactory because clean up does not occur when the developer determines that it is necessary.

`Finalize` is protected, which means that the code that creates the instance does not have direct access to this clean up code, so you cannot get around this problem by calling it explicitly. The C++ compiler provides a public method called `__dtor` that calls the `Finalize` method, and since this method is public, client code can call it directly, or can call the `delete` operator on the object reference. Note that when you call `delete` on a managed C++ object, the result is merely to call `__dtor`: It has no effect on the memory occupied by the object. VB.NET and C# do not have the `delete` operator, but clients written in these languages can call the `__dtor` method directly. However, this is not natural to those languages and .NET has an established pattern to handle this situation.

If you have a native C++ class that implements a destructor, it means that the C++ class has resources that should be freed. In many situations, these resources should be freed as soon as possible, in which case the managed wrapper class should implement the `IDisposable` interface to provide a `Dispose()` method that will free the resource. C# developers have the `using` statement that identifies a block of code that determines the effective lifetime of an object and ensures that `IDisposable::Dispose()` is called after the block of code has completed.

Another issue involves the initialization of objects. In .NET, constructors are used solely for initialization when an object is created. Native C++ is a little more forgiving because it allows you to call constructors explicitly within your class. The most common use of this facility is when you have overloaded constructors that have common code, in which case one constructor can call another constructor to use its initialization routine. The managed C++ compiler does not allow you to call a constructor like this, and instead you have to generate a separate private initialization method.

Finally, it is worth pointing out that native C++ code provides a type of method (or constructor) overloading through parameter default values: Client code can omit the parameters with default values to get that value. .NET does not allow default parameters, but it does allow overloading, so the solution to this problem is to call the most generic overload and explicitly pass the default values.

Data Marshalling

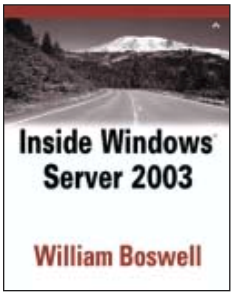
The public interface to your class must have managed types and preferably should have CLS-compliant types. The methods of your native C++ code will usually have unmanaged types, so you will need to convert your managed types into unmanaged types before calling the native code and convert the results into managed types. I will leave the details of data marshalling to another article, but in general, your friend here is the `Marshal` class in `System::Runtime::InteropServices`. This class provides static methods to convert unmanaged strings to managed strings and vice versa, and to allocate and deallocate arbitrary amounts of memory.

Summary

Managed C++ is the only .NET language that allows you to mix managed and unmanaged code, and this facility is provided to allow you to use existing native C++ code in .NET classes. The C++ compiler gives you the option of compiling an entire library as managed code, or to compile the bulk of the code as native x86 so that the code runs in the environment where it was designed to run. Next month, I will explain more about the porting process. w::d

Inside Windows Server 2003

William Boswell
1376 pages
Addison-Wesley Professional; 2nd edition
$59.99
ISBN: 0735711585
http://www.aw-bc.com/catalog/academic/product/0,4096,0735711585,00.html/



AS THE TITLE SUGGESTS, *Inside Windows Server 2003*, by William Boswell, is a comprehensive look at all the features in Windows Server 2003 and how you might deploy them. Accordingly, it covers the full gamut of subjects including installation and upgrades, DNS, Active Directory Services and Replication, designing and deploying Windows Server 2003 Domains, security, group policies, data storage, encryption, remote access, system failures, and more.

In some ways, you could consider Boswell's *Inside Windows Server 2003* a sequel to his previous work *Inside Windows 2000*. Indeed about 16 of the 21 chapters have the same titles and many of them have the same topic lists. New in *Inside Windows 2003* are chapters on Public Key Infrastructure (PKI) and on encryption. Each chapter opens with a bulleted section listing all the new features for Windows 2003 Server.

Boswell assumes the reader has a basic knowledge of Windows NT servers and classic NT domains. Without this background, the strongly motivated with an operating system might get by. The author points out that this book would not be a good place to start if you are just learning about networking for the first time, nor is *Inside Windows Server 2003* structured like an exam-cram type of text. However, because it includes most of the info required for Windows Server 2003 exams, it would be a handy study guide for those on the path. Readers primarily interested in IIS 6.0 would be better served by *Microsoft IIS 6.0 Administrator's Pocket Consultant*, by William R. Stanek (Microsoft Press, 2003).

The chapters on upgrading domains from NT4 and Windows 2000 were both lucid and relevant. The diagrams are particularly helpful in visualizing what's happening in each of these scenarios. The practical advice on how to prepare for upgrade failure is alone worth the price of the text. Nightmare scenarios such as the loss of a DNS server and domain controller are also covered in detail.

As you might expect, Active Directory (AD) is the single largest topic relevant to Windows Server 2003. This includes distinct chapters on AD services, replication, maintenance, and directory security. The author covers new features in security such as permission sets, the effective permission calculator, and smart card support.

In other areas of Windows Server 2003, such as file systems, it is sufficient to simply list improvements in performance, security, and support for newer devices like DVD-RAM. For administrators just moving from NT4, Boswell provides a bottom-up survey of NTFS internals from sectors to partitions.

Rather than taking a strictly how-to approach, the author provides in-depth background information on each subject as a motivation. After completing the functional descriptions, the text provides overviews of common operations. For example, in regard to file systems, this means step-by-step procedures for enabling file compression, reparse points, defragmentation, and NTFS conversion.

Inside Windows Server 2003 includes an exceptionally good index. This is a key ingredient in a useful server reference book. It provides more than 70 pages of entries at the detail of individual command-line switches, filenames, topics, and procedures. In a book of nearly 1400 pages, such a resource is indeed critical.

I found some of the advice in installation and upgrade chapters to be inapplicable in modern scenarios. Much of the troubleshooting involves obsolete hardware, such as multimedia cards and ISA busses, which I haven't seen on new computers in the past three years. I'm not sure why you would be spending $600 or more for an operating system to run on a machine that is worth far less than that.

Of course, I've barely scratched the surface of what's available to you in William Boswell's *Inside Windows Server 2003*. I would recommend this book most strongly to NT4 administrators who are making the great leap forward to Windows 2003 Server right now. It will also make a useful adjunct for those studying to become certified on this new, industrial-strength operating system. 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/.*

Table of Contents

ADVERTISER INDEX

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

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

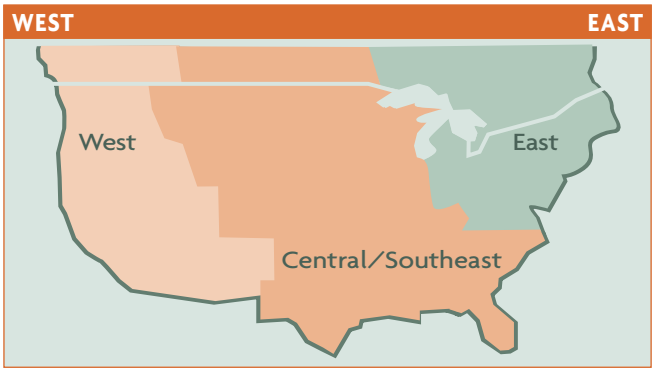
WEB: Go to the development tool page on our web site, www.windevnet.com. From there you can link to the advertisers below.

www.windevnet.com

windows:: developer NETWORK
APPLICATION DEVELOPMENT FROM WINDOWS TO WEB

ADVERTISER	PAGE
Alexsys Corporation	35
BeCubed Software, Inc.	40
Borland	1
Dice	33
dtSearch Corporation	5
Extended Systems	C4
FairCom	17
Intorel	36
Iocomp Software	13
Kinook Software Inc.	40
Marx Software Security	40
Melissa Data	40
Programmer's Paradise	9
Sourcegear	21
Tal Technologies Inc.	40
Thb Componentware	40
Windows Developer Network CD-ROM	2
Windows Developer Network Security E-Book	4

ADVERTISER	PAGE
Windows Developer Network Special Issue	C2
Xoreax Software	30
Zero G Software	C3



	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

Table of Contents

NEW PRODUCTS

Submit new product announcements to wdletter@cmp.com.

Velocitis Announces Flywheel 7.1

Flywheel is a process-agnostic tool for visualizing, designing, and refactoring Microsoft Visual C# .NET and Visual Basic .NET source code, which integrates with Microsoft Visual Studio .NET 2002 and 2003. Flywheel creates UML style visualizations directly from CSharp or VB.NET source code; there is no separate visualization model. The source code is the model, and real-time synchronization keeps visualizations and source code current. Flywheel Professional version 7.1 includes user interface enhancements, Visual Studio .NET integration improvements, and improved documentation.

Velocitis Inc.
325.347.0414
http://www.velocitis.com/

RainCode Updates XMLBooster

RainCode Corp. has added C# to the list of languages supported by XMLBooster, a product designed to provide more efficient integration of XML data within applications. XMLBooster statically generates XML parsers for C, C++, C#, COBOL, Delphi, Java, or Ada applications. By using native code generation rather than a generic approach, XMLBooster aims to improve performance for transaction-based applications. The software also generates structural documentation and a GUI to edit specific XML files; the GUI can be incorporated into the final application.

RainCode S.P.R.L.
32.2.522.06.63
http://www.xmlbooster.com/

Datalight Launches Reliance File System

Datalight has released Reliance, a fault-tolerant file system for embedded computers. Reliance can act as a standalone file system or run in parallel with other file systems. It arranges files and directories contiguously on the storage media in order to maintain a high level of performance and diminish the frequency of running system-defragmenting and check-disk utilities. The transactional system uses transaction points while updating and storing data, allowing the file system to hold data safely in an area of the disk that is not being used by the previous transaction while updates occur. At each transaction point, the data is kept intact until the update has been verified.

Datalight Inc.
425.951.8086
http://www.datalight.com/

Desaware Licensing System .NET Ships

The Desaware Licensing System is a licensing system for .NET based on cryptographic certificates. Designed for per server/machine and component licensing, it can be configured for both moderate and high security scenarios. With 128-bit cryptographic licensing, the Desaware Licensing System does not depend on hidden files, registry entries, or background services. Instead, installation codes are generated on a server for use during installation of Microsoft .NET Framework-based applications and components, and during activation, the code is verified by the server before it issues a licensing certificate bound to the client machines.

Desaware Inc.
408.377.4770
http://www.desaware.com/

Atalasoft Releases dotImage

Atalasoft has released dotImage for the Microsoft .NET Framework, a new imaging component containing a class library with hundreds of imaging-related features, as well as WinForm and Microsoft ASP.NET WebForm components and controls to display, manipulate, and print images. Built with managed C#, dotImage is not a COM or ActiveX wrapper and there are no ActiveX dependencies. Atalasoft is also working on a WebForm image control for Microsoft ASP.NET, which will mimic much of the functionality of the WinForm product in a server-side control.

Atalasoft Inc.
413.572.4443
http://www.atalasoft.com/

QA Systems Delivers QStudio for Java Pro 2.0

QStudio for Java is a source-code quality assessment tool for Java development organizations that integrates with JBuilder, Oracle 9i JDeveloper, Eclipse, WebSphere Studio, and Visual Age. QStudio for Java Pro 2.0 has two major enhancements: the ability to check for Enterprise Java Beans compliance and the ability to analyze incomplete source bases. QStudio for Java Pro supports over 200 rules by default, some of which can be used to instantiate new customized rules. QStudio also allows users to add their own rules (and the rules developed by the PMD community) using open-source PMD specification capabilities.

QA Systems Inc.
512.637.6100
http://www.qasystems.com/

Seapine Software Announces QA Wizard 2.1

Seapine Software has made QA Wizard 2.1 available for automated functional testing of Windows and web applications. Enhancements to QA Wizard 2.1 include functional load testing capabilities, international language support, and the ability to test Microsoft Windows applications. QA Wizard can search

Table of Contents

NEW PRODUCTS

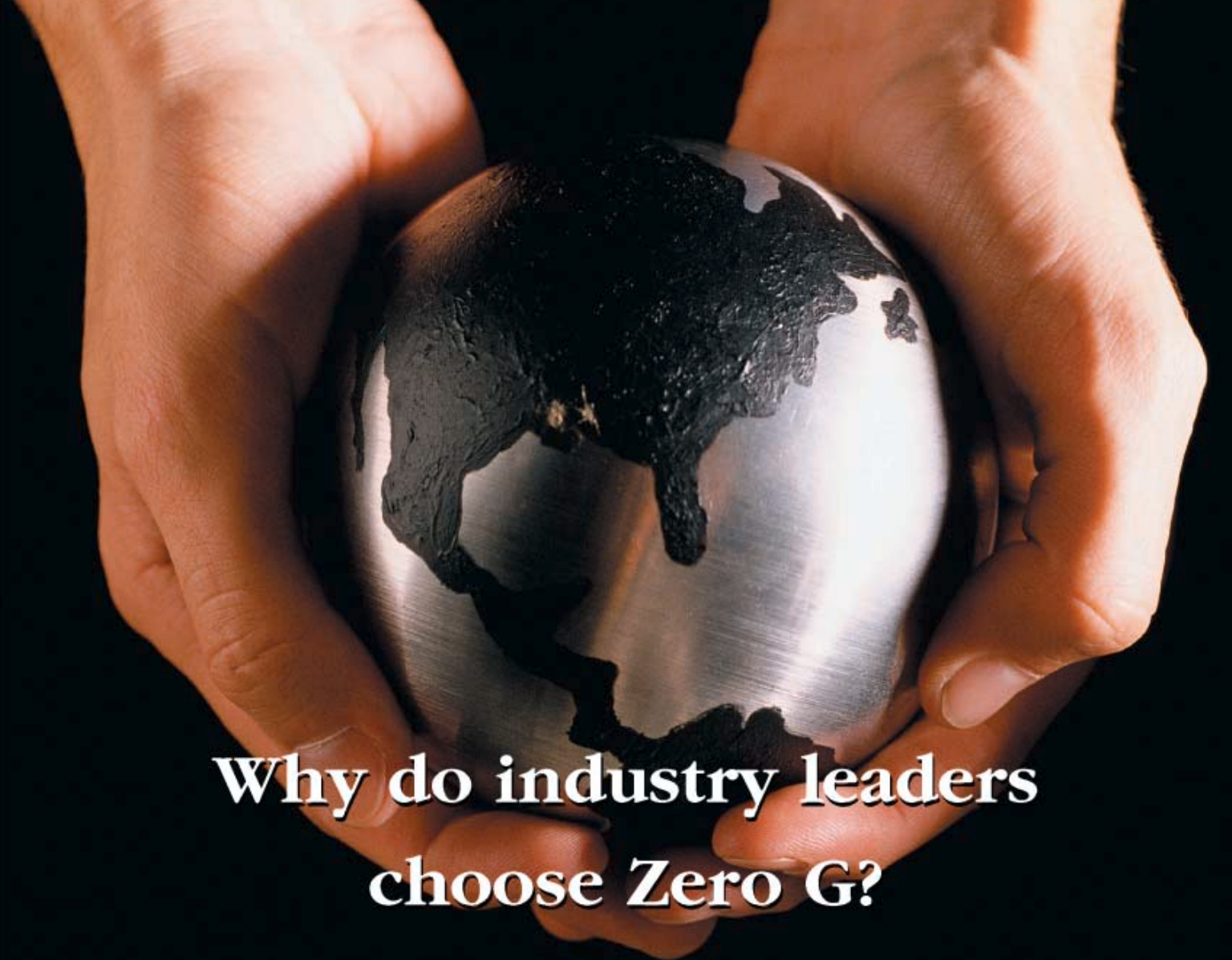
and bind to Windows objects, handling many user interface changes automatically. The software integrates with Seapine's Surround SCM and Microsoft's Visual SourceSafe.

Seapine Software Inc.
513.754.1655
http://www.seapine.com/

dtSearch 6.2 Released

The dtSearch Text Retrieval Engine lets developers add dtSearch's text retrieval capabilities to web-based and other applications. The dtSearch Engine supports SQL, ADO, C++, .NET, Delphi, and Java, and the 6.2 release has additional Java, C++ and .NET (ASP.NET, VB.NET, C++.NET, C#) code samples and extensions to the developer APIs. dtSearch offers over two dozen indexed, unindexed, fielded, and full-text search options, and the new release also includes forensic search enhancements: Text segments in large data blocks, such as those recovered through an "undelete" process, can be automatically parsed. Language recognition algorithms can detect text in a variety of languages.

dtSearch Corp.
301.263.0731
http://www.dtsearch.com/



Why do industry leaders choose Zero G?

Today's world of software design is complex, where a myriad of components - app servers, development languages, etc. - conspire to make application development more complicated than ever before. Zero G's solutions make deploying and delivering software applications less complicated, for both you and your customers. The industry award-winning InstallAnywhere and PowerUpdate from Zero G combine the functionality that enterprise software developers need with an unparalleled ease of use that saves time and money. That's why industry leaders like Sun, Novell and IBM choose us.

ZERO G
SOFTWARE

YOUR SOFTWARE DEPLOYMENT PARTNER

WWW.ZEROG.COM


Extended Systems®
going to
x t r e m e s
to provide you the ultimate database experience
Advantage Database Server is a robust client/server RDBMS that provides extreme performance, scalability and reliability. Experience the rush of developing with a database that provides navigational and SQL data access. Experience the excitement of deploying a cross-platform database that works on existing network infrastructures. Experience the thrill of easily creating triggers, stored procedures and referential integrity—without a DBA. Experience data management without the harsh costs of ownership. Experience Advantage, a solution that can enable your wildest adventures. Download a free trial version at: www.AdvantageDatabase.com/go/xtremewin or phone 800-235-7576, ext. 5030
ADVANTAGE™
DATABASE SERVER
See us at the Borland Conference, booth #321
2003 READERS CHOICE Delphi INFORMANT MAGAZINE BEST DATABASE CONNECTIVITY Advantage TDataSet Descendant Extended Systems
2002 READERS CHOICE Delphi INFORMANT MAGAZINE BEST DATABASE ENGINE Advantage Database Server Extended Systems
2001 READERS CHOICE Delphi INFORMANT MAGAZINE BEST DATABASE ENGINE Advantage Database Server Extended Systems
All trademarks and registered trademarks are the properties of their respective holders. ©2003 Extended Systems

Listing 4 Implementation of enumerator handlers

```
/* //////////////////////////////////////////////////////////
 * ...
 *
 * Extract from MOCtlFns.c
 *
 * Copyright (C) 1998-2003, Synesis Software Pty Ltd.
 * (Licensed under the Synesis Software Standard Public License:
 *  http://www.synesis.com.au/licenses/ssspl.html)
 *
 * ...
 * ////////////////////////////////////////////////////////// */

/* IEnumXXXX */

HRESULT EnumXXXX_Reset(LPVOID itf)
{
  LPENUMSTRING  pen = (LPENUMSTRING)itf;

  return pen->lpVtbl->Reset(pen);
}

/* IEnumString */

HRESULT EnumString_NextItem(LPVOID itf, LPEnum2WndItem pitem, LPVOID *pvalue)
{
  LPENUMSTRING  pen = (LPENUMSTRING)itf;
  HRESULT       hr  = pen->lpVtbl->Next(pen, 1, &pitem->olestr, NULL);

  if(hr == S_OK)
  {
    *pvalue = pitem->olestr;
  }

  return hr;
}

void EnumString_ClearItem(LPVOID pen, LPEnum2WndItem pitem, LPVOID value)
{
  CoTaskMemFree(pitem->olestr);
}

/* IEnumBSTR */

HRESULT EnumBSTR_NextItem(LPVOID itf, LPEnum2WndItem pitem, LPVOID *pvalue)
{
  LPENUMBSTR  pen = (LPENUMBSTR)itf;
  HRESULT     hr  = pen->lpVtbl->Next(pen, 1, &pitem->bstr, NULL);

  if(hr == S_OK)
  {
    *pvalue = pitem->bstr;
  }

  return hr;
}

void EnumBSTR_ClearItem(LPVOID pen, LPEnum2WndItem pitem, LPVOID value)
{
  SysFreeString(pitem->bstr);
}

/* IEnumVARIANT */

HRESULT EnumVARIANT_NextItem(LPVOID itf, LPEnum2WndItem pitem, LPVOID *pvalue)
{
  LPENUMVARIANT pen = (LPENUMVARIANT)itf;
  HRESULT       hr  = pen->lpVtbl->Next(pen, 1, &pitem->var, NULL);

  if(hr == S_OK)
  {
    hr = VariantChangeType(&pitem->var, &pitem->var, VARIANT_ALPHABOOL,
VT_BSTR);

    if(SUCCEEDED(hr))
    {
      *pvalue = pitem->var.bstrVal;
    }
    else
    {
      VariantClear(&pitem->var);
    }
  }

  return hr;
}

void EnumVARIANT_ClearItem(LPVOID pen, LPEnum2WndItem pitem, LPVOID value)
{
  VariantClear(&pitem->var);
}

/* IEnumGUID */

HRESULT EnumGUID_NextItem(LPVOID itf, LPEnum2WndItem pitem, LPVOID *pvalue)
{
  LPENUMGUID  pen = (LPENUMGUID)itf;
  HRESULT     hr  = pen->lpVtbl->Next(pen, 1, &pitem->guid, NULL);

  if(hr == S_OK)
  {
    hr = StringFromCLSID(&pitem->guid, SyCastRaw(LPOLESTR*, pvalue));
  }

  return hr;
}

void EnumGUID_ClearItem(LPVOID pen, LPEnum2WndItem pitem, LPVOID value)
{
  OleString_Destroy(SyCastRaw(LPOLESTR*, &value));
}

/* IEnumUnknown */

HRESULT EnumUnknown_NextItem(LPVOID itf, LPEnum2WndItem pitem, LPVOID *pvalue)
{
  LPENUMUNKNOWN pen = (LPENUMUNKNOWN)itf;
  HRESULT       hr  = pen->lpVtbl->Next(pen, 1, &pitem->punk, NULL);

  if(hr == S_OK)
  {
    /* Use the VariantChangeType() mechanism to get the value property of
     * the object.
     */
    VARIANT dest;

    VariantInit(&dest);

    hr = Dispatch_GetProperty(pitem->punk, DISPID_VALUE, &dest);

    if(SUCCEEDED(hr))
    {
      hr = VariantChangeType(&dest, &dest, VARIANT_ALPHABOOL, VT_BSTR);

      if(SUCCEEDED(hr))
      {
        *pvalue = OleString_Dup(dest.bstrVal);
      }

      VariantClear(&dest);
    }

    if(NULL != pitem->punk)
    {
      pitem->punk->lpVtbl->Release(pitem->punk);
    }
  }

  return hr;
}

void EnumUnknown_ClearItem(LPVOID pen, LPEnum2WndItem pitem, LPVOID value)
{
  OleString_Destroy(SyCastRaw(LPOLESTR*, &value));
}
```

DISPID_VALUE

CONVERTING OLE strings, BSTRs, VARIANTs, and even GUIDs to strings that may then be inserted into windows controls seems conceptually pretty straightforward. However, doing so with COM objects (via IUnknown) doesn't seem anywhere near as obvious. As you can see from Listing 4, the value is obtained from a function Dispatch_GetProperty(). This function lives in a proprietary Synesis library, so I've included the definition here for anyone who wishes to include IEnumUnknown handling.

```
SYFNCOMDECL Dispatch_GetProperty( LPUNKNOWN    punk,
                                  DISPID       dispid,
                                  LPVARIANT    pvarResult)
{
  HRESULT     hr;

  if(NULL == punk ||
     NULL == pvarResult)
  {
    hr = E_POINTER;
  }
  else
  {
    LPDISPATCH  pdisp;

    hr = punk->QueryInterface(IID_IDispatch, (void**)&pdisp);

    if(SUCCEEDED(hr))
    {
      UINT        iArgErr = (UINT)~0;
      DISPPARAMS  params  =
      {
        NULL, NULL, 0, 0
      };

      hr = pdisp->Invoke(dispid, IID_NULL,
                         LOCALE_SYSTEM_DEFAULT,
                         DISPATCH_PROPERTYGET,
                         &params, pvarResult,
                         NULL, &iArgErr);

      pdisp->Release();
    }
  }

  return hr;
}
```

The function calls IDispatch::Invoke() with DISPATCH_PROPERTYGET for the given dispid, and returns the automation return value as the property's value. The significant thing is that the DISPID used—DISPID_VALUE (0)—is a predefined one whose meaning is as the "value" of the object on which the call is made. For example, an Excel spreadsheet cell object would return its contents for its value property.

The other thing to note from Listing 4 is the use of VARIANT_ALPHABOOL in the call to VariantChangeType() call within the handlers for IEnumUnknown and IEnumVARIANT. This simply makes the conversion from Boolean to string result in "True" or "False" rather than "–1" or "0", as can be seen in Figure 1.

—M.W.

Listing 17 ConvertRecordsetToXml

```
' These two functions show how to create the XML document representing multiple
' recordsets using VBA code. Note that this code will work in VB.NET with lit-
tle
' modification.
Public Function ConvertRecordsetToXml(ByVal voRecordset As ADODB.Recordset _
) As String
    Dim loStream As New ADODB.Stream

    loStream.Open
    voRecordset.Save loStream, adPersistXML
    loStream.Position = 0

    ConvertRecordsetToXml = loStream.ReadText()
End Function

Public Function ConvertRecordsetsCollectionToXmlDocument( _
    ByVal voRecordsets As Collection _
) As MSXML.DOMDocument
    Dim loXmlDoc As New MSXML.DOMDocument
    Dim loXmlResults As MSXML.IXMLDOMElement
    Dim loRecordset As ADODB.Recordset

    Set loXmlResults = loXmlDoc.createElement("results")
    loXmlDoc.appendChild loXmlResults

    For Each loRecordset In voRecordsets
        Dim loXmlRecordsetDoc As New MSXML.DOMDocument
        loXmlRecordsetDoc.loadXML ConvertRecordsetToXml(loRecordset)
        loXmlResults.appendChild loXmlRecordsetDoc.documentElement
    Next loRecordset

    Set ConvertRecordsetsCollectionToXmlDocument = loXmlDoc
End Function
```

Listing 18 Sending the order

```
' Send orders to the server for processing.
Private Sub CommandButton3_Click()
    Dim loRecordsets As Collection
    Dim loXmlRequestDocument As MSXML.DOMDocument
    Dim loXmlResponseDocument As MSXML.DOMDocument
    Dim lsUrl As String

    Set loRecordsets = CreateOrderRecordsets()

    lsUrl = "http://localhost/LiteWebServicesCS/PlaceOrder.asmx"

    'get the order
    Set loXmlRequestDocument = ConvertRecordsetsCollectionToXmlDocument( _
        loRecordsets)

    'upload the order
    Set loXmlResponseDocument = _
        InvokeLiteWebService(lsUrl, loXmlRequestDocument, "POST")

    'do something with the response...
    '...
End Sub
```

Table of Contents