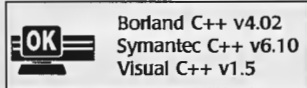


A Filter for WordPerfect Graphics Files

Michael Bertrand



A graphics filter is code that enables a program to read images created by other programs in foreign formats, converting them to whatever format the program uses internally. If the program defines its graphics filters as DLLs with a specific function interface, then new graphics filters can be added without altering the original program. In this article I explain the basic principles of writing an interactive filter/viewer for vector graphics images in Windows, using WordPerfect's WPG format for illustration. The application discussed here, SHOWWPG, can read WPG v5.x files from disk, scale them interactively as is done in WordPerfect, and copy them to the clipboard as Windows metafiles. In other words: the incoming foreign format is WPG, and the internal native format is Windows metafile; SHOWWPG converts WPG files to Windows metafiles. Since a Windows metafile is a static list of GDI calls, there is little distinction between rendering the image and translating it to Windows metafile format; the same code is used to do both. SHOWWPG is too long to be printed in its entirety, so the code listings (funcs.c – Listing 1) focus on a few pivotal issues; the entire source code is on this issue's code disk (see the table of contents for availability information).

WordPerfect graphics files consist of vector graphics. In vector graphics, images consist of a series of primitives, such as lines, polygons, ellipses, and so on. Each primitive record in the vector graphics file contains coordinates defining the primitive; a line record, for example, would contain $(x1, y1)$ values for the first endpoint and $(x2, y2)$ values for the second endpoint. Drawing and CAD programs typically implement vector formats because of the natural scaling such formats afford (doubling all coordinates doubles the size of the image). This contrasts to bitmapped or raster graphics, where color codes for each pixel in a rectangular region define an image. Bitmap formats include PCX, BMP, and TIF, natural formats for paint programs and for scanners.

Mike Bertrand teaches Mathematics and programming at Madison Area Technical College, Madison, WI 53704.

Converting Records

A vector graphics filter traverses the incoming file record by record, converting each record to its equivalent in the native format. SHOWWPG's traverser function is *PlayWPGFile()*, which goes through the WPG file in memory, calling function *DrawWPGRecord()* to convert each record to equivalent Windows GDI calls. *DrawWPGRecord()* switches on the type of WPG record, calling utility functions to convert records (*DrawLine()*, *DrawPoly()*, *DrawEllipse()*, and so

on). Windows' device independence comes to the rescue here — if you pass *DrawWPGRecord()* a screen device context, then it renders the WPG image on the screen, while if you pass it a metafile device context, it creates a metafile. Windows makes little distinction between a vector graphics viewer and filter.

Since different formats have different capabilities, record matching is an imperfect business. Simple records like lines and polylines (a series of connected line segments) convert directly from WPG format to equivalent

GDI calls. Ellipse records present difficulties, since WPG ellipses can include elliptical arcs, pie slices, chords, and rotated ellipses. The first three can be matched to Windows *Arc()*, *Pie()*, and *Chord()* calls, but since Windows doesn't provide for rotated ellipses, you would have to write your own algorithm to implement this feature.

SHOWWPG takes some pains to implement WPG's "fill attribute" record. A fill attribute is a color and pattern setting used to fill the interior of all subsequent polygons in the WPG file until the next fill attribute record is encountered. This is the same concept as Windows' brushes, used to fill polygonal areas. SHOWWPG calls *SetBrush()* every time a fill attribute record is encountered. *SetBrush()* creates and selects a Windows brush matching the WPG fill attribute record, making sure to delete the previous brush to prevent all these brushes from adding permanently to Windows' global heap. If the fill attribute record includes a pattern, SHOWWPG creates a Windows bitmap matching the pattern, then calls Windows' *CreateDIBPatternBrush()* to create a brush based on the bitmap. Fortunately, Windows provides for 8-pixel-square pattern brushes, the largest fill pattern encountered in WPG fill attribute records.

WPG's "line attribute" matches the Windows concept of "pen", specifying color, style, and width settings whenever lines are drawn. This record is difficult to implement, since Windows' pen support is minimal. SHOWWPG implements WPG line-attribute color settings by creating and selecting Windows pens, much as with fill attributes and brushes. Since Windows pens have few styles available, and those only at one pixel

Help Magician Pro 3.0

The Professional Help Authoring Tool That Everyone Can Use!

Beginners Love It Because It's So Easy.

With Help Magician Pro 3.0, you can develop online help and documents seamlessly in a true WYSIWYG environment much like WinHelp. If you have manuals or documents, Help Magician Pro will import them from any popular Windows word processor and convert them to online help. You can simultaneously test your help files while working on them.

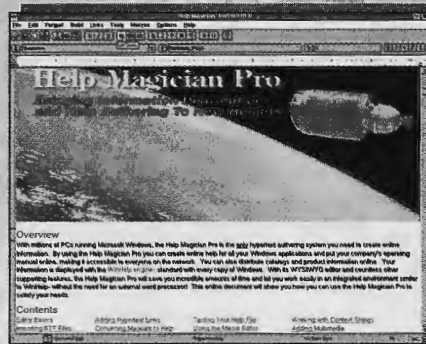
Experts Love It Because It's So Powerful.

Help Magician Pro has all the flexibility you've come to expect from a professional help authoring tool like support for ALL WinHelp 3.1 features, a fool-proof macro editor, multi-file project management, multimedia support, automatic glossary creation, and much more.

Order your copy of the Help Magician Pro 3.0 today. Only \$249! 30-day satisfaction money-back guarantee. Corporate site and network licensing available.



says "Designing help systems is so easy that you might even want to use it as a hypertext authoring tool. Certainly for Windows programmers it's indispensable."



Actual Editing Environment

Here's The Scoop.

The Help Magician is a proven product with years of customer support and satisfaction. We can demonstrate that the Help Magician Pro has more features and is faster and easier to use than any other help authoring tool. Drop us a line at 1-800-542-2742 to get more details and a FREE fully functional demo disk. The Help Magician Pro now includes the Help Compiler and SHED editor.



Software
Interphase
Incorporated

82 Cucumber Hill Rd, #213
Foster, RI 02825
Voice: (401) 397-2340
Fax: (401) 397-6814

□ Request Reader Service #287 □

Figure 1 Mapping WordPerfect graphics and Windows screen coordinates

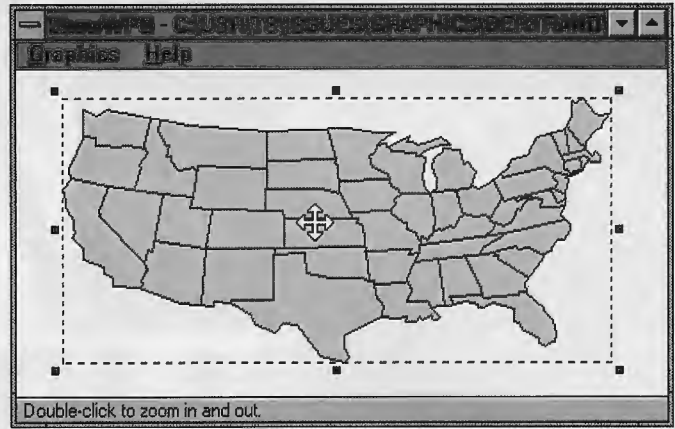
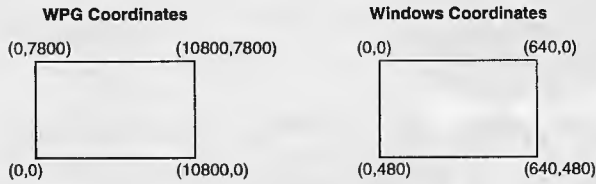


Figure 2 SHOWWPG displaying a WPG file

thickness, supporting fill attribute style and width settings would require an appeal to Windows' `LineDDA()` call, enabling Windows' line-drawing with any attributes.

SHOWWPG doesn't attempt to implement text or embedded bitmaps, both possible in WPG files. The traverser, `PlayWPGFile()`, ignores and skips over unimplemented records. Implementing a new record type requires adding the record as a case constant in `PlayWPGFile()`'s switch, then calling and writing the function to support the new record (that is, translating it into GDI calls).

Scaling

Converting one vector format to another always involves scaling of coordinates, a core idea. Both WordPerfect graphics and Windows have the concept of a polyline, a series of straight line segments connected end to end. But Windows' `Polyline()` cannot directly render the image from the WPG file because the coordinate systems are different. Coordinates in WPG files are based on WPG space, which is 10,800 units wide by 7800 units high with the origin, or (0,0) point, in the lower left. This differs

Make a lasting first impression with EDI Install Pro

EDI Install Pro is a powerful, full featured installation toolkit designed to make your work as effortless as possible.

Standard, professional interface.

Your customers will feel right at home with EDI Install Pro's standard, professional interface. We don't clutter our windows with useless gadgets or hokey graphics. In our opinion, a clean, standard interface, makes for a better product and leaves a lasting impression.



No script language to learn.

Don't waste your time learning yet another script language. Our simple information file makes creating powerful installations a breeze. In fact, using the INF Maker utility you can complete even complex installations in less than an hour!

Includes a complete uninstaller.

Included with EDI Install Pro is an incredible utility that allows your users to remove your applications from their system in one easy step! Our uninstaller removes or changes .INI files; deletes application files,

Program Manager groups and optionally user data files.

No hidden costs!

Unlike some of our competitors, we don't charge royalties, and we don't require that you purchase a license for each product you distribute. Ask our competitors about their licenses - you'll be surprised.

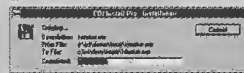


You'll be in good company.

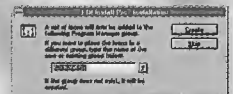
What do AT&T, Banyan Systems, Bell Canada, BP Oil, Cirrus Logic, Fannie Mae, Eastman Kodak, NCR Canada, Pacific Bell, Polaroid Corp., SunSelect, TRW, Xerox, Ziff-Davis Publishing, and the US Army Corps of Engineers have in common? They all bought EDI Install Pro. Shouldn't you?

Some EDI Install Pro Features:

Standard, professional interface; 3D optional • Dithered, tiled or bitmap background • Bulletin bitmaps (billboards) • Progress dialog keeps users informed • Selectable components for custom installs • No programming required • DLL expandable • File compression & splitting • Version resource checking • Disk branding with user



name • Auto font install • Create & modify .INI files • Creates Program Manager groups & icons • Built-in readme viewer • Small size (~80Kb) • Support for floppy, hard disk, CD-ROM, Network, and e-mail distribution • Ask about our new OEM package!



All this plus much, much more!

Order Now For Only \$179.95!

See our evaluation version on the BBS, or on CompuServe's "WINSK" forum, file "INSPRO.EXE".



24-2979 Panorama Drive
Coquitlam, BC V3E 2W8
Canada

Telephone/Fax: (604) 945-3198
Eschalon BBS: (604) 945-7602
CompuServe: 76625,1320

VISA cards, US/Canadian checks and bank drafts accepted (order forms must accompany all draft orders). Sorry, no POs accepted. Canadian residents add 7% GST. BC residents add 7% PST. Add \$10 Shipping & handling (\$15 overseas), \$20 for Federal Express (\$45 overseas). All Prices are in US currency.

For European and multilingual orders, please contact: Windowshare S.A.R.L. (France), voice at (+33) 87-30-85-57, fax at (+33) 87-32-37-75, or CompuServe at 100031,3257.

EDI Install Pro, EDI Uninstall, and the Eschalon Development Inc. logo are trademarks of Eschalon Development Inc. Other names are (registered) trademarks of their respective companies. All prices and specifications subject to change without notice.

markedly from standard screen resolutions of 640x480 or 800x600, with the origin at the upper left (see Figure 1).

In converting from WPG space to a screen space of 640x480, for example, I apply a scaling transformation to a point (x,y) in WPG space to calculate its corresponding point (x',y') on the screen. Wherever (x,y) is located in WPG space, its corresponding point (x',y') is in the same relative position on the screen. A scaling transformation is denoted by:

$$(x,y) \rightarrow (x',y')$$

where x' and y' are separate x and y transformations:

$$x' = \text{scaleX} * x + \text{offsX}$$

$$y' = \text{scaleY} * y + \text{offsY}$$

You can calculate what scale factors and offsets to use by picking two WPG coordinates and deciding which Windows coordinates you want to map them to. For example,

Listing 1 *funcs.c* — Core functions for WPG graphics filter

```
long WINAPI _export WndProc(HWND hWnd, UINT message,
                          UINT wParam, LONG lParam)
/*
USE: SHOWWPG's main window procedure.
IN: Standard WndProc() parameters.
WM_CREATE : Set normal caption
WM_ACTIVATE: Set minimized caption
WM_SIZE : Update global ClientRect
WM_COMMAND : IDM_RETRIEVE triggers GetOpenFileName()
              IDM_COPY copies to clipboard
              IDM_ABOUT shows About DlgBox
WM_PAINT : Traverse and display WPG file
WM_MOUSEMOVE : Pass along to Modify()
WM_LBUTTONDOWN: to size or translate
WM_LBUTTONUP : "
WM_LBUTTONDOWNBLCLK: Pass along to Modify() to zoom in
WM_RBUTTONDOWNBLCLK: Pass along to Modify() to zoom out
WM_DESTROY : Finalize.
```

hBuf is a key global variable that is non-NULL if and only if it is a valid memory handle to a WPG file. Check hBuf to see if you have a file to show.

```
*/
{
HANDLE hNewMem; /* temp handle to file mem */
PAINTSTRUCT ps; /* needed in WM_PAINT */
HDC hdc; /* needed in WM_PAINT */
LPSTR lpIs; /* ptr to last '\ ' in string */
static FARPROC lpDlg; /* for About box function */

switch (message)
{
case WM_CREATE:
/* Write default caption for main window. */
wsprintf(Caption, "%s - (Untitled)", AppName);
SetWindowText(hWnd, Caption);
/* Default iconic name is just app name. */
lstrcpy(IconicName, AppName);
break; /* WM_CREATE */

case WM_ACTIVATE:
/* When minimized, use AppName only for caption,
* else full caption.
*/
SetWindowText(hWnd, IsIconic(hWnd) ? IconicName :
Caption);
break; /* WM_ACTIVATE */

case WM_SIZE:
/* Update global ClientRect. */
GetClientRect(hWnd, &ClientRect);
break; /* WM_SIZE */

case WM_COMMAND:
switch (wParam)
{
case IDM_RETRIEVE:
/* Get file name, read into memory. */
if ((hNewMem = LoadWPGFile(hWnd)) == NULL)
```

```
break;
/* If hBuf valid, free it and reset. */
if (hBuf) GlobalFree(hBuf);
hBuf = hNewMem;
break; /* IDM_RETRIEVE */

case IDM_COPY:
/* hBuf non-NULL means file is in memory. */
if (hBuf) CopyWMFToClipboard(hWnd);
break; /* IDM_COPY */

case IDM_ABOUT:
lpDlg=MakeProcInstance((FARPROC)AboutDlgProc,
hInst);
DialogBox(hInst, "AboutBox", hWnd, lpDlg);
FreeProcInstance(lpDlg);
break; /* IDM_ABOUT */
}
}
```

☑ TWAIN support for your application ☐ ☒ ☓

- Do you want to support scanners with your Windows application?
- Do you think it makes sense to support the industry standard TWAIN?
- Do you want to save your resources?
- Do you want to save money and time?

The **TWAIN Integration Kit™** is the best choice to implement TWAIN in MS-Windows applications. Forget about the estimated 20 'man days', for the development of a TWAIN implementation. With the TIK you have TWAIN in your application within a day or less. For the value of only a few 'man days', you will get the powerful and easy to use TIK DLL. The Price is only \$1995 and no runtime licenses are payable. Order now or test the trial version first with a subset of commands for only \$250. We anxiously await your order or inquiry for further details on the TIK.

30-DAY MONEY-BACK GUARANTEE

JUNGLAUS SOFTWARE ENGINEERING P.O. Box 270 202
D- 40525 Duesseldorf Germany
FAX +49 211 562 31 12 CompuServe 100334.2207



Free demo available via CompuServe. Download TIKDEMO from WINSDK forum, section Public Utilities.

☐ Request Reader Service #222 ☐

Listing 1 *continued*

```

        break; /* WM_COMMAND */

case WM_PAINT:
    hDC = BeginPaint(hWnd, &ps);
    if (hBuf) PaintWindow(hDC);
    EndPaint(hWnd, &ps);
    break; /* WM_PAINT */

case WM_MOUSEMOVE:
case WM_LBUTTONDOWN:
case WM_LBUTTONUP:
case WM_LBUTTONDOWNBLCLK:
case WM_RBUTTONDOWNBLCLK:
    if (hBuf)
        Modify(hWnd, message, lParam);
    else
        /* SetCursor() every MOUSEMOVE if no file. */
        SetCursor(HCursor[NO_HIT]);
    break; /* WM_MOUSEMOVE... */

case WM_DESTROY:
    /* Free memory handle if valid. */
    if (hBuf) GlobalFree(hBuf);
    if ((lpIs = _fstrchr(fileName, '\\')) != NULL)
    {
        /* Excise file name so only path left. */
        *lpIs = 0;
        /* Write current path to SHOWWPG.INI. */
        WritePrivateProfileString("Defaults", "Path",
            fileName, "SHOWWPG.INI");
    }
    PostQuitMessage(0);
    break;

default: /* Passes it on if unprocessed */
    return DefWindowProc(hWnd, message, wParam, lParam);
} /* switch */

return (NULL);
}

void LOCAL Modify(HWND hWnd, UINT message, LONG lParam)
/*
USE: Sizing and translating tool.
IN: WndProc() parameters passed along.
NOTE: State tool with route WAITING-->START_MODIFYING
-->MODIFYING-->WAITING. START_MODIFYING is the state
for the first WM_MOUSEMOVE after tool has started.
There are 9 modifications: translation + 8 types of
resizing. Which modification is triggered depends on
which hot spot the cursor was in at WM_LBUTTONDOWN.
Each modification has its own section in
RestrictCursor(), DrawDraggingRect(), UpdateScale(),
functions called respectively at WM_LBUTTONDOWN,
WM_MOUSEMOVE, and WM_LBUTTONUP (RestrictCusor()
initializes, DrawDraggingRect() draws interactively,
UpdateScale() finalizes).
*/
{
    HDC hDC; /* DC to draw on */
    static POINT pt; /* button-down point */
    static HPEN hPen; /* dashed pen */
    static HRGN hRgn; /* clip region */
    static int hs; /* hot spot index */
    static int state; /* system state */

    switch (message)
    {
        case WM_LBUTTONDOWN:
            if (state == WAITING)
                if ((hs = PtInHotSpot(&TargetRect,
                    MAKEPOINT(lParam))) != NO_HIT)
                {
                    state = START_MODIFYING;
                    hPen = CreatePen(PS_DOT, 1, 0L);
                    hRgn = CreateRectRgn(0, 0, ClientRect.right,
                        ClientRect.bottom-(ILINE_HT+2));
                    pt = MAKEPOINT(lParam);
                    RestrictCursor(hs, hWnd, &pt);
                }
            break; /* WM_LBUTTONDOWN */

        case WM_MOUSEMOVE:
            switch (state)
            {
                case WAITING:
                    /* Set Cursor for hot spot or Arrow. */
                    SetCursor(HCursor[PtinHotSpot(&TargetRect,
                        MAKEPOINT(lParam))]);
                    break;
                case START_MODIFYING:
                    state = MODIFYING;
                    hDC = GetDC(hWnd);
                    SelectObject(hDC, hRgn);
                    /* Erase Framed rect (XOR mode). */
                    SetROP2(hDC, R2_NOTXORPEN);
                    FrameOurRect(hDC, &TargetRect);
                    /* Draw rect with SOLID pen first time. */
                    DrawDraggingRect(hs, hDC, pt);
                    pt = MAKEPOINT(lParam);
                    /* Switch to DASHED pen for next draw rect.*/
                    SelectObject(hDC, hPen);
                    DrawDraggingRect(hs, hDC, pt);
                    ReleaseDC(hWnd, hDC);
                    break;
                case MODIFYING:
                    hDC = GetDC(hWnd);
                    /* Draw in XOR mode with DASHED pen. */
                    SetROP2(hDC, R2_NOTXORPEN);
                    SelectObject(hDC, hPen);
                    SelectObject(hDC, hRgn);
                    /* Erase last rect. */
                    DrawDraggingRect(hs, hDC, pt);
                    pt = MAKEPOINT(lParam);
                    /* Draw new rect. */
                    DrawDraggingRect(hs, hDC, pt);
                    ReleaseDC(hWnd, hDC);
                    break;
            } /* switch (state) */
            break; /* WM_MOUSEMOVE */

        case WM_LBUTTONUP:
            if (state==MODIFYING || state==START_MODIFYING)
            {
                DeleteObject(hPen);
                DeleteObject(hRgn);
                UpdateScale(hs, pt);
                /* Set Cursor for hot spot or Arrow. */
                SetCursor(HCursor[PtinHotSpot(&TargetRect,
                    MAKEPOINT(lParam))]);
                /* RePaint only if modification started. */
                if (state == MODIFYING)
                    InvalidateRect(hWnd, NULL, TRUE);
                state = WAITING;
            }
            break; /* WM_LBUTTONUP */

        case WM_LBUTTONDOWNBLCLK:
            if (state == WAITING &&
                PtInRect(&TargetRect, MAKEPOINT(lParam)))
            {
                UpdateScale(ZOOM_IN, pt);
                InvalidateRect(hWnd, NULL, TRUE);
            }
            break; /* WM_LBUTTONDOWNBLCLK */

        case WM_RBUTTONDOWNBLCLK:
            if (state == WAITING &&
                PtInRect(&TargetRect, MAKEPOINT(lParam)))
            {

```

given a 640x480 screen, you might decide to map WPG coordinate (0,0) to Windows coordinate (0,480) and WPG coordinate (10800,7800) to Windows coordinate (640,0), as shown in Figure 1. Two equations arise from associating each x with its corresponding x' :

$$0 = \text{scaleX} * 0 + \text{offsX}$$

$$640 = \text{scaleX} * 10800 + \text{offsX}$$

Solving these equations results in $\text{offsX} = 0$ and $\text{scaleX} = 640/10800 = 0.059$. After similar calculations in y , the specific scale transformation can be written down:

$$x' = 0.059 * x$$

$$y' = -0.061 * y + 480$$

The scale factors 0.059 and -0.061 indicate that the original coordinates are shrunk to about 6% of their original values in order to fit on the screen; the minus sign reverses the sense of y due to the different placement of the origin. An offset of 480 must then be added to the y coordinate to get the image on the screen. SHOWWPG's function *CalcScaleParms()* calculates the scale parameters based on the current target rectangle, much as in this example, except that any target rectangle may be used. The term "scale parameters" includes scale factor and offset for both x and y .

A straight line between $(x1,y1)$ and $(x2,y2)$ in WPG space is rendered by first scaling the two points and then drawing the line between the scaled points with Windows' *MoveTo()* and *LineTo()*. For a polyline, first scale all the coordinates and then pass the scaled points to Windows'

Listing 1 *continued*

```

    UpdateScale(ZOOM_OUT, pt);
    InvalidateRect(hWnd, NULL, TRUE);
}
break; /* WM_RBUTTONDOWNCLK */
} /* switch (message) */
}

BYTE LOCAL DrawWPGRecord(HDC hDC, LPBYTE FAR *p,
                        SCALE sc)
/*
USE: Decipher and draw WPG record.
IN: hDC = handle to DC in which to draw
    *p = ptr to start of WPG record
    sc = SCALE struct to use in drawing record
OUT: *p = ptr to start of next record
RET: Return record type.
NOTE: p is a ptr to a ptr to the start of the WPG
      record. The ptr scans thru the record and ends by
      pointing to the next record. The actual ptr, main-
      tained by the caller, is updated as well (this is
      why we need a ptr to a ptr). Note that:

        p = ptr to ptr
        *p = ptr
        **p = data
*/
{
    BYTE recType; /* WPG record type */
    DWORD recLen; /* WPG record length, as DWORD */

    /* Get RecordType, advance ptr. */
    recType = *(*p)++;

    recLen = GetRecordLength(p);

    /* Process record. Each type of record being handled
    * appears as a case constant; records not listed are
    * ignored and jumped over.
    */
    switch(recType)
    {
    case WPG_FILLATTR:
        SetBrush(hDC, *p, SET_MID);
        break;
    case WPG_LINEATTR:
        SetPen(hDC, *p, SET_MID);
        break;
    case WPG_LINE:
        DrawLine(hDC, *p, sc);
        break;
    case WPG_POLYLINE:
    case WPG_POLYGON:

```

```

        DrawPoly(hDC, *p, sc, recType);
        break;
    case WPG_ELLIPSE:
        DrawEllipse(hDC, *p, sc);
        break;
    } /* switch */

    /* Advance ptr to next record for caller. */
    *p += recLen;

    return(recType);
}

```

Installigence™

Make The Best First Impression!

Spend your time finishing your product, not its installation, with the best Windows installation builder on the market. You'll create your install in minutes with its object oriented installation factory™. Define install operations and order with property inspectors, not some bad language.

It's the only install that automates file compression and disk layout with a drag and drop interface that leaves its competitors far behind. You get all the features you want and it only uses ~80KB of disk space! It's yours for a limited time at a special price, because you really shouldn't be using an install that isn't as good as your product.

- ▼ Buy before 10-1-94 for a FREE Chicago Upgrade!
- ▼ Supports multiple file groups, partial and un-install.
- ▼ File compression, splitting, and version checking.
- ▼ System file, .INI, CONFIG.SYS and registry support.
- ▼ Progman groups, readme viewer and billboards.
- ▼ Gradient, bitmap or custom background window.
- ▼ International support and many more features.
- ▼ Royalty-free distribution.
- ▼ Mastercard and Visa accepted.

\$99.00 In September
Order Now 1-800-494-0550

22845 NE 8th Street Suite 314
 Redmond, WA 98053-7299
 Phone 206-836-0111
 FAX: 206-868-0550

*Instance
 Corporation*

□ Request Reader Service #313 □

Polyline(). This is the magical core of vector graphics – the scaled image looks the same as the original!

The Target Rectangle

In general, the target for display is not the entire screen, but a rectangle smaller than the screen, resizable by the user. This target rectangle is a key global variable, called *TargetRect*, in *SHOWWPG*. The user resizes the target rectangle by dragging little black boxes at the corners and sides of the rectangle, as is done in *WordPerfect*, or by double-clicking for zooms (left double-click to zoom in, right double-click to zoom out). After the user resizes the target rectangle, the image within the rectangle is scaled, with the scaling transformation mapping WPG space into the target rectangle. The scaling transformation is developed as in the example above, except that the WPG file maps to the target rectangle only, rather than to the entire screen. After the scaling transformation is applied, *PlayWPGFile()* renders the scaled image. The current scale transformation, depending on *TargetRect*, is stored in another key global structure, *Scale*, which holds the scale factors and offsets.

The function *InitScaleParms()* calculates an image's initial target rectangle when the file is loaded. The image's WPG space size, together with reasonable margins, is used

to calculate a target rectangle fitting in the middle of the client area. From that point, the user adjusts the target rectangle interactively, with *SHOWWPG* updating scale parameters at each adjustment, until a new file is loaded and *InitScaleParms()* starts the cycle anew.

Reading a WPG File into Memory

SHOWWPG provides a common dialog box that lets the user choose WPG files to render. The program calls *GlobalAlloc()* for a memory block sufficient to hold the entire file:

```
hMem = GlobalAlloc(fileSize);
```

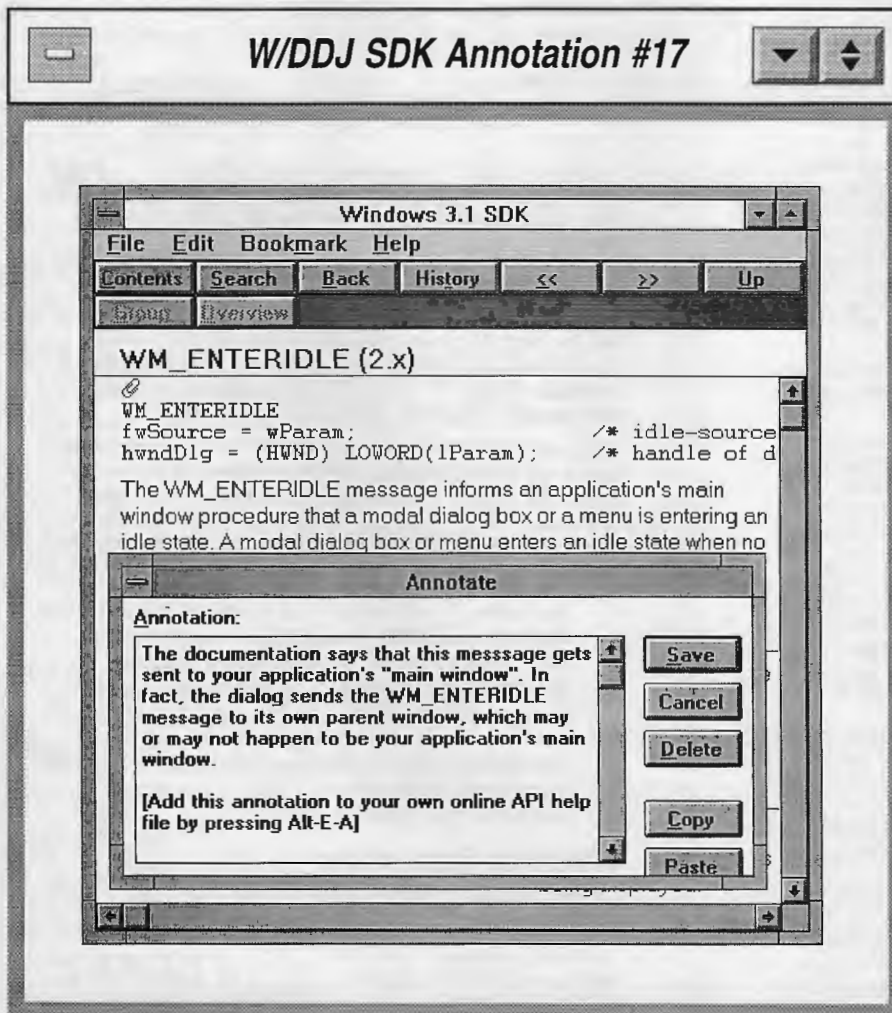
The memory block remains valid, continuing to hold the file, until another file is chosen. The file data is not changed, but serves as the image archive for rendering. Scaled data (for indeterminately sized polylines, for example) is stored in supplementary memory blocks. Iterated scaling (scaling already scaled data) can corrupt an image, so I always want to be one step away from the original.

Choosing a new image displaces any existing image, so the code frees the current *hMem* (with *GlobalFree()*) before generating a new *hMem* for the new image. Failing to free unused memory handles is a dire error in Windows, since unfreed blocks continue to occupy the global memory heap, reducing the amount of memory available to all applications for the rest of the Windows session.

Interactive Aspects

After a WPG file has been loaded into memory, the user can resize or translate the target rectangle by dragging the little black box knobs on its corners and edges or by double-clicking to zoom. Figure 2 shows how the program displays a WPG file. *SHOWWPG* takes note when the cursor is over one of the knobs, adjusting the cursor shape accordingly. The upper right knob, for example, is linked to the 'northeast – southwest' arrow cursor, the same one Windows itself uses to signal the availability of resizing when a cursor passes over the upper right corner of a window. The program responds to a left button down in one of the knobs by initiating a drag. The eight knobs are called hot spots for this reason. The target rectangle itself is the ninth hot spot; it is linked to the arrow cross and to translation when dragging ensues. The nine hot spots are determined by the target rectangle, with function *PtInHotSpot()* returning the index of the hot spot a point is in, if any.

The drags are managed by function *Modify()*, a tool sensitive to the



stream of mouse events and their meaning. Pressing the left mouse button down on the knob at the top left of *TargetRect*, for example, begins the top left drag, where the top left corner is moved but the opposite right bottom corner is anchored at its original location. The corner drags affect both scale factors since both dimensions of *TargetRect* are changed. The top-middle drag, initiated by pressing the left button down on the top middle knob, causes only the top edge of *TargetRect* to move. In this case, the x scale factor remains unchanged, while the y scale factor will change depending on the top edge's position at the end of the drag. Translating is also available by dragging *TargetRect* itself; in this case only the offsets and not the scale factors are affected. There are nine different kinds of drag, each one a different modification. Double-clicking triggers automatic zooms without any dragging. When a drag or zoom is done, both the target rectangle and scaling transformation must be updated.

The drags have much in common. All are initiated by the user's pressing the left mouse button down in a hot spot; all are in progress as the user moves the mouse, or drags; and all need to update the global structures *TargetRect* and *Scale* when the drag is done. The common functionality calls for a single function to manage all nine drags — function *Modify()*. This way only one block of code would need to be rewritten to change the way drags work — perhaps implementing the right mouse button or shift-drag (dragging with the shift key down).

Listing 1 *continued*

```

}

void LOCAL InitScaleParms(HANDLE hMem, LPSCALE lps,
                        LPRECT lpr)
/*
  USE: Initialize target rect and scale parameters.
  IN:  hMem = handle to memory containing WPG data.
  OUT: lps = ptr to scale parameters structure
      lpr = ptr to target rectangle : region on the
          screen into which image is mapped.
  NOTE: Uses global ClientRect, sets global WPGsp.
*/
{
  LPBYTE lpWPG; /* ptr to WPG data */
  double scaleX; /* X scale factor */
  double scaleY; /* Y scale factor */
  int top; /* top of target rect if loose fit - y */
  int left; /* left of target rect if loose fit - x */

  /* Generate ptr to WPG data. */
  lpWPG = (LPBYTE) GlobalLock(hMem);

  /* If find WPG_START1 as first record, use for WPG
   * space; else use defaults set at initialization.
   */
  if (*(lpWPG + WPG_OFFS_FIRSTREC) == WPG_START1)
    WPGsp = *((LPSIZE)(lpWPG + WPG_OFFS_WPGSIZE));

  /* Unlock memory handle. */
  GlobalUnlock(hMem);

  /* Calc scale factors in both dimensions. */
  scaleX = (double) ClientRect.right / WPGsp.cx;

```

**Locked into old
PASCAL or BASIC?
Wish Your Software Was In
C?**

Then Don't Re-Invent the Wheel!

**Automatically translate your code into readable
and maintainable C with
PASCAL and BASIC to C Translators**

Available for most popular variants

eg. Turbo Pascal, VAX Pascal/Basic, Microsoft Pascal/Basic

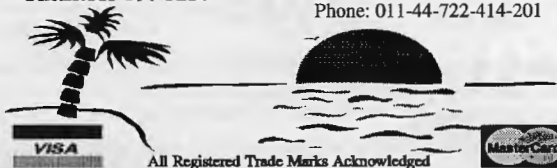
For more information call now!

Technosoft (US)

PO Box 8210
Rockford, IL 61126-8210
Phone: 815-397-3214

Technosoft (Europe)

Enterprise House
Cherry Orchard Lane
Salisbury, SP2 7LD
Phone: 011-44-722-414-201



□ Request Reader Service #253 □

NEW RELEASE v2.50 **ToolsKan** *More features*

For SDK & Visual Basic

3D Chart

- Over 30 of 2D & 3D chart styles
- Rotation & scrolling
- Supports printing & clipboard

Toolbox

- Creates buttons from bitmaps or text
- Supports scrolling
- 3D buttons w/ color customization
- Single/multiple/no-state button groups

Ribbon

- 3D items with color customization
- Supports combobox, text, & buttons

Field Validation

- Validates date, time, number fields & "PIC" statements

Meter

- Creates vertical, horizontal, & circular gauges with choice of needle or color bar as indicators
- Linear & logarithmic scales

Table

- Column & row split windows
- Multiple row & column selections
- Check boxes/radio buttons/bitmaps/editable/combobox column
- Input validation
- Color customization

Status Bar

- Auto scrolled text
- Stretchable field width
- Colored progress bar
- Show date, time, & key states

Free Demo from BBS.

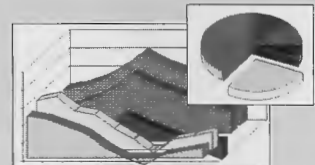
No Royalties. 30-day MBG.
Optional with source code.

Tel: (408) 263-9881

Fax: (408) 263-9883
BBS: (408) 263-0892

Kansmen Corporation
P.O. Box 360070
Milpitas, California 95036
USA

**Source code for the
Borland 3D Chart is available!**



Split windows

	First Name	Last Name	St	Top Sales	Sales > 50K
2	Abby	Coleman	California		<input type="checkbox"/>
3	Gab	Fessler	Massach		<input type="checkbox"/>
4	Carl	Perz	Texas		<input type="checkbox"/>
5					<input type="checkbox"/>
16	Vern	Sperman	Michiga		<input type="checkbox"/>
18	John	White	Arizona		<input type="checkbox"/>
19	Pete	Hanilton	Californ		<input type="checkbox"/>
28	Mark	Sawyer	Wiscon		<input type="checkbox"/>
21	Jack	Bradley	Illinoi		<input type="checkbox"/>
22	Lisa	Hayes	Marylan		<input type="checkbox"/>
23	Mass	Smith	Kansas		<input type="checkbox"/>
24	Dan	Jackson	Florida		<input type="checkbox"/>
25	Alan	Blatz	Indiana		<input type="checkbox"/>
26	Gina	Wilson	Utah		<input type="checkbox"/>

**Consulting &
Contract Programming Available**

□ Request Reader Service #247 □

Modify() implements a state table to respond appropriately to sequences of mouse events. A left button down (Windows message *WM_LBUTTONDOWN*) on a hot spot triggers a state transition from *WAITING* to *START_MODIFYING*, causing drag initialization. As the drag progresses, sending a stream of *WM_MOUSEMOVES* to *Modify()*, the state changes to *MODIFYING*, in which the dashed target rectangle on screen is updated continuously as the mouse moves. Finally, when the left button is released (signaled by receipt of message *WM_LBUTTONUP*), the entire image is resized according to the new target rectangle, and the state reverts to *WAITING*, the quiescent state between drags. State tables are a way to grasp complex interactive sequences like this. Dragging represents a circuit through the state table,

WAITING -> START_MODIFYING -> MODIFYING -> WAITING

with state transitions triggered by receipt of a certain message while in a certain state.

Details about each kind of drag are relegated to helper functions like *DrawDraggingRect()*, which is called in states *START_MODIFYING* and *MODIFYING* to redraw the dashed target rectangle as it is being dragged. Another helper is *UpdateScale()*, which updates *TargetRect* and *Scale* as appropriate when the drag is finished. *UpdateScale()* consists mainly of a switch updating *TargetRect* appropriately depending on the type of modification. A top left drag, for example, begins when *Modify()* learns that the drag started with the

Listing 1 *continued*

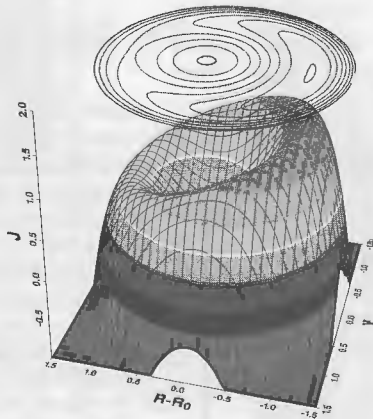
```

scaleY = (double) ClientRect.bottom / WPGsp.cy;

/* Fit target rect into client rectangle with same
 * aspect ratio as WPG space, allowing for margins.
 */
if (scaleX < scaleY)
{
    /* TargetRect fits snugly left and right, with
     * extra space top and bottom.
     */
    scaleX *= (1.0 - 2.0 * INIT_MARGIN);
    /* Calc TargetRect with margin based on INIT_MARGIN
     * on left/right, margin = top units on top/bottom
     */
    top = (ClientRect.bottom - scaleX * WPGsp.cy) / 2;
    lpr->left = ClientRect.right * INIT_MARGIN;
    lpr->top = top;
    lpr->right = ClientRect.right * (1.0 - INIT_MARGIN);
    lpr->bottom = ClientRect.bottom - top;
    /* Calc scale parms based on this TargetRect.*/
    CalcScaleParms(WPGsp, *lpr, lps);
}
else
{
    /* Target rect fits snugly top and bottom, with
     * extra space left and right.
     */
    scaleY *= (1.0 - 2.0 * INIT_MARGIN);
    /* Calc TargetRect with margin based on INIT_MARGIN
     * on top/bottom, margin = left units on left/right
     */
    left = (ClientRect.right - scaleY * WPGsp.cx) / 2;

```

Orbits correspond to $J=\text{constant}$ contours



$$J = 1 - (r^2 - 1)^2 + \epsilon \cos(\theta)$$

Version 7.0 For DOS, DOS '286, OS/2, Windows and Windows NT

- High resolution vector graphics
- 248 colors for all plot elements
- Log plots to any base
- 3-D surfaces shaded to level contours
- Printer output limited only by size of output medium
- New — patch plots, box and whisker plots, staircase plots
- Convert high-resolution output to PIC, GEM, HPGL, HPGL2, CGM, SCODL, TIFF, PostScript (Levels 1 and 2), Tektronix 4105 formats, and GUI metafiles
- Create color separations
- Statistical and smoothing functions
- Use any of your PostScript or True-type fonts in GraphiC for only \$59

GraphiC is an environment-independent C-library that allows you to create every sort of scientific and engineering plot. No special knowledge is required to program in GUI environments.

Scientific Endeavors Corporation

508 N. Kentucky St., Kingston, TN 37763

(800) 998-1571; (615) 376-4146; FAX: (615) 376-1571

Request Reader Service #204

November 1994

Developer's Toolkit

ZyINDEX

Fastest, Most Powerful Text Retrieval Technology Available

Based on ZyINDEX—leader from the beginning in PC text retrieval

- Search 1 GB in Less Than 5 Seconds
- Up to 50 Million Documents, 10 GB Total Per Index
- Powerful Searches: Word, Phrase, Proximity, Boolean, Wild Cards and More
- Works Directly with MS Word, WordPerfect, AmiPro, dBASE, ASCII, and Others

Ideal for use with high-level application development environments such as Visual Basic, ToolBook, KnowledgePro, and ObjectVision.

Windows, DOS, and NT libraries available \$3,995

Call for Specs and Demo.
800-894-6602

ZyLAB

Division of Information Dimensions, Inc.

100 Lexington Drive • Buffalo Grove, IL 60089
Phone: (708) 459-8000 • Fax (708) 459-8054

Request Reader Service #111

Windows/DOS Developer's Journal — Page 41

Listing 1 *continued*

```
lpr->top = ClientRect.bottom * INIT_MARGIN;
lpr->left = left;
lpr->bottom = ClientRect.bottom * (1.0 - INIT_MARGIN);
lpr->right = ClientRect.right - left;
/* Calc scale parms based on this TargetRect. */
CalcScaleParms(WPGsp, *lpr, lps);
}

void LOCAL CalcScaleParms(SIZE wpgsp, RECT tr,
                          LPSCALE lps)
/*
USE: Calculate scale parameters.
IN:  wpgsp = WPG space size, x = width, y = height
     tr    = target rect on screen where image goes
OUT: lps = ptr to struct with scale parameters
*/
{
/* Scale equation solutions in x. */
lps->scaleX = (double) (tr.right - tr.left) / wpgsp.cx;
lps->offsX = tr.left;

/* Scale equation solutions in y. */
lps->scaleY = (double) (tr.top - tr.bottom) / wpgsp.cy;
lps->offsY = tr.bottom;
}
```

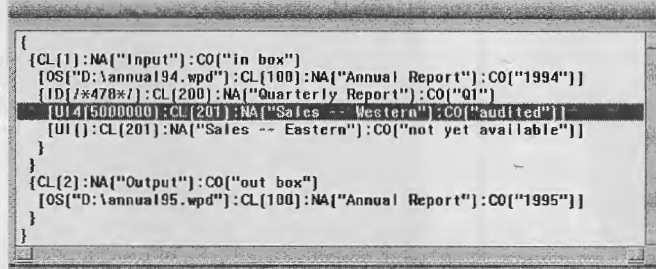
user's depressing the left mouse button on the top left knob, or hot spot. When the user later releases the left mouse button, after dragging, *Modify()* calls *UpdateScale(hs,*

pt) to adjust *TargetRect* appropriately, where *hs* is the hot spot variable holding the type of modification and *pt* is the point at which the user released the left mouse button. This is the new top left corner of *TargetRect* and is all the information needed to recalculate *TargetRect*. Once the switch has determined the new *TargetRect*, *UpdateScale()* calculates the new scale parameters and stores them in global structure *Scale*.

SHOWWPG as a Graphics Filter

Writing GDI calls to a screen DC (display context) displays the image, while writing the GDI calls to a metafile DC creates a metafile. A Windows metafile is a static list of GDI calls which can be saved as a disk file or, as in SHOWWPG, copied to the clipboard. One of *PlayWPGFile()*'s parameters is an *hDC*, or handle to a display context; this is the sole determinant of whether the image is displayed or translated to a metafile. Once the metafile has been transferred to the clipboard, you have truly captured it in the Windows system, since any Windows application capable of pasting metafiles from the clipboard can now retrieve the image. The image can be manipulated or added to within the retrieving program, and resaved in the native format of the program or any other format the program can export to. SHOWWPG has filtered the WPG graphic into Windows. □

Can You Browse Your Program's Files?



The Gamelon Browser. One of the productivity tools included with Gamelon, the object-based file API for developers. There's nothing like Gamelon. Create free-form or structured multi-platform files easily and save time and money in the process. Available now, single-user, royalty-free. Windows 3.1: \$395. OS/2 or Windows NT: \$495. C and C++

gamelon.
File I/O Library

Menai Corporation

1010 El Camino Real, Suite 370, Menlo Park, California 94025
1.800.GAMELON • FAX 415.853.6453 • BBS 415.617.5726 • info@menai.com

□ Request Reader Service #102 □

Notice To Our Subscribers

Occasionally, *Windows/DOS Developer's Journal* makes its mailing list available to vendors of products we think our readers will find interesting. Current subscribers receive free information in the mail

If you prefer that your name not be used in these mailings, please let us know. Just copy or clip this form and send it with your name and address to:

Windows/DOS
DEVELOPER'S JOURNAL

Suite 200
1601 West 23rd Street
Lawrence, KS 66046 USA