



“Borland” and “Borland C++” are registered trademarks, and “Delphi” is a trademark of Borland International, Inc.

“IBM” and “OS/2” are registered trademarks of International Business Machines Corporation.

“Microsoft” and “Microsoft Windows” are registered trademarks, and “Visual Basic” and “Visual C/C++” are trademarks of Microsoft Corporation.

“Paint Shop Pro” is a trademark of JASC, Inc.

“Watcom” is a trademark of Sybase, Inc.

“CompuPhase” and “EGI” are registered trademarks of ITB CompuPhase, and “AniSprite” and “Panodome” are trademarks of ITB CompuPhase.

Copyright © 1997–1999, ITB CompuPhase, Brinklaan 74-b, 1404 GL Bussum, Netherlands (Pays Bas); tel.: +31 35 693 9261; fax: +31 35 693 9293; CompuServe: 100115,2074; e-mail: info@compuphase.com; WWW: <http://www.compuphase.com>

No part of this manual may be reproduced or transmitted by any means (electronic, photocopying or otherwise), without prior *written* permission by ITB CompuPhase.

The examples and programs in this manual have been included for their instructional value. They have been tested with care, but are not guaranteed for any particular purpose.

Typeset with T_EX in the “Computer Modern” and “Pandora” typefaces at a base size of 11 points.

Table of Contents

INTRODUCTION	1
Panorama overview	2
MAKING PANORAMIC IMAGES	4
Making adjustments to the panoramic image	6
Storing panorama attributes	7
USING THE PANODOME ENGINE	10
Compiling and linking	10
Opening a panorama	10
Creating a viewer window	11
Responding to events	12
Closing down	13
Adding a compass	13
FUNCTION REFERENCE	15
DATA REFERENCE	33
MESSAGE REFERENCE	35
ERROR CODES	37
APPENDICES	37
A: Handling DIBs	38
B: Resources	40
INDEX	43

Introduction

Panodome is a toolkit for embedding views of cylindrical panoramic images into an application. Making a standard panoramic image navigator with Panodome requires only a few simple function calls. Panodome also lets you extent the viewer and combine it with sound and animation toolkits, in order to create lively panoramas, or to make interactive games.

Features:

- ◇ High performance, critical routines hand-optimized in assembler.
- ◇ Combines with EGI (a frame animation toolkit), AniSprite (a sprite animation toolkit) and many other graphic and animation libraries.
- ◇ Support for hot spots (which generate events) and tooltip or balloon text on these hot spots.
- ◇ Free pan, tilt and zoom.
- ◇ Compass orientation.
- ◇ Partial cylinders, horizon correction, . . .
- ◇ Multiple concurrent panoramic views.
- ◇ Windows DLL interface, libraries/definition files for C/C++.
- ◇ Many configuration options; both low level and high level functions.
- ◇ Source code is available.

Panoramic images create a photo-realistic, three dimensional, navigable environment. A panoramic image captures the surroundings of a location in a 360° cylindrical environment. The spectator is immersed in this environment and he or she has control over viewed area, through panning, tilting, zooming and hot spot selection. The Panodome viewer warps the panoramic image to create the illusion that the spectator is immersed in a 3D space, rather than just scrolling through a flat panoramic image.

Panoramic imaging presents opportunities to a variety of entertainment and educational uses. Adventure style games with realistic (photographed or scanned) or semi-realistic backgrounds are one example. Guided tours and real estate images are another (museums, exhibitions, building sites).

When compared with full 3D engines, panoramic images represent a trade-off between image quality, speed of display and the ability to move freely through a scene. Panoramic imaging takes the path of pre-rendering the scenes as much as possible. This allows you to use high quality, very detailed, photo-realistic images for the panorama. 3D engines are usually restricted to textures of limited size and of limited number and the number of polygons in a scene directly affects

the frame rate. The drawback of panoramic imaging is that you can only move along pre-computed paths.

Game and multi-media objects, whether these are flat sprites or volume-objects generated by a real-time 3D renderer, can be put on top of a panorama. This technique allows to spend most of the available computer power on rendering the (relatively small) foreground objects, and have beautifully high resolution backgrounds at low processor cost.

Panorama overview

There exist at least three types of panoramic images: cylindrical, cubical and spherical. The techniques underlying each kind of panoramic image are quite different. Hence, it is important not to confuse the varieties.

Spherical panoramic images are usually taken with a fish-eye lens. The photographer captures two pictures in opposite directions using a camera with a fish-eye lens. Each picture covers a hemisphere; the combination of both pictures covers the full sphere. The quality of a spherical panorama is uneven, because the source images are much denser along the (circular) circumference than in the centre. That is, the pixels in the centre of the image represent a much smaller area of the photographed scene than the pixels near the circumference. The effect of a fish-eye lens is that of a magnifying lens: there is much detail in the centre, but very little detail at a 90° angle.

Cubical panoramic environments consist of 6 images, one for every face of a cube. The images for cubical panoramas are usually generated by a computer, because the angle of a camera lens would need to be 90° to capture the required images (and still produce rectilinear images).

Cylindrical panoramas offer only limited freedom of “tilting”. That is, one can turn around in a full circle, but the amount by which one can look up and down is restricted. The images for a cylindrical panorama are best captured with a special panoramic camera with a rotating lens and a vertical diaphragm. However, the most common method of image acquiring for panoramic images is to capture a series of pictures with a standard camera that turns around a single point of rotation, usually on a tripod. After digitizing the pictures into a set of computer image files, these images are stitched together with computer software. The source images can be captured with nearly any type of lens and camera combination; a common “35 mm film” camera is adequate.

Cylindrical panoramas are the most popular of the panorama varieties, and they are the only type supported by Panodome. One reason why the use of cylindrical is so wide spread in comparison to spherical or cubical panoramas is that cylindrical panoramas are relatively easy to create using common tools.

Making panoramic images

A panoramic image is usually made from one or more photographs that are scanned in (or, in the case of a digital camera, simply transferred). For computer generated panoramic images, look at appendix B for a link to Wasabi software's "SkyPaint". Rendering packages like POV-ray and Bryce can also produce panoramic images.

The first step in making a panoramic image, suitable for Panodome to display, is to take the photographs. Here, already, you have several choices:

- ◇ Put a standard camera on a tripod and take 8 to 12 pictures while rotating. Post-processing software later warps and stitches the images together in a way that mimics the operation of a panoramic camera.
- ◇ Put a video camera on a tripod and rotate it around the (vertical) axis while filming. After digitizing the movie, you have a large number of overlapping frames to choose from (stitching software usually needs pictures with 30% to 50% overlap, so you do not need all frames of the movie). The quality of the final panoramic image is not very high, because of the unavoidable noise in individual frames of a video stream.
- ◇ Use a panoramic camera with a narrow vertical shutter and a rotating lens. These cameras are expensive and the film that they use may be difficult to develop and to scan. The biggest advantage of panoramic cameras is that no post-processing software is required (and hence, no quality loss).
- ◇ Attach a panoramic lens on a standard camera. The lens is actually a cone-shaped mirror that is mounted in front of the camera lens. The picture that is taken via the panoramic "lens" is heavily deformed, which must be corrected with special software. Panoramic lenses are fairly expensive.

The first alternative in the above list is also the most common. It achieves high quality at low material costs. Because of the number of photographs that must be scanned in and the post-processing step, this technique may become quite costly, time-wise, when you have to create a large number of photographs, however.

To use the "standard camera on a tripod" technique, there are a few criterions to which you must adhere:

- ◇ The camera must rotate on an axis perpendicular to the axis of the lens. For a horizontal horizon, this means that the axis through the lens must be level (spirit-level) at every orientation.

- ◇ The tripod should have a radial scale, or the camera should be corrected for parallax. Otherwise, getting even angle increments while taking pictures becomes very hard. Note that stitching software needs the pictures to overlap, and that the software usually specifies a minimum and a maximum overlap.
- ◇ The lens must produce rectilinear images—that is, straight lines in the input image must appear as straight lines in the resulting image. This rules out fish-eye lenses.
- ◇ When photographing objects near to the camera (say, within one meter of the lens), the rotation point should also be precisely below the lens. Most cameras, when fitted on a tripod, rotate around a point that is 20 to 40 millimeter behind the lens.

Some manufacturers sell “panoramic tripod heads” that keep the camera horizontally leveled, automatically snap to the desired angles and that position the rotation point of the camera just below the lens. There exist even motorized tripod heads, see also appendix B.

While I mentioned a “radial scale” above, one issue I have stepped over is by how many degrees you must turn the camera after each picture. Another way of asking basically the same question is: how many pictures must one take for the photographs to stitch correctly. The first thing to determine is the “field of view” in degrees of a single picture. Some camera or lens manuals give this value, but otherwise you can calculate it from the focal length of the lens.

$$fov = 2 \times \arctan \left(\frac{width/2}{focal\ length} \right)$$

where *width* is the usable width of the film in millimeter. The frame size of common 35 mm film is 36 × 24 millimeter, but the frame edges are often cropped off, typically reducing the frame size by 5%. That is, in landscape mode, a good rule-of-thumb value for the *width* is 34 mm. The “*focal length*” value is the focal length of the lens, also in millimeter.

As an example: the field of view (*fov*) of a common camera with a lens with a focal length of 35 mm is approximately 52°.

Stitching software requires a minimum overlap for the photographs; a common value is 20°. When subtracting the required overlap from the field of view, you get the maximum angle at which the camera may turn from one photograph to the next. To continue my example of the standard camera with a 35 mm lens, the field of view is 52° and the *maximum* rotation angle is 32°. The number

of pictures required flows from this angle, because a full circle is 360° . In this example, the number of pictures is 12 (the division $360/32$ gives 11.25, which must be rounded upwards). The final step is to get the final rotation angle from this “number of pictures” (note that the angle calculated earlier represented the *maximum* rotation angle; for stitching it is better to make all pictures overlap by the same amount, including the first and last pictures). Again, one finds this value by dividing 360° by the number of pictures. Here, the rotation angle becomes 30° .

Making adjustments to the panoramic image

After acquiring the panoramic image in one way or another, the next step is to make it ready for the Panodome viewer. The Panodome viewer itself supports only Windows Bitmap images with 256 colours. This file type may be directly supported by your stitching/post-processing software; if not, use one of the many graphic image editors to convert the output of the stitching software to the Windows Bitmap file format.

Alternatively, you can use one of the many graphic libraries to load an image into memory and pass it on to the Panodome functions. Panodome requires the image in an in-memory DIB format. Any graphic library for Windows that is worth its salt can provide you the images that they load in DIB format.

• Horizon adjustment

In the initial photographs, the horizon is precisely halfway the height of each photo (this flows from the requirements for taking the photographs mentioned earlier). After stitching and cropping the image, this no longer needs to be true, however. Panoramic images that are taken with a panoramic “lens” (actually a mirror) also need not to have the horizon in the middle of the image.

If the horizon is not exactly halfway the height of the panoramic image, you should add a horizon adjustment instruction to the image. The horizon adjustment is an integer value that gives the offset in pixels of the horizon from the vertical centre of the image. For example, if the panoramic image is 100 pixels high, and the horizon is at 30 pixels from the top of the image, then the horizon adjustment should be 20 (or, $100/2 - 30$). In other words, the horizon is 20 pixels above the vertical middle line of the image.

See the section “storing panoramic attributes” below for how to attach the horizon adjustment to the image.

- **Make sure the image has a width that is a multiple of 4 pixels**

For technical reasons, the Panodome viewer requires that each input image has a width that is a multiple of four pixels. This requirement follows from the way that Windows stores DIBs (device independent bitmaps). Many stitching programs already generate images that comply with this requirement, and otherwise, you need to stretch or compress the image width to the nearest multiple of four pixels with any graphic editor.

You can also call `pan_ScaledIB` to scale a image at run time, if needed. This is what the `PANVIEW` example program does.

Storing panorama attributes

The attributes, like the horizon correction and the angle covered by a non-full cylinder panorama, are best stored inside the same computer file that also holds the pixels of the panoramic image. Several formats, like CompuServe's GIF and PPM ("Portable Pixel Map") support the inclusion of general purpose comments in the picture file, which suits this purpose perfectly. Other file formats, like ZSoft's PCX or Microsoft's BMP, do not allow such extensions and, hence, require that the properties be stored in a separate companion file.

At the lowest level, the Panodome engine holds the properties in a structure that is documented at page 33 (the "PANATTRIB" structure). Whatever the source of the panoramic image properties, filling in this structure with the relevant information is all that is needed.

In the (common) cases that the panoramic attributes are available either in a string, for example from a "comment" field from the image file, or in a text file, Panodome offers two convenient functions to parse the attributes and fill in the low level structure. The string-based function `pan_AttribString` can handle ASCII/ANSI strings that are optionally, but not necessarily, zero-terminated. Function `pan_AttribFile` retrieves the required information from a text file.

Both functions use the same syntax to specify the properties of the panoramic image. As stated before, the information is kept in a stream (or string) of ASCII/ANSI characters. In this string, each attribute starts with a keyword and it has its value or parameter enclosed in parentheses behind the keyword. The keywords defined so far are:

horizon(*value*)

Gives the offset of the “horizon” of the panoramic image from the vertical centre of the image. A positive value indicates that the horizon (of the panoramic image) is *above* the vertical centre, a negative value moves the horizon *below* the vertical centre. See also the section “horizon adjustment” on page 6. If absent, the horizon is assumed to be exactly at the vertical centre of the image.

angle(*value*)

The field of view (in degrees) that the panoramic image presents. If absent, Panodome assumes the image to be a full 360° panorama.

orientation(*value*)

The position on a compass of the left edge of the panoramic image. This setting is in degrees counter-clockwise. For example, when setting the orientation to 90°, the left edge of the image points to the West.

hotspot(*x1 y1 x2 y2 label text*)

Adds a rectangular hot spot to the image. The “label” is a general purpose string whose meaning depends on the application using the Panodome viewer. A common case is that the label is the filename or the URL of the panoramic image that one jumps to when clicking on the hot spot.

The “text” is a general purpose string that may serve for tooltips or balloon texts. The “text” parameter is optional.

Several hot spots (with different coordinates) may refer to the same label.

If the strings for the label or the text contain space characters or other non-alphanumeric characters, they should be enclosed in quotes. Both single and double quotes are acceptable.

userstring(*string*)

A general purpose string that a viewer may use for any purpose. A typical employ of the user string is to set a sound track for the panorama.

Line breaks, spacing and indenting are all irrelevant in the attribute string. The keywords in the attribute string should be all lower case, the functions `pan_AttribString` and `pan_AttribFile` are case sensitive. To accommodate

the PPM file format, the “#” character, which is used for comments in PPM files, is equivalent to white space.

When stored as a separate file, the suggested extension is .PAN. Below is an example of several settings for the “Arena” example image.

```
horizon(-20)
angle(360) orientation(90)
hotspot(404 167 463 252 "@close" "Leave the panorama")
```

FIGURE 1: *Syntax of the panorama attribute file*

Using the Panodome engine

This chapter presents a simple panoramic image viewer. The code snippets are based on the “PANVIEW” example program that comes with the Panodome toolkit.

The following sections describe the process in more detail, but basically the steps needed to display a panorama are:

- ◇ Open a panorama with `pan_Create` after reading in the image bits and (optionally) the panorama attributes.
- ◇ Create a navigation window for the panorama and to attach the panorama to that window.
- ◇ Add settings to the panorama; respond to events while navigating.
- ◇ Destroy the window and close the panorama upon exit.

Note that, in pure example spirit, error checking is lacking from the code snippets in this chapter. In real live programs, you should of course check the return value of each function.

Compiling and linking

Each source file that calls the Panodome functions should include `PANODOME.H`.

Linking is less straightforward, due to incompatibilities between the tools of compiler vendors there is a library file per supported compiler brand:

- ◇ `PAN32M.LIB` for Microsoft Visual C/C++ version 4.0 or higher, and for other compilers that are compatible with the COFF object library format.
- ◇ `PAN32B.LIB` for Borland C++ 4.5 or higher.
- ◇ `PAN32W.LIB` for Watcom C/C++ 10.6 or higher.

Opening a panorama

The first step is to open the panoramic image and to read it in memory in the DIB format. The Panodome toolkit only reads in 256 colour Windows Bitmap (“.BMP”) files from disk, either in uncompressed or RLE compressed format. It does not have functions to read image files from any of the other common raster image file formats, but many imaging toolkits floating around on the marketplace can provide the image data in DIB format.

Next to the image, you will also want to read in the attribute file for the panoramic image, if one exists, or fill in a “PANATTRIB” structure by hand (see

page 33 for the field descriptions). When filling in the PANATTRIB structure manually, make sure that you set the “reserved” field to zero; this field may be used in future versions of the Panodome viewer. The “flags” field should also be zero.

With the raw image data and the attributes in hand, a call to `pan_Create` is all that is needed to obtain a handle to a panorama. With the handle, one can already extract a “view” and pass it on to a sprite library to display it. The simplest way to display any view from a Panodome panorama, and to obtain some navigational capabilities, is to create a special window for it... which is discussed next.

```

LPBITMAPINFO Image;
PANATTRIB Attribs;
PANINFO Pan;
RECT rect;

...

Image = pan_LoadDIB("arena.bmp", NULL);
pan_AttribFile("arena.pan", &Attribs);
GetClientRect(hwnd, &rect);

Pan = pan_Create(Image, NULL, &rect, &Attribs);

```

FIGURE 2: *Opening the panorama*

As a side note, `pan_Create` *copies* the information that it needs from the DIB header and the PANATTRIB structures, but it does not copy the bits of the image. In other words, you may not free the memory to the DIB image bits (the pointer `Image` in the above example) until the panorama is closed.

Another aside is that the DIB format separates the image header (“BITMAPINFO”) from the pixel data. Many graphic libraries, however, allocate the header and the pixel data in one continuous memory chunk. Panodome can work with both schemes. As used in the above example, `pan_LoadDIB` returns a single pointer to a memory block with the DIB header prepended to the pixel data, and `pan_Create` accepts this format. See appendix A for details.

Creating a viewer window

You must open a panorama before you can create a viewer window. The viewer window is also called the “navigation window”, since it allows you to pan, tilt and zoom through the panoramic image.

There is no need to specify a location for the viewer window; the window uses the same location (and size) as that of the viewport. Many other attributes for the viewer window are also copied from the previously opened “panorama” data structure. For the call to `pan_CreateWindow`, only three parameters are needed: the panorama, the window style and the parent window.

A typical style for the viewer window is that of a child window that is embedded inside its parent. Child windows do not automatically receive the input focus upon creation. To allow keyboard navigation, it is therefore best to set the focus to the viewer window *explicitly* after creation.

```

HWND hwndPan;
...
hwndPan = pan_CreateWindow(Pan, WS_VISIBLE | WS_CHILD, hwndParent);
SetFocus(hwndPan);

```

FIGURE 3: *Creating a viewer window & set the input focus*

The parent window is also the window to which the viewer window sends its navigation *events*, like clicking on a hot spot. You can change this afterwards by calling `pan_SetHandle` with the code `PAN_HANDLE_NOTIFYWND`.

Responding to events

The user can freely pan and tilt in the panorama or zoom in and out. There are some actions, however, that the application wishes to be kept informed about, such as a click on a hot spot. The viewer window sends such notification messages to its parent window (or any other window of your choosing).

By default, the message that the viewer windows sends is `PAN_NOTIFY`, which is defined as `(WM_USER+79)`. In case this message conflicts with other “user” message numbers that the application uses, you can change this message number with `pan_SetValue`.

The notification message only gives you the index (or sequence number) of the hot spot that fired. To get the more useful attributes, such as the hot spot’s label, use `pan_GetData`.

```

PANINFO Pan;
LPSTR label;
...
case PAN_NOTIFY:
    if (wParam == PN_SELHOTSPOT) {

```

```

    label = pan_GetData(Pan, PAN_DATA_HOTSPOTLBL, LOWORD(lParam));
    GoToNextPanorama(label);
} /* if */
break;

```

FIGURE 4: *Intercepting a click on a hot spot*

The function `GoToNextPanorama` is one that would be written by you. Of course, you can do anything else upon reception of an event.

Closing down

Eventually, you will have to clean up the panorama. There are two things that you must explicitly clean: the panorama and the DIB image bits for the panorama that you passed to `pan_Create` earlier. Function `pan_Delete` removes the panorama data structure; it can also delete the image bits, provided that the image was loaded with `pan_LoadDIB` or that the memory for the image data was allocated with `pan_AllocResource`. Closing the panorama automatically destroys the viewer window (if there is one).

```

PANINFO Pan;
...
pan_Delete(Pan, TRUE);

```

FIGURE 5: *Closing a panorama & deleting the image*

Adding a compass

Panodome provides a simple compass that you can put on the screen to show the current orientation. This can keep you from “getting lost” in a presentation based on panoramas. Creating a compass is a simple matter of calling `pan_CreateCompass`. To set the position of the needle of the compass, you must also monitor the `PN_MOVE` event of viewer window.

```

PANINFO Pan;
HWND hwndCompass;
int orientation;
...
hwndCompass = pan_CreateCompass(20, 40, hwndParent, FALSE);
...
case PAN_NOTIFY:
    switch (wParam) {

```

```
case PN_MOVE:
    if (IsWindow(hwndCompass)) {
        orientation = pan_GetValue(Pan, PAN_VALUE_ORIENTATION);
        PostMessage(hwndCompass, PAN_COMMAND, PC_ORIENTATION, orientation);
    } /* if */
    break;
} /* switch */
break;
```

FIGURE 6: *Setting the compass*

The final parameter in `pan_CreateCompass` determines whether the compass window is a child window or a popup window. The user can move a “popup compass” around by clicking and dragging anywhere inside the window. The location of a “child compass” is fixed (relative to its parent window).

Function reference

First a few global remarks:

- ◇ Since the C++ classes are just a thin layer on top of the C function set, they are discussed on the same pages. There are three classes, but the bulk is in the `PanView` class, for the panoramic image manipulation and viewer. The other two classes are `PanCompass` and `PanAttrib` (which manages the attribute strings and the attribute files, see page 7).
- ◇ If a function fails, you can use `pan_Error` to get the reason for failure.

pan_AllocResource Allocate memory

`pan_AllocResource` allocates a memory block.

Syntax: `LPVOID pan_AllocResource(long Size)`

`Size` The requested number of bytes.

Returns: A pointer to a memory block of at least the requested size, or `NULL` if unsuccessful.

Notes: The memory block should be freed with `pan_FreeResource`.

See also: `pan_FreeResource`

pan_AttribFile Read panorama attributes from a file

`pan_AttribFile` reads panoramic image settings from a specified text file.

Syntax: `BOOL pan_AttribFile(LPCSTR Filename, LPPANATTRIB Attribs)`

`BOOL PanAttrib::File(LPCSTR Filename)`

`Filename` The complete filename (including extension) of the text file with the panorama attributes.

`Attribs` A pointer to the structure where to store the panorama attributes in.

Returns: `TRUE` on success and `FALSE` on failure.

See also: `pan_AttribString`

pan_AttribString Read panorama attributes from a string

`pan_AttribString` parses a text string that contains attributes for a panoramic image. The string is optionally zero-terminated.

Syntax: `BOOL pan_AttribString(LPCSTR CmdString, int Length,
LPPANATTRIB Attribs)`

`BOOL PanAttrib::String(LPCSTR CmdString, int Length=-1)`

`CmdString` The string containing the panorama attributes.

`Length` The length of the “string” parameter. If set to -1, the function assumes that “string” is zero-terminated.

`Attribs` A pointer to the structure where to store the panorama attributes in.

Returns: `TRUE` on success and `FALSE` on failure.

See also: `pan_AttribFile`

pan_CheckHotSpot Check for hot spots at a location

`pan_CheckHotSpot` determines if a coordinate pair (in the viewport) points into a hot spot (in the source image). If the coordinate pair is at a hot spot, the function also returns the sequence number of the hot spot.

Syntax: `BOOL pan_CheckHotSpot(PANINFO Pan, int X, int Y,
LPINT Index)`

`BOOL PanView::CheckHotSpot(int X, int Y, LPINT Index)`

`Pan` The panorama handle.

`X, Y` The position in the viewport.

`Index` Will hold the sequential number of the hot spot at the indicated position upon return. This parameter may be `NULL`.

Returns: `TRUE` if the location (X,Y) is on a hot spot and `FALSE` otherwise.

Notes: You can use the returned index of the hot spot to query the label or text fields of the hot spot with `pan_GetData`. Hot spot information is attached to the panorama in the call to `pan_Create`.

The viewer window sends a notification message to inform the parent window of clicks or hits on a hot spot.

See also: `pan_GetData`, `pan_Create`

pan_Create

Create a panorama

`pan_Create` creates an instance of a panorama and allocates all memory that it needs for the manipulation of the panorama. The function does not create a window for the panorama, but the position and size of this (optional) window are set in `pan_Create`.

Syntax: PANINFO pan_Create(LPBITMAPINFO BitsInfo,
LPVOID ImageBits, LPRECT ViewPort,
LPPANATTRIB Attribs)

BOOL PanView::Create(LPBITMAPINFO BitsInfo,
LPVOID ImageBits, LPRECT ViewPort,
PanAttrib *Attribs=NULL)

BitsInfo A pointer to the DIB header of the source image.

ImageBits A pointer to the pixel data of the source image. If this parameter is NULL, the function assumes that the pixel data follows the DIB header (in the `BitsInfo` parameter) in a single memory block.

ViewPort The position and size of the viewport.

Attribs Points to a structure with the attributes and settings of the panorama. If NULL, the function assumes a cylindrical panorama with no horizon correction and no hot spots.

Returns: The C functions return the PANINFO handle that most other Panodome functions need. The C++ method returns TRUE on success and FALSE on failure.

See also: `pan_AttribFile`, `pan_AttribString`, `pan_CreateWindow`, `pan_LoadDIB`

pan_CreateCompass Create a compass window

`pan_CreateCompass` creates a small window with a circular scale (North, West, South and East) and a needle. The compass window may be used to indicate the orientation in a panorama.

Syntax: `HWND pan_CreateCompass(int X, int Y, HWND hwndParent, BOOL Child)`

```
HWND PanCompass::Create(int X, int Y,
                        HWND hwndParent=NULL,
                        BOOL Child=FALSE)
```

X, Y The location of the upper left corner of the compass window.

hwndParent The window handle of the parent window. Use `NULL` if the compass window should not have a parent.

Child If `TRUE`, the compass window is created as a “child” window of the specified parent. If `FALSE`, the compass window becomes a popup window.

Returns: The window handle of the compass window.

Notes: The size of the compass window is 65×65 pixels.

To set the position of the needle, you must send the compass window a `PAN_COMMAND` message. C++ programs can also use the `SetOrientation` method of the `PanCompass` class.

See also: `PAN_COMMAND`

pan_CreateWindow Create a panorama viewer window

`pan_CreateWindow` creates a viewer window for a previously opened panorama. The viewer window provides navigational capabilities and hot spot detection.

Syntax: `HWND pan_CreateWindow(PANINFO Pan, DWORD dwStyle, HWND hwndParent)`

```
HWND PanView::CreateWindow(DWORD dwStyle,
                            HWND hwndParent=NULL)
```

Pan	The panorama handle.
dwStyle	The window style. This must be a combination of the WS_ <i>xxx</i> flags from the CreateWindow function of the Microsoft Windows API. When you set the dwStyle parameter to zero, the function creates a (visible) child window without border.
hwndParent	The window handle of the parent window. Use NULL if the viewer window should not have a parent. You <i>must</i> give a valid parent window when the dwStyle parameter includes the WS_CHILD flag. The parent window is also the window that receives the notification messages, by default.

Returns: The window handle of the viewer window, or **NULL** on failure.

See also: **pan_Create**

pan_Delete

Close the panorama

pan_Delete closes a panorama and releases all memory.

Syntax: **BOOL pan_Delete(PANINFO Pan, BOOL DeleteDIB)**

BOOL PanView::Close(BOOL DeleteDIB=FALSE)

Pan The panorama handle.

DeleteDIB If **TRUE**, the function also frees the memory for the source image that was passed to **pan_Create**. The memory resource for the source image must have been allocated with **pan_AllocResource** for this to work.

Returns: **TRUE** on success and **FALSE** on failure.

Notes: After calling this function, any reference to the “**Pan**” handle is invalid. **pan_Delete** frees all memory allocated for the animation.

See also: **pan_Create**

pan_Error Return the error code

`pan_Error` returns the most recent error code for a panorama.

Syntax: `int pan_Error(PANINFO Pan)`

`int PanView::pan_Error()`

Pan The panorama handle.

Returns: The last error code; zero if the latest function was successful.

Notes: If the **Pan** parameter is `NULL`, the `pan_Error` function returns the latest *global* error number.

See page 37 for a list of error codes.

pan_FreeResource Release a memory resource

`pan_FreeResource` releases a memory block back to the operating system. The memory block must have been allocated by a Panodome function for this to work correctly.

Syntax: `LPVOID pan_FreeResource(LPVOID Resource)`

Resource A pointer to a memory block that was previously allocated by `pan_AllocResource` or another Panodome function.

Returns: This function always returns `NULL`.

Notes: `pan_FreeResource` frees memory blocks that are allocated by the following functions:

`pan_AllocResource`

`pan_LoadDIB`

`pan_ScaleDIB`

See also: `pan_AllocResource`, `pan_LoadDIB`, `pan_ScaleDIB`

pan_GetData

Access internal data

`pan_GetData` returns pointers to data tables that the Panodome engine maintains for each panorama.

Syntax: LPVOID `pan_GetData`(PANINFO Pan, int Code, int Index)

LPVOID `PanView::GetData`(int Code, int Index=0)

Pan The panorama handle.

Code Specifies the data that is requested. It is one of the following:

PAN_DATA_BITMAPINFO

A pointer to the DIB header for the viewport.

PAN_DATA_HOTSPOTLBL

A pointer to the label of a hot spot. The “**Index**” parameter gives the sequential (zero-based) number of the hot spot.

PAN_DATA_HOTSPOTLIST

A pointer to the raw hot spot list, which is implemented as an array of `PANHOTSPOT` structures.

PAN_DATA_HOTSPOTTEXT

A pointer to the “text” of a hot spot. The “**Index**” parameter gives the sequential (zero-based) number of the hot spot.

PAN_DATA_IMAGE

A pointer to the DIB of the source image (the image passed to `pan_Create`).

PAN_DATA_LENS

A pointer to the “lens” data structure. The “lens” is a data structure that contains the warping information that `pan_GetView` uses.

PAN_DATA_VIEWPORT

A pointer to the pixel data of the DIB for the viewport.

Index The meaning of the “Index” depends on the Code parameter.

Returns: A pointer to the data that was requested, or NULL on failure.

See also: `pan_GetHandle`, `pan_GetRect`, `pan_GetValue`

pan_GetHandle Access internal handles

`pan_GetHandle` returns handles to various Microsoft Windows objects that the Panodome engine maintains for each panorama.

Syntax: `HANDLE pan_GetHandle(PANINFO Pan, int Code)`

`HANDLE PanView::GetHandle(int Code)`

Pan The panorama handle.

Code Specifies the handle that is requested. It is one of the following:

PAN_HANDLE_PALETTE

The handle of the palette that `pan_Create` created for the panorama.

PAN_HANDLE_NOTIFYWND

The handle to the window that receives the notification messages.

PAN_HANDLE_VIEWERWND

The handle to the viewer window; i.e, the handle that `pan_CreateWindow` returned.

Returns: A pointer to the handle that was requested, or NULL on failure.

See also: `pan_GetData`, `pan_GetRect`, `pan_GetValue`, `pan_SetHandle`

pan_GetRect Query bounding boxes or hot spot positions

`pan_GetRect` returns pointers into lists of rectangles that the Panodome engine maintains for each panorama.

Syntax: `LPRECT pan_GetRect(PANINFO Pan, int Code, int Index)`

```
LPRECT PanView::GetRect(int Code, int Index=0)
```

Pan	The panorama handle.
Code	Specifies the rectangle that is requested. It is one of the following: PAN_RECT_HOTSPOT A pointer to the bounding box of a hot spot. The “ Index ” parameter gives the sequential number of the hot spot (starting from zero).
Index	The meaning of the “ Index ” depends on the Code parameter.

Returns: A pointer to the rectangle that was requested, or NULL on failure.

See also: `pan_GetData`, `pan_GetHandle`, `pan_GetValue`

pan_GetValue Read panorama settings and attributes

`pan_GetValue` returns current settings and fixed attributes for a panorama.

Syntax: `int pan_GetValue(PANINFO Pan, int Code)`

```
int PanView::GetValue(int Code)
```

Pan	The panorama handle.
Code	Specifies the value that is requested. It is one of the following: PAN_VALUE_ANGLE The field of view of the source image in degrees. It is 360 for a 360° cylindrical panorama. PAN_VALUE_HORIZON The offset of the horizon from the vertical centre of the source image (see also page 6). PAN_VALUE_ID The “ ID ” of the panorama, which must have been set by <code>pan_SetValue</code> .

PAN_VALUE_MAXZOOM

The maximum zoom factor that is allowed; when zero (the default), there is no maximum.

PAN_VALUE_MSGNUM

The numeric value of the notification message that the viewer window sends. The default message value is PAN_NOTIFY.

PAN_VALUE_NUMHOTSPOTS

The number of hot spots that the panorama has.

PAN_VALUE_ORIENTATION

The orientation (the position of the needle on the compass), based on the current (x,y) position in the panorama.

PAN_VALUE_RADIUS

The radius, measured in pixels, of the cylinder of the panorama.

PAN_VALUE_SPEED

The timer interval in milliseconds at which the viewer window refreshes. The default is 50 ms, which gives 20 frames per second.

PAN_VALUE_SRCHEIGHT

The height of the source image in pixels.

PAN_VALUE_SRCWIDTH

The width of the source image in pixels.

PAN_VALUE_VPHEIGHT

The height of the viewport image in pixels.

PAN_VALUE_VPWIDTH

The width of the viewport image in pixels.

PAN_VALUE_XPOS

The current “horizontal position” in the panorama, measured as the offset in pixels from the left edge of the source image.

PAN_VALUE_YPOS

The current “vertical position” in the panorama, measured as the offset in pixels from top edge of the source image.

PAN_VALUE_ZPOS

The current “distance” from the panoramic screen, measure as the distance in pixels from the centre of the cylinder.

Returns: A value that was requested, or zero on failure.

See also: `pan_GetData`, `pan_GetHandle`, `pan_GetRect`, `pan_SetValue`

pan_GetView Extract an unwarped DIB from the panorama

`pan_GetView` returns a pointer to a DIB that represents an unwarped (perspective corrected) view at the current position of the panorama.

Syntax: `LPVOID pan_GetView(PANINFO Pan, LPBITMAPINFO BitsInfo)`

`LPVOID PanView::GetView(LPBITMAPINFO BitsInfo)`

Pan The panorama handle.

BitsInfo Points to a `BITMAPINFO` structure (a header and a colour table with 256 entries) that will contain the DIB header information for the DIB pixels that the function returns. This parameter may be set to `NULL`.

Returns: A pointer to the DIB pixels.

See also: `pan_CreateWindow`, `pan_Create`

pan_LoadDIB Load a Windows Bitmap image

`pan_LoadDIB` reads an image in the Microsoft Windows “Bitmap” format from disk into memory. The function supports both uncompressed and RLE compressed bitmap images. `pan_LoadDIB` will only read 256-colour images.

Syntax: `LPVOID pan_LoadDIB(LPCSTR Filename, LPBITMAPINFO BitsInfo)`

Filename The complete filename of the image.

BitsInfo Points to a `BITMAPINFO` structure (a header and a colour table with 256 entries) that will contain the DIB header information for the DIB pixels that the function returns. This parameter may be set to `NULL`.

Returns: A pointer to the DIB pixels, or a pointer to the DIB header (see notes).

Notes: If the “`BitsInfo`” parameter is `NULL`, function `pan_LoadDIB` allocates a single memory block for the DIB header and the pixel data. The function returns a pointer to the header; the pixel data immediately follows this header. If “`BitsInfo`” is not `NULL`, `pan_LoadDIB` only allocates memory for the pixels and it stores the header in the structure that “`BitsInfo`” points to.

The memory block that `pan_LoadDIB` allocates must be freed with `pan_FreeResource`.

See also: `pan_Create`, `pan_ScaleDIB`

pan_MapCoordinates

Convert coordinates

`pan_MapCoordinates` converts “viewport” coordinates to “source” coordinates, depending on the current pan, tilt and zoom.

Syntax: `BOOL pan_MapCoordinates(PANINFO Pan, int X, int Y, LPPOINT Point)`

`BOOL PanView::MapCoordinates(int X, int Y, LPPOINT Point)`

Pan The panorama handle.

X, Y The coordinates in the viewport.

Point Will hold the coordinates in the source image (the image passed to `pan_Create`) upon return.

Returns: `TRUE` on success or `FALSE` if the coordinates are invalid.

See also: `pan_CheckHotSpot`

pan_MinImageSize Inquire the minimum image dimensions

`pan_MinImageSize` returns the minimum image width and height that can be displayed in the specified viewport.

Syntax: `BOOL pan_MinImageSize(LPINT Width, LPINT Height, LPRECT ViewPort, int Angle)`

<code>Width,</code>	
<code>Height</code>	The returned minimum width and height for an image that the viewport can hold. The returned width is always a multiple of 4 bytes.
<code>ViewPort</code>	Keeps the size of the viewport.
<code>Angle</code>	The angle that the panoramic image covers; use zero for a full cylinder.

Returns: `TRUE` on success or `FALSE` if the angle or the viewport parameters are invalid.

See also: `pan_Create`

pan_Move Move the viewport relative to its current position

`pan_Move` pans, tilts or zooms the current viewport, relative to its current position.

Syntax: `WORD pan_Move(PANINFO Pan, int DeltaX, int DeltaY, int DeltaZ)`

`WORD PanView::Move(int DeltaX, int DeltaY, int DeltaZ)`

<code>Pan</code>	The panorama handle.
<code>DeltaX</code>	The change in pan, a positive value turns to the right.
<code>DeltaY</code>	The change in tilt, a positive value moves down.
<code>DeltaZ</code>	The change in zoom, a positive value zooms in.

Returns: A series of flags that indicate whether the position was blocked (invalid) in any direction. It is a combination of the flags `PAN_BOUND_X`, `PAN_BOUND_Y` and `PAN_BOUND_Z`.

Notes: It usually makes no sense to zoom in with a factor higher than, say, 10. There is no technical limit on zooming in, but if desired, the maximum zoom factor can be set with `pan_SetValue`.

The minimum zoom factor is bound by the size of the source image and the size of the viewport.

See also: `pan_GetValue`, `pan_SetPos`, `pan_SetValue`.

pan_Replace

Change the image for a panorama

`pan_Replace` replaces the image for a panorama (that was created earlier) while keeping the current viewport settings.

Syntax: `BOOL pan_Replace(PANINFO Pan, LPBITMAPINFO BitsInfo,
LPVOID ImageBits, LPPANATTRIB Attribs,
BOOL DeleteDIB)`

```
BOOL PanView::Replace(LPBITMAPINFO BitsInfo,
                      LPVOID ImageBits,
                      PanAttrib *Attribs=NULL,
                      BOOL DeleteDIB=FALSE)
```

Pan The panorama handle.

BitsInfo A pointer to the DIB header of the new source image.

ImageBits A pointer to the pixel data of the new source image. If this parameter is `NULL`, the function assumes that the pixel data follows the DIB header (in the `BitsInfo` parameter) in a single memory block.

Attribs Points to a structure with the attributes and settings of the new source image. If `NULL`, the function assumes a cylindrical panorama with no horizon correction and no hot spots.

DeleteDIB If `TRUE`, the function also frees the memory for the previous image (the one that `pan_Replace` replaces). The memory resource for the source image must have been allocated with `pan_AllocResource` for this to work.

Returns: `TRUE` on success or `FALSE` on error.

See also: `pan_AttribFile`, `pan_AttribString`, `pan_Create`, `pan_LoadDIB`

pan_ScaleDIB Resize a device independent bitmap

`pan_ScaleDIB` scales a DIB up or down, at pixel precision.

Syntax: `LPVOID pan_ScaleDIB(LPBITMAPINFO SrcInfo, LPVOID SrcBits, LPBITMAPINFO DstInfo, int Width, int Height, BOOL DeleteSrc)`

SrcInfo A pointer to the DIB header of the source image.

SrcBits A pointer to the pixel data of the source image. If this parameter is `NULL`, the function assumes that the pixel data follows the DIB header (in the `SrcInfo` parameter) in a single memory block.

DstInfo A pointer to the DIB header of the resized image. This header is filled by `pan_ScaleDIB` to reflect the new size.

width, height The size of the new image.

DeleteSrc If `TRUE`, the function deletes the source image upon successful completion. The memory block for the source image must have been allocated by a Panodome function for this to work correctly.

Returns: A pointer to the DIB pixels of the scaled image, or a pointer to the DIB header of the scaled image (see notes).

Notes: If the “`DstInfo`” parameter is `NULL`, `pan_ScaleDIB` allocates a single memory block for the DIB header and the pixel data of the scaled image. The function returns a pointer to the header; the pixel data immediately follows this header. If “`DstInfo`” is not `NULL`, `pan_ScaleDIB` only allocates memory for the pixels and it stores the header in the structure that “`DstInfo`” points to.

The memory block allocated by `pan_ScaleDIB` must be freed with `pan_FreeResource`.

See also: `pan_FreeResource`, `pan_LoadDIB`

pan_SetHandle Set internal handles

`pan_SetHandle` adjusts a handle to a Microsoft Windows object. The Panodome engine maintains a few Microsoft Windows objects for its operation.

Syntax: `BOOL pan_SetHandle(PANINFO Pan, int Code, HANDLE Handle)`

`BOOL PanView::SetHandle(int Code, HANDLE Handle)`

Pan The panorama handle.

Code Specifies the handle that must be set. It is one of the following:

PAN_HANDLE_PALETTE

The handle of the palette that the viewer window uses. By default, `pan_Create` creates a palette from the colour table in the source image.

PAN_HANDLE_NOTIFYWND

The handle to the window that must receive the notification messages. By default, this the the window that `pan_CreateWindow` creates.

Handle The value of the handle referred to by the “Code” parameter.

Returns: `TRUE` on success, or `FALSE` on failure.

See also: `pan_GetHandle`, `pan_SetValue`

pan_SetPos Move the viewport to an absolute position

`pan_SetPos` pans, tilts or zooms the current viewport to a new position (regardless of the current position).

Syntax: `WORD pan_SetPos(PANINFO Pan, int X, int Y, int Z)`

`WORD PanView::SetPos(int X, int Y, int Z)`

Pan The panorama handle.

X The horizontal position, measured in pixels from the left edge of the source image.

- Y** The vertical position, measured in pixels from the top edge of the source image.
- Z** The current “distance” from the panoramic screen, measure as the distance in pixels from the centre of the cylinder. A positive value is towards the screen.

Returns: A series of flags that indicate whether the position was blocked (invalid). It is a combination of the flags `PAN_BOUND_X`, `PAN_BOUND_Y` and `PAN_BOUND_Z`.

Notes: It usually makes no sense to zoom in with a factor higher than, say, 10. There is no technical limit on zooming in, but if desired, the maximum zoom factor can be set with `pan_SetValue`.

The minimum zoom factor is bound by the size of the source image and the size of the viewport.

See also: `pan_GetValue`, `pan_Move`, `pan_SetValue`.

pan_SetValue Change panorama settings and attributes

Syntax: `BOOL pan_SetValue(PANINFO Pan, int Code, int Value)`

`BOOL PanView::SetValue(int Code, int Value)`

Pan The panorama handle.

Code Specifies the value that must be set. It is one of the following:

PAN_VALUE_ID

The “ID” of the panorama. The ID of a panorama is passed with the notification messages. It is not used in any other way by the Panodome functions.

PAN_VALUE_MAXZOOM

The maximum zoom factor that is allowed; when zero (the default), there is no maximum.

PAN_VALUE_MSGNUM

The numeric value of the notification message that the viewer window sends. The default message value is `PAN_NOTIFY`.

PAN_VALUE_ORIENTATION

The orientation (the compass position) in degrees. This value is related to the horizontal position.

PAN_VALUE_SPEED

The timer interval in milliseconds at which the viewer window refreshes. The default is 50 ms, which gives 20 frames per second. When set to zero, the viewer window does not create a timer.

PAN_VALUE_XPOS

The current “horizontal position” in the panorama, measured as the offset in pixels from the left edge of the source image.

PAN_VALUE_YPOS

The current “vertical position” in the panorama, measured as the offset in pixels from top edge of the source image.

PAN_VALUE_ZPOS

The current “distance” from the panoramic screen, measure as the distance in pixels from the centre of the cylinder.

Value The new value of the “Code” parameter.

Returns: **TRUE** on success, or **FALSE** on failure.

See also: **pan_GetValue**, **pan_SetHandle**, **pan_SetPos**

Data reference

Definition of various data structures used by the Panodome functions.

PANATTRIB Panorama attributes

```
typedef struct {
    int angle;
    int horizon;
    int orientation;
    int numhotspots;
    LPPANHOTSPOT hotspots;
    char userstring[128];
    int flags;
    long reserved;
} PANATTRIB, FAR *LPPANATTRIB;
```

angle The “field of view” angle of the panorama. It is 360 for a full cylinder.

horizon The horizon correction, specified as the offset in pixels from the vertical centre. See also page 6

orientation The compass orientation of the left edge of the source image in degrees counter-clockwise (zero is North).

numhotspots The number of entries in the array that the “hotspots” field points to.

hotspots Pointer to an array with hot spot information.

userstring This field is unused by the Panodome toolkit; it may contain any kind of information. Usually, the contents of this field are read from the attribute file, see page 7.

flags Used for internal purposes; when filling this structure in manually, set this field to zero.

reserved Reserved for future expansion; set to zero.

PANHOTSPOT Hot spot data

```
typedef struct {
    int left, top, right, bottom;
    char label[128];
    char text[128];
} PANHOTSPOT, FAR *LPPANHOTSPOT;
```

left, top

right, bottom

The bounding rectangle of the hot spot.

label

The label, or action name, for the hot spot. This field often contains the filename of the next panoramic image to open.

text

General purpose text, for example for tooltips or balloon help.

Message reference

PAN_COMMAND Control message for panorama window and compass

A window created with `pan_CreateWindow` or `pan_CreateCompass` handles this message to adjust its status or carry out a command. The code for the setting must be passed in the `wParam` parameter of the `SendMessage` or the `PostMessage` functions.

wParam: Holds the code of setting to adjust:

PC_ORIENTATION

lParam the orientation of the needle in the compass window. It must be a value between 0 and 360.

PC_TIMERTICK

This message lets you replace the built-in timer for the panorama window with one of your own. Each message is considered a “tick” of an external timer, so your application should send `PC_TIMERTICK` messages on a regular interval. When using the `PC_TIMERTICK` messages the internal timer should be shut off by setting it to 0.

lParam: The value of `lParam` depends on the setting (in `wParam`).

Notes: The value of the `PAN_COMMAND` message is `WM_USER+78`.

See also: `pan_CreateCompass`, `pan_CreateWindow`, `pan_SetValue`

PAN_NOTIFY Notification message for panoramas

The window specified in the call to `pan_Create` receives this notification message on various events.

wParam: Holds the “code” of notification:

PN_HALT

The panorama has stopped moving. The low word of `lParam` is zero.

PN_HITHOTSPOT

The mouse cursor “floats” above a hot spot. The low word of `lParam` is the sequence number of the hot spot.

PN_MOVE

The panorama has moved (position or zoom). The low word of `lParam` is zero.

PN_SELHOTSPOT

The user clicked on a hot spot. The low word of `lParam` is the sequence number of the hot spot.

lParam: Holds the associated “data” of the notification. The low word of `lParam` depends on the value of `wParam`. The high word contains the “id” of the panorama. This value can be set with `pan_SetValue` with code `PAN_VALUE_ID`.

Notes: The value of the `PAN_NOTIFY` message is `WM_USER+79`.

See also: `pan_Create`, `pan_SetHandle`

Error codes

PAN_ERR_NONE

No error.

PAN_ERR_ANGLE

Invalid angle in the panorama attribute structure.

PAN_ERR_COORDS

Invalid coordinates.

PAN_ERR_IMGFORMAT

The source image has an unsupported type, it is not an uncompressed 256-colour DIB.

PAN_ERR_IMGSIZE

The width in pixels of the source image is not a multiple of 4.

PAN_ERR_INDEX

The “index” parameter to a function was out of range.

PAN_ERR_MEMORY

Insufficient memory.

PAN_ERR_PARAM

General “invalid parameter” error.

PAN_ERR_VPSIZE

The viewport is too large for the panorama to fit in. Reduce the viewport size or scale the source image.

Handling DIBs

APPENDIX A

Panoramas are attractive because the spectators are immersed in an environment that surrounds them. Panning through a panorama is quite different from sliding a flat picture below a viewport. A photograph is a flat surface that you look at from some distance. A panorama pulls you *inside* the photograph. The constant adjustment of the perspective view of the area that you are looking at, adds realism and indeed gives the sensation that the image is folded around you.

However, in practice, panoramas usually show deserted, static environments: rooms with furniture, but no people; parks with trees and statues, but otherwise empty. (Partly this is due to the difficulty of photographing panoramic scenes with moving objects.) So, as a spectator of a panorama, you are immersed in a silent, still world. This is a bit of a surrealistic experience.

Animation can change all that. One of the design criterions of Panodome was to make the engine flexible enough to combine with a maximum of graphic and animation libraries. Adhering to common standards and formats is an important first step into that direction, and hence Panodome's "image format" is a device independent bitmap (DIB). Other graphic and animation engines from CompuPhase (EGI and AniSprite) also use DIBs as their basic data type, so attaching Panodome to EGI or to AniSprite is quite "natural".

However, I would not want to force you to use only other CompuPhase products just because you work with Panodome. Most other graphic and animation libraries offer some kind of interface to a DIB (the other libraries that I am aware of do not use DIBs as their internal image format, as do Panodome, EGI and AniSprite, but they *do* offer conversion functions to and from DIBs). As you might imagine, however, not all libraries access DIBs in the same way.

A device independent bitmap essentially separates the header from the pixel data. In a Windows Bitmap file (.BMP extension), the pixel data immediately follows the header, but the basic functions from the Windows SDK (e.g. `CreateDIBitmap`) take pointers to the header and the pixel data of a DIB in separate parameters. This has led to two major ways in which graphic libraries deal with DIBs: they either follow the Windows SDK functions and take two pointers to the header and the pixel data, or they use only a single pointer to the header with the assumption that the pixel data immediately follows the header. Panodome is compatible with both DIB handling schemes, and this duality permeates through the engine.

Panodome is both a producer and a consumer of DIBs. As a producer, function `pan_GetView` returns a DIB for the current viewport; a sprite library could use this DIB to draw moving objects onto the panorama. As a consumer, a frame animation engine, like EGI, can also contain a movie of panoramic images and *source* the frames in DIB format to Panodome, which then unwarps the frames and shows the result in its viewport.

A function like `pan_Create` is prototyped as:

```
PANINFO pan_Create(LPBITMAPINFO BitsInfo, LPVOID ImageBits,
                  LPRECT ViewPort, LPPANATTRIB Attribs)
```

where the first parameter points to a DIB header and the second parameter points to the pixel data. Both may be in separate memory blocks. The “header” memory block may be deleted after `pan_Create` returns, but the pixel data needs to stay allocated until the panorama is closed.

However, if the second parameter, `ImageBits`, is `NULL`, the function assumes that the pixel data immediately follows the header. In that case, you may not delete the header memory block after the call to `pan_Create`, because the pixel data resides in the same memory block.

A typical function that returns a DIB is `pan_LoadDIB`. The function is prototyped

```
as: LPVOID pan_LoadDIB(LPCSTR Filename, LPBITMAPINFO BitsInfo)
```

If the second parameter, `BitsInfo`, points to a memory block that can hold a `BITMAPINFO` structure, `pan_LoadDIB` stores the DIB header in that memory block and it returns a pointer to a memory block that holds *only* the pixel data. The header and the pixel data are thus stored in two separate memory blocks, one of which you must have allocated yourself before the call to `pan_LoadDIB`.

If, on the other hand, `BitsInfo` is `NULL`, `pan_LoadDIB` allocates a memory block for both the DIB header and the pixel data and it returns a pointer to the start of that memory block. There is only a single “handle” to the DIB, which may make the manipulation of the DIB easier.

Although these examples discuss `pan_Create` and `pan_LoadDIB`, the concepts apply to all functions in Panodome that take or return pointers to DIBs.

Resources

APPENDIX B

• CompuPhase’s homepage

The CompuPhase homepage contains information on Panodome’s companion products EGI and AniSprite, as well as articles and technical papers on animation from a programmer’s perspective (although nothing “moves” in a panorama, it is generally put in the “animation” category because the real-time adjustments of the perspective).

Please visit us at:

<http://www.compuphase.com>

Relevant topics are:

- ◇ The frame animation engine EGI and the sprite animation engine AniSprite, good companions to Panodome.
- ◇ Details on Windows’ Palette Manager: how to create an identity palette, how to prevent Windows from shuffling your carefully crafted palette, why identity palettes are faster, . . . recommended.
- ◇ A freely available simple and fast scripting language that can be used to make the panorama animations interactive.
- ◇ An assortment of tips and resources for computer animation.

Upon request, we can also send you this information by e-mail (in HTML format) or by fax. Our address is on the inside title page.

• Stitching software

Producers of software packages to create panoramic images for Microsoft Windows environments are:

- ◇ Live Picture produces **PhotoVista** for creating panoramas from photographs:
<http://www.livepicture.com/>.
- ◇ PictureWorks makes **SpinPanorama** to create panoramas from photographs:
<http://www.kaidan.com/>.
- ◇ **VideoBrush Panorama** makes panoramas from a digitized video:
<http://www.videobrush.com/>.
- ◇ Wasabi offers a plug-in for Photoshop (and Paint Shop Pro), called **SkyPaint**, that helps in retouching panoramas, or creating them by hand:
<http://www.wasabisoft.com/>.

• Lenses and tripod heads

Several manufacturers offers equipment to make the photographing for panoramas a little faster or more convenient:

- ◇ CycloVision's **ParaShot** is a parabolic mirror that can snap a 360° image at once:
<http://www.cyclovision.com/>.
- ◇ Be Here Corporation offers a similar panoramic "lens":
<http://www.behere.com/>.
- ◇ Kaidan has an assortment of tripod heads that can adjust the rotation point below the camera lens and that offer radial scales and bubble levels:
<http://www.kaidan.com/>.

Index

- AniSprite, 38
- BITMAPINFO, 11
- BMP files, 7, 10, 25
- Borland C++, 10
- C language, 10
- C++ language, 15
- Case sensitive, 8
- Compass
 - orientation, 8, 13, 24, 32, 33
 - size, 18
 - window, 18
- Compuserve GIF, 7
- CreateDIBitmap, 38
- Cubical panoramas, 2
- Cylindrical panoramas, 2
- Device Independent Bitmap, 38
- DIB format, 10, 11, 38
- EGI, 38
- Event, *see also* Notify message, 12, 13, 19
- Field of view, 5, 8, 33
- Focal length, 5
- Focus (input), 12
- Frame size, 5
- Getting lost, 13
- GIF files, 7
- Horizon adjustment, 6
- Hot spot, 8, 12, 16, 21, 23, 24, 33
 - label, 12
- Input focus, 12
- Microsoft Visual C/C++, 10
- Navigation window, *See* Viewer window
- Needle, 18
- Notify message, 12, 19, 22, 31
- PAN files, 9
- pan_AllocResource, 15
- pan_AttribFile, 7, 8, 15
- pan_AttribString, 7, 8, 16
- pan_CheckHotSpot, 16
- PAN_COMMAND, 18, 35
- pan_Create, 17, 39
- pan_CreateCompass, 13, 14, 18, 35
- pan_CreateWindow, 11, 18, 35
- pan_Delete, 19
- pan_Error, 20
- pan_FreeResource, 20
- pan_GetData, 12, 21
- pan_GetHandle, 22
- pan_GetRect, 22
- pan_GetValue, 23
- pan_GetView, 25
- pan_LoadDIB, 11, 25, 39
- pan_MapCoordinates, 26
- pan_MinImageSize, 27
- pan_Move, 27
- PAN_NOTIFY, 12, 24, 31, 35
- pan_Replace, 28
- pan_ScaleDIB, 29

`pan_SetHandle`, 12, 30

`pan_SetPos`, 30

`pan_SetValue`, 12, 31

Panorama

 cubical, 2

 cylindrical, 2

 horizon adjustment, 6

 id, 31

 spherical, 2

`PC_ORIENTATION`, 35

`PC_TIMERTICK`, 35

PCX files, 7

`PN_HALT`, 35

`PN_HITHOTSPOT`, 36

`PN_MOVE`, 36

`PN_SELHOTSPOT`, 36

PPM files, 7, 9

Radial scale, 5

Rectilinear images, 5

RLE compression, 10, 25

Speed, 32

Spherical panoramas, 2

Surrealism, 38

Timer interval, 32

Tripod, 5

 panoramic heads, 5

User data, 8

Viewer window, 11–13, 17, 18, 30, 32

Watcom C/C++, 10

White space, 9

Windows Bitmap file format, *See*
 BMP files

Zoom factor

 maximum, 24, 28, 31

 minimum, 28

ZSoft PCX, 7