VCET API Manual

Viewing and Conversion Enabling Technology

VCET API Manual

Contents

Introduction

Introduction

If your application requires file viewing, printing or conversion, then you need CSI's Multi-format Controls. They provide the most effective way to get your MS-Windows product to market sooner and at lower cost.

Your development team will be able to focus on the functionality of your application, rather than the time consuming (and expensive) coding, testing and debugging needed to support an ever-changing landscape of file types and formats.

CSI has Multi-format Controls to meet all your file viewing, printing and conversion needs, with support for a large number of file formats in the following categories:

- ❒ Word Processors
- ❒ Spreadsheets
- ❒ Databases
- ❒ Raster Graphics
- ❒ Vector Graphics
- ❒ Compressed Files

Their efficient high-level interface makes these controls easy to integrate into your product. They allow your application to:

- ❒ Perform text search and highlighting.
- ❒ View all or part of a file, pan and zoom.
- ❒ Print all or part of a file, with many options.
- ❒ Convert files, with many options.
- ❒ Copy all or part of a file to the clipboard.
- ❒ And more...

And CSI's Multi-format Controls do not rely on filename extensions to determine file type, but rather on file structure. For example, if you name a raster graphics file MY_FILE.ABC, the control will recognize the file's format (BMP, PCX, TIFF, GIF, etc.) and treat it accordingly.

CSI's Multi-format Controls can make use of our extensive range of import and export filters, which we are continually updating and expanding. We also provide the tools to add additional file formats. And since new filters are automatically recognized, you won't have to recompile your source code to use them.

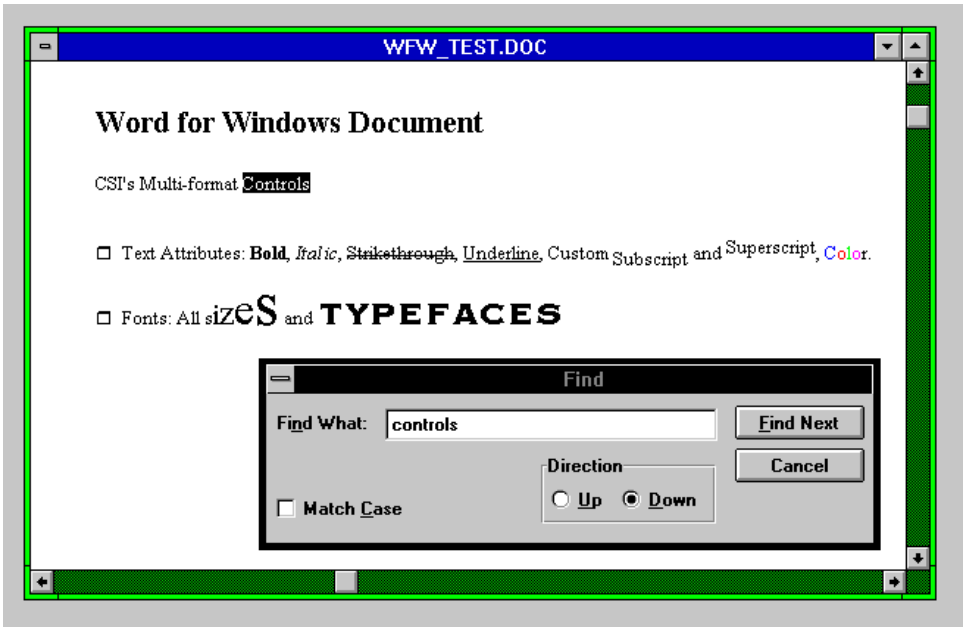
Possible applications include:

- ❒ File viewing utilities.
- ❒ Document management software. Software that manages a large variety of files, provides viewing ability, and possibly indexing.
- ❒ Fax related software that requires the conversion of various files to a common fax format.
- ❒ E-Mail related software, requiring that a large variety of files be distributed electronically and accessed by many users.
- ❒ Other possible areas:
  - CD-ROM
  - Multimedia
  - Peripheral Manufacturers, i.e., printers, scanners, fax cards, etc.

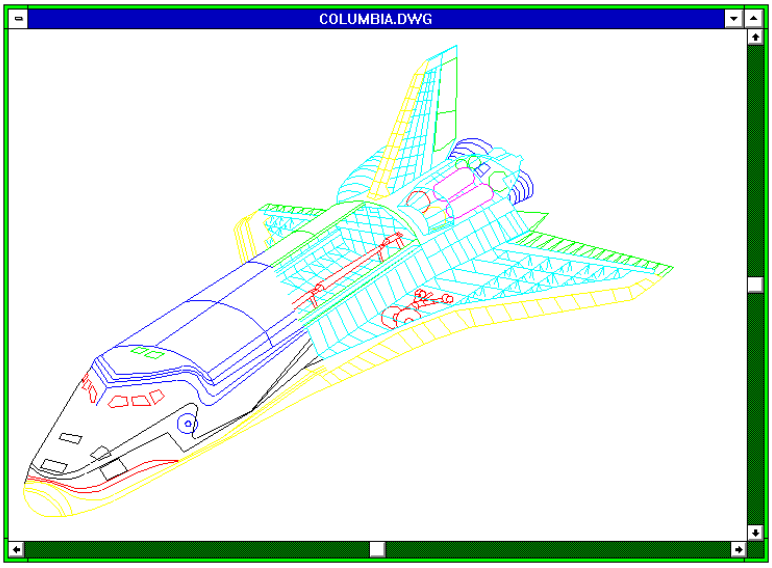
About this Document

This document is designed to give the programmer a quick start with the **CSI Multi-format Controls API**. It contains detailed descriptions of the various functions and messages, as well as summaries of these in order to facilitate lookup.

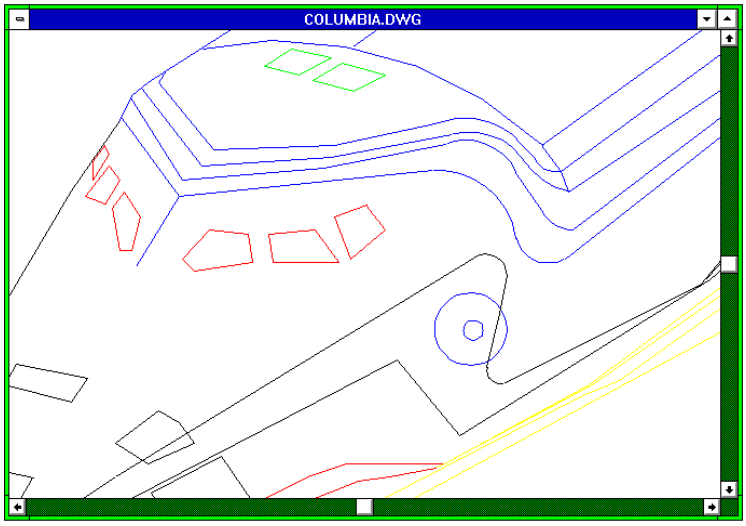
Here are some of the things that you can do with the controls.



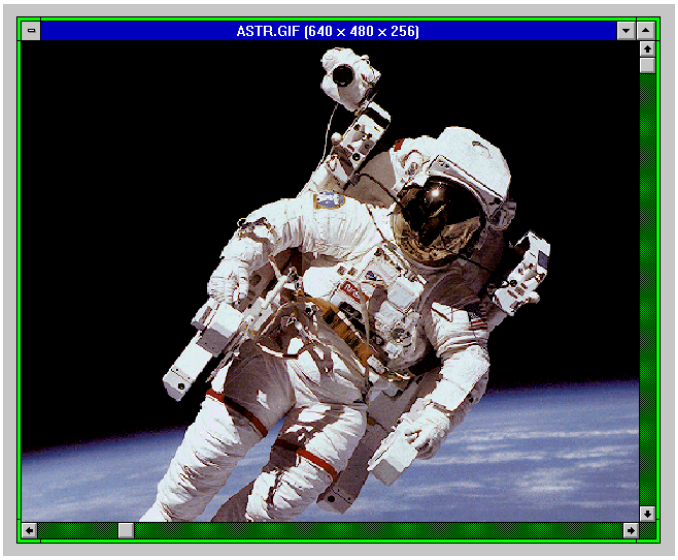
Perform a text search in a word processor file.



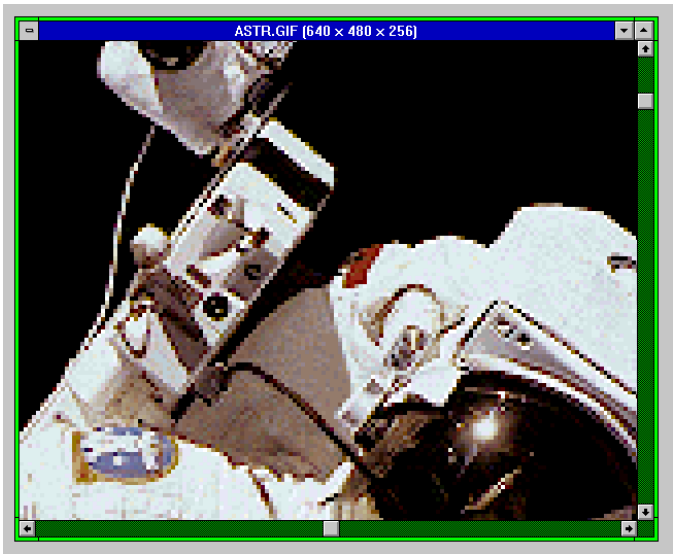
Display CAD files normally...



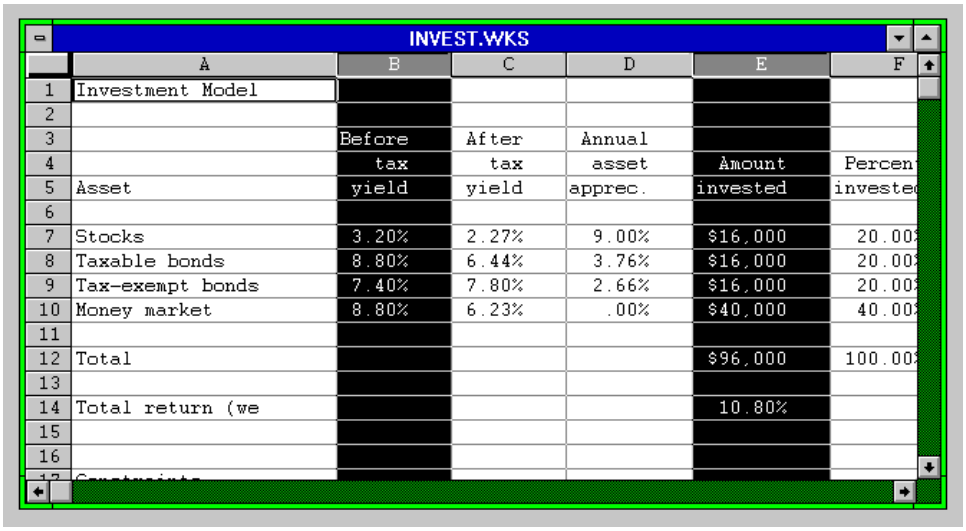
... or zoom in or out.



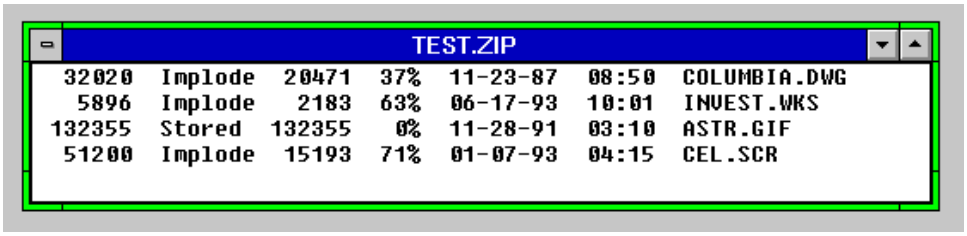
Display bitmaps normally...



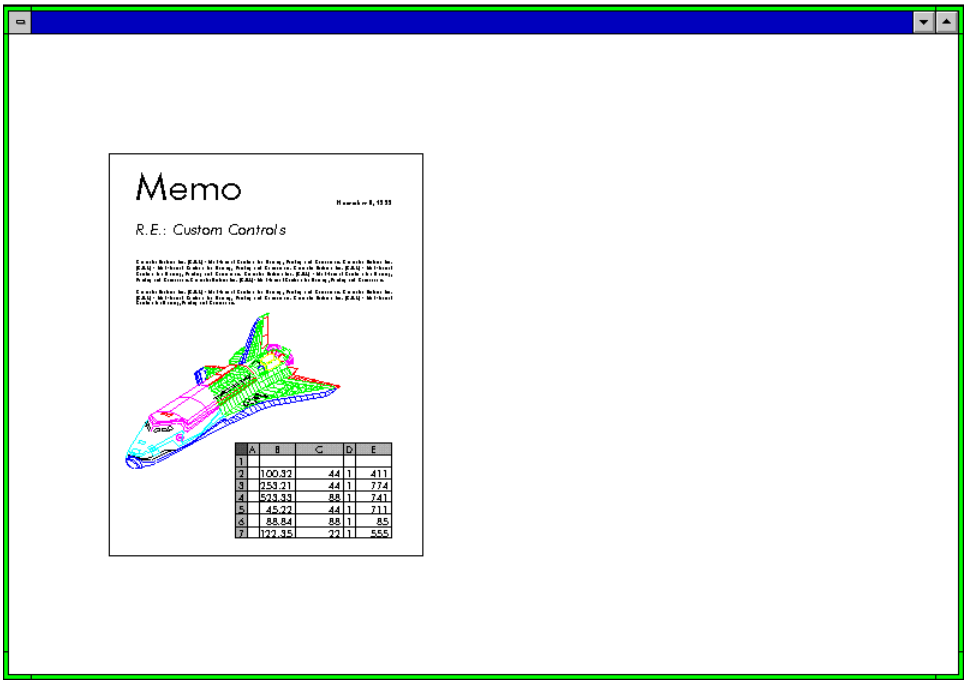
... or zoom in or out.



Select areas in a spreadsheet or database and copy to the clipboard.



View the contents of archives.



Several controls rendering their contents to a display context for viewing or printing.

Overview

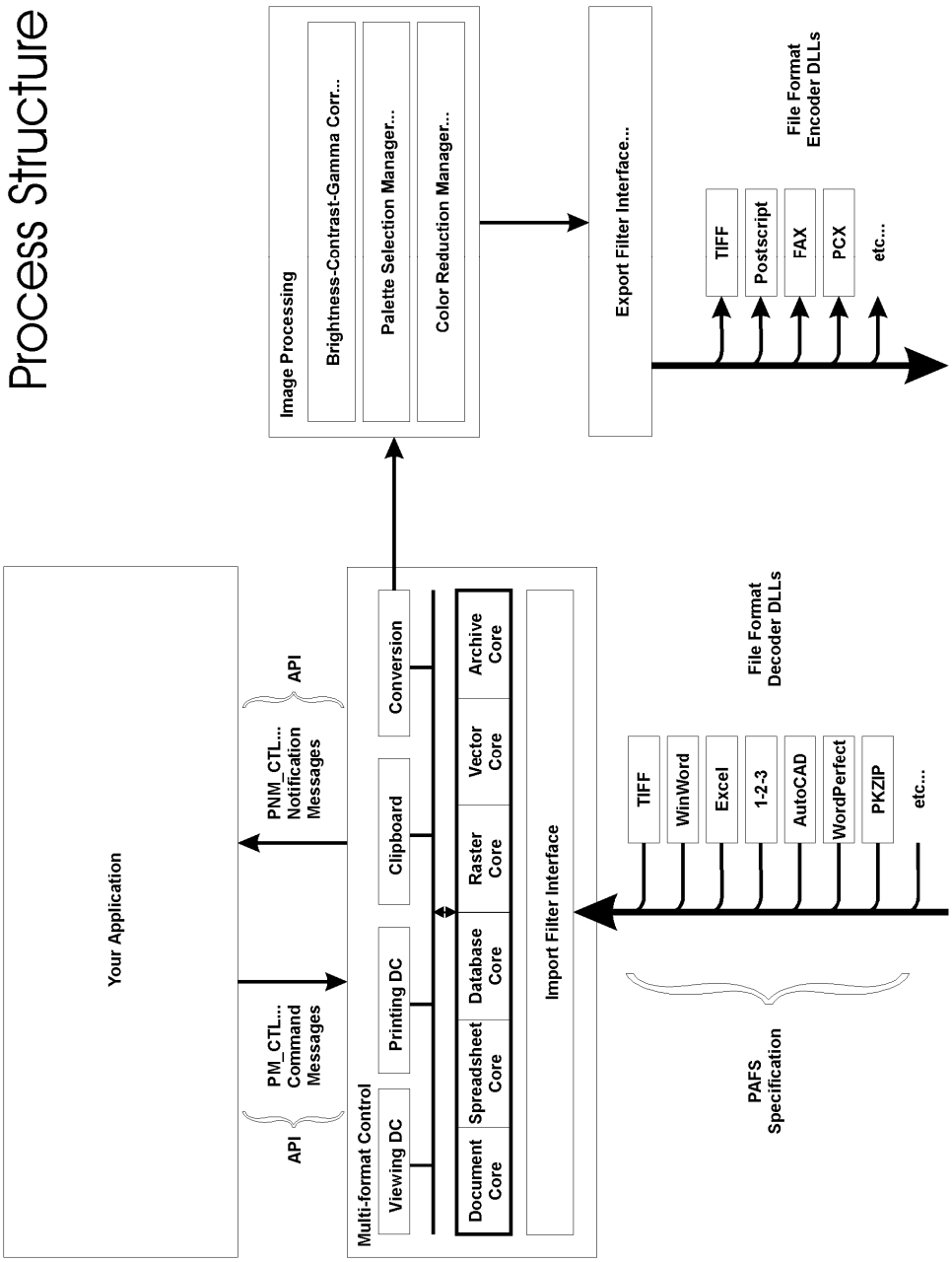
Overview

CSI's Multi-format Controls enable an application to easily display, print or convert the contents of a given file. There are six types of controls: raster, vector, document, database, spreadsheet, and archive, each capable of handling a specific class of file formats. Depending on how a control is created, it may or may not directly manage resize, scroll, mouse, and keyboard events. Once a control has been created, an application communicates with it using the handle of the control window and a set of messages by calling the Windows functions **SendMessage()** and **PostMessage().**

There are control-specific messages to perform such tasks as obtaining information about a control, enabling and disabling certain capabilities of a control, making a copy of a control, displaying given portions of a file in a control window, copying given portions of a file to the clipboard, searching for a string in a given file and highlighting the result, converting from client to world coordinates and vice versa, printing given portions of a file, etc. There are also messages to render given portions of a file onto a device context or metafile.

Some newer messages include C^{++} classes such as STL containers, Boost smart pointers and VCET interfaces. Whenever these messages are used, the application should have access to the C^{++} Standard Template Library and to the Boost C^{++} Libraries version 1.34.1 (may need to be installed separately).

The process structure is outlined in the following pages.



Please refer to the diagram on the preceding page for the discussion that follows:

A single function call will provide your application with a control it can interact with via a powerful set of command and notification messages. These messages make up the bulk of the Application Programming Interface (API) for the controls. The remainder consists of the control creation function mentioned above, as well as a function to retrieve the last error code.

The control itself is made up of six cores, one for each type of file (word processor, spreadsheet, database, raster graphic, vector graphic, and archive). Each core has viewing, printing, clipboard transfer, and conversion capabilities. Which of the cores is active is determined by the type of file being processed.

The controls read in files through the Import Filter Interface. Import filters, i.e., the File Format Decoders, are individual Dynamic Link Libraries (DLLs), each one specific to a particular file format.

We have included the specifications for this interface within these documents. These specifications allow you to write your own File Format Decoders, which are also compatible with our retail product. Please refer to the PAFS (PANORAMIC! Additional Format Support) section of this document for more information.

The conversion process includes the ability to perform image processing prior to conversion. Capabilities include, but are not limited to, brightness adjustment, contrast adjustment, and gamma correction. A Palette Selection Manager provides for several palette selection methods (expandable).
The number of color reduction methods is also expandable via the Color Reduction Manager.

The conversion itself is achieved through the Export Filter Interface, which makes use of the various File Format Encoder DLLs at its disposal.

Functions

Function Summary

Function Name	Purpose
PAN_CreateControl()	Creates a CSI Multi-format Control.
PAN_FreeControls()	Unloads the CSI Multi-format Control DLLs.
PAN_GetCtlErrorCode()	Returns the error code of the last error.
PAN_LoadControls()	Loads the CSI Multi-format Control DLLs.

Function Descriptions

This section provides detailed information on all of the available functions. It includes the function's purpose, syntax, description, what parameters are expected and what values are returned.

Remarks

An application using the CSI Multi-format Controls should call **PAN_LoadControls()** once at the start of the execution of the program, and make a single call to **PAN_FreeControls()** when the program is terminating. While executing, the application uses **PAN_CreateControl()** to create controls, and sends **PM_CTLDESTROY** messages followed by **DestroyWindow** (with the handle of the control window as a parameter) calls to remove controls.

A control is not visible immediately after being created, therefore an application can perform any initialization operations on the control before making it visible. To make the control visible, the Windows function **ShowWindow()** is called with the handle of the control window as the first parameter.

A control must be destroyed using the Windows function **DestroyWindow()** with the handle of the control window as the parameter, but only after it has been given a chance to clean up its internal data structures through the sending of a **PM_CTLDESTROY** message.

PAN_LoadControls()

Purpose Loads the CSI Multi-format Control DLLs.

Syntax **int PAN_LoadControls(LPCSTR** *szInifile*
DWORD *dwReserved***);**

Description The facilities of the CSI Multi-format Controls are provided by a number of dynamically linked libraries. This function, called by the application program, ensures that all the DLLs are made available and are initialized correctly.

Parameters *szInifile* specifies the initialization file which the controls will use to store and retrieve private information. If **NULL**, the controls will use pctl.ini.

dwReserved is a special variable which must be set to zero (0).

Returns This function returns **TRUE** if the libraries were successfully loaded, **FALSE** otherwise..

Example This code fragment loads the controls.

```
#include "pctl.h"

BOOL   LoadOk;          // Boolean indicating if Multi-format
                        // Controls loaded successfully.

int PASCAL WinMain(HINSTANCE hInst, ...)
{
        // Load controls.
        bLoadOk = PAN_LoadControls();

        if (!bLoadOk) {return (0);}

        // Main message loop.
}
```

PAN_CreateControl()

Purpose Creates a CSI Multi-format Control.

Syntax

```
HWND PAN_CreateControl(
                HWND        hwnd,
                WORD        ctlID,
                LPRECT      ctlRect,
                FARPROC     ctlConsumeProc,
                WNDPROC     ctlNotifyProc);
```

Description

Parameters *hwnd* Window handle.

If *hwnd* is **NULL**, *ctlID* is ignored and a stand-alone window of the given size is created (see *CtlRect*). Otherwise, if *ctlID* is greater than zero, a child window of the given size (see *CtlRect* is created). The control manages all resize, scroll, mouse, and keyboard events unless instructed otherwise. See description for **PM_CTLGETCAPS** and **PM_CTLSETCAPS**.

Whether or not a control manages resize, scroll, mouse, and keyboard events directly, the API provides control-specific messages to perform such operations.

ctlID Control ID. Must be unique for each child control of a given application parent window. See description for *hwnd*.

ctlRect Control window rectangle. Specifies origin and extents (left, top, right, bottom) in client coordinates for child windows and in screen coordinates for stand-alone windows.

ctlConsumeProc Pointer to procedure.

If *ctlConsumeProc* is not **NULL**, it must be a pointer, obtained using the Windows function **MakeProcInstance(),** to a procedure which will be called by the control during processing in order to yield to other Windows tasks. Within this procedure, the calling application should call the Windows function **PeekMessage()** to remove events from the Windows message queue and then translate/dispatch those messages using the Windows functions **TranslateMessage()** and **DispatchMessage().** The application can also handle any accelerator key messages by calling the appropriate Windows functions.

ctlNotifyProc Pointer to window procedure.

If *ctlNotifyProc* is not **NULL**, it must be a pointer obtained using the Windows function **MakeProcInstance()** to a window procedure which will receive all control notification messages. The **HWND** parameter contains the handle of the control window sending the message and the **UINT** parameter contains the notification message. The **WORD** and **LONG** parameters are used to send message-dependent information.

Returns This function returns a handle to the control window on success or **NULL** on failure.

Example This code fragment creates a control, sets it up, and then displays it.

```
#include "pctl.h"

HWND  hControlWnd;              //Stores the handle to the control.
RECT    CtlRect;                // Size of the control.


SetRect( &CtlRect, 0,0,400,400);

// Create a basic control in a stand-alone window.
hControlWnd=PAN_CreateControl(
          NULL, 1, &CtlRect, NULL, NULL);

// setup the control.
// …

// Show the control.
ShowWindow( hControlWnd, SW_SHOWNA);
```

PAN_FreeControls()

Purpose Unloads the CSI Multi-format Control DLLs.

Syntax **int PAN_FreeControls(***void***);**

Description Removes the CSI Multi-format Controls from memory. Called by the application before it terminates.

Returns This function returns **TRUE** if the libraries were unloaded, **FALSE** if the data structures required to free them were unavailable..

Example This code fragment demonstrates the unloading of the controls, .

```
#include "pctl.h"

BOOL   bLoadOk;          // Boolean indicating if Multi-format Controls
       // loaded successfully.


int PASCAL WinMain(HINSTANCE hInst, ...)
{

       // Load controls.
       bLoadOk = PAN_LoadControls();

       if (!bLoadOk) {
              return (0);
       }

       // Main message loop.
       while (GetMessage(&msg, NULL, 0, 0)) {
              ...
              TranslateMessage(&msg);
              DispatchMessage(&msg);
       }

       PAN_UnloadControls();
}
```

PAN_GetCtlErrorCode()

Purpose Returns the error code of the last error.

Syntax **int PAN_GetCtlErrorCode(***void***);**

Description All control-specific error codes begin with the prefix **PAN_CTLERR**.

Parameters None.

Returns Error code of last error (see list below).

Example This code fragment verifies that no error occurred on the last operation.

```
int     ErrCode;

ErrCode = PAN_GetCtlErrorCode();

if (ErrCode == PAN_CTLERRNONE)
        MessageBox( HWindow, "no errors", "Error Code", MB_OK);
```

Error codes

PAN_CTLERRALREADYINSTALLED
"Control library already installed."
INTERNAL ERROR: The control library cannot continue because it is already installed.

PAN_CTLERRBADCAPS
"Bad control capability."
The control library encountered an invalid capability.

PAN_CTLERRBADCLPBRDFMT
"Bad Clipboard Format."
The parent application has supplied an invalid clipboard format.

PAN_CTLERRBADCTLTYPE
"Bad control type."
INTERNAL ERROR: The control library encountered an invalid format class.

PAN_CTLERRBADDATA
"Bad data."
INTERNAL ERROR: The control library encountered invalid data.

PAN_CTLERRBADINSTANCE
"Bad instance handle."
INTERNAL ERROR: The control library encountered an invalid instance handle.

PAN_CTLERRBADLMBACTION
"Bad Mouse Action."
The parent application has supplied an invalid mouse action.

PAN_CTLERRBADLPARAM
"Bad **LONG** message parameter."
The last API message invoked by the parent application was passed an invalid **LONG** parameter.

PAN_CTLERRBADMESSAGE
"Bad control message."
The control library encountered an invalid message.

PAN_CTLERRBADMODE
"Bad control mode."
The parent application has supplied an invalid control mode (**PAN_CTLMODE**...) to an API function or message.

PAN_CTLERRBADPARAM
"Bad function parameter."
The last API function or message invoked by the parent application was passed an invalid parameter.

PAN_CTLERRBADWINDOW
"Bad window handle."
The control library encountered an invalid window handle.

PAN_CTLERRBADWPARAM
"Bad **WORD** message parameter."
The last API message invoked by the parent application was passed an invalid **WORD** parameter.

PAN_CTLERRCANNOTADDCTLDATA
"Cannot add control data."
INTERNAL ERROR: The control library encountered an error while initializing control-specific data.

PAN_CTLERRCANNOTALLOCMEM
"Cannot allocate memory."
INTERNAL ERROR: The control library encountered an error while allocating memory with the Windows **GlobalAlloc** function.

PAN_CTLERRCANNOTBEGINFILE
"Cannot Begin file."
INTERNAL ERROR: A decoder DLL encountered an error while initializing prior to processing a portion of a file.

PAN_CTLERRCANNOTCONFIGURETABLE
"Cannot configure table."
INTERNAL ERROR: The control library encountered a problem while configuring a data table.

PAN_CTLERRCANNOTCREATECONTROL
"Cannot create control."
INTERNAL ERROR: The control library encountered a problem while creating a control.

PAN_CTLERRCANNOTCREATECTLDATA
"Cannot create control data."
INTERNAL ERROR: The control library cannot create control-specific data.

PAN_CTLERRCANNOTCREATEDC
"Cannot create device context."
INTERNAL ERROR: The control library encountered an error while creating a device context with the Windows **CreateDC** function.

PAN_CTLERRCANNOTCREATEMETAFILE
"Cannot Create Metafile."
INTERNAL ERROR: The control library encountered a problem while creating a metafile with the Windows **CreateMetafile** function.

PAN_CTLERRCANNOTCREATEOBJECT
"Cannot create GDI object."
INTERNAL ERROR: The control library encountered an error while creating a Windows GDI object.

PAN_CTLERRCANNOTCREATEPALETTE
"Cannot create palette."
INTERNAL ERROR: The control library encountered an error while creating a Windows palette.

PAN_CTLERRCANNOTCREATEWINDOW
"Cannot create window."
INTERNAL ERROR: The control library encountered a problem while creating a window with the Windows **CreateWindow** function.

PAN_CTLERRCANNOTENDFILE
"Cannot End file."
INTERNAL ERROR: A decoder DLL encountered an error while cleaning up after processing a portion of a file.

PAN_CTLERRCANNOTFINDCTLDATA
"Cannot find control data."
INTERNAL ERROR: The control library cannot find control-specific data.

PAN_CTLERRCANNOTFINDDLL
"Cannot find support DLL."
INTERNAL ERROR: The control library cannot find one of its support DLLs.

PAN_CTLERRCANNOTGETDC
"Cannot get device context."
INTERNAL ERROR: The control library cannot obtain the handle of a device context using the Windows **GetDC** or **BeginPaint** functions.

PAN_CTLERRCANNOTGETPROCADDRESS
"Cannot get procedure address."
INTERNAL ERROR: The control library encountered a problem while fetching the address of a DLL procedure with the Windows **GetProcAddress** function.

PAN_CTLERRCANNOTINITTEXT
“Cannot initialize text”
INTERNAL ERROR: The control library encaountered an error while trying to initialize text support.

PAN_CTLERRCANNOTLOADDLL
"Cannot load support DLL."

INTERNAL ERROR: The control library encountered a problem while loading one of its support DLLs.

PAN_CTLERRCANNOTLOCKCTLDATA
"Cannot lock control data."
INTERNAL ERROR: The control library cannot lock control-specific data with the Windows **GlobalLock** function.

PAN_CTLERRCANNOTLOCKMEM
"Cannot lock memory."
INTERNAL ERROR: The control library encountered a problem while locking memory with the Windows **GlobalLock** function.

PAN_CTLERRCANNOTMAKEPROCINSTANCE
"Cannot make procedure instance."
INTERNAL ERROR: The control library encountered a problem while making a procedure instance with the Windows **MakeProcInstance** function.

PAN_CTLERRCANNOTOPENCLIPBOARD
"Cannot Open Clipboard."
INTERNAL ERROR: The control library is unable to open the clipboard

PAN_CTLERRCANNOTPROCESSFILE
"Cannot Process file."
INTERNAL ERROR: A decoder DLL encountered an error while processing a file.

PAN_CTLERRCANNOTQUERYFILE
"Cannot Query file."
INTERNAL ERROR: A decoder DLL encountered an error while initializing prior to processing a file.

PAN_CTLERRCANNOTREGISTERCLASS
"Cannot register window class."
INTERNAL ERROR: The control library encountered a problem while registering a window class with the Windows **RegisterClass** function.

PAN_CTLERRCANNOTTERMINATEFILE
"Cannot Terminate file."
INTERNAL ERROR: A decoder DLL encountered an error while cleaning up after processing a file.

PAN_CTLERRDLLFAILED
"Support DLL failed."
INTERNAL ERROR: One of the support DLLs of the control library encountered an error and cannot continue.

PAN_CTLERRINTERNAL
“Internal error”
INTERNAL ERROR: The control library encountered an error during processing.

PAN_CTLERRMISC
"Undefined error."
An undefined error has occurred.

PAN_CTLERRNOBLOCK
"No blocks found"
The parent application requested a block to be displayed, but no blocks exist for the current page.

PAN_CTLERRNOFILESET
"No file set."
The last API function or message invoked by the parent application required a file to have been set, but none was found.

PAN_CTLERRNOLAYERS
"No layers found"
The parent application requested a layer state to be set, but no layers exist for the page.

PAN_CTLERRNONE
"No error."
No error has occurred.

PAN_CTLERRNOSELSET
"No Selection Set."
There is nothing selected in the control.

PAN_CTLERRNOTCOMPATIBLE
"Function or message not compatible."
The last API function or message invoked by the parent application is not compatible with the format class of the currently loaded file.

PAN_CTLERRNOTIMPLEMENTED
"Function or message not yet implemented."
The last API function or message invoked by the parent application is not yet implemented.

PAN_CTLERRNOTINSTALLED
"Control library not installed."
INTERNAL ERROR: The control library cannot continue because it was not successfully installed.

PAN_CTLERRNOTSUPPORTED
"Control type not supported."
The control library encountered an unsupported format class.

PAN_CTLERRNOVIEW
"No views found"
The parent application attempted to get or set a view, but no views exist for the current page.

PAN_CTLERRNOXREFS
"No XRefs found"
The parent application attempted to set the XRef state, but no XRefs exist for the current page.

Messages

All control-specific messages begin with either the prefix **PM_CTL** for command messages, or **PNM_CTL** for notification messages, and use the Windows message calling convention. All command messages return an indication of success or failure, except where noted. The function **PAN_GetCtlErrorCode()** can also be used to obtain the error code of the last error.

Structures and Unions used by Messages

The use of coordinates has been considerably changed from previous versions of the controls. Under version 1.2, all view coordinates are in units that can be used for display and rendering purposes. In previous versions, textual classes used coordinates that were character offset based, which could not easily be used for viewing and printing operations. In version 1.2, all view coordinates for textual classes (archive, database, document, spreadsheet) are in terms of **TWIPS** (twentieth of a point). Raster units remain in pixels, while the vector control retains the use of arbitrary units.

Textual operations that work in terms of a text stream, such as searching, are provided with the new caret based coordinate system. This is similar to the “world” coordinates for document files provided in previous versions of the controls, but has the addition of a “flow” identifier. This field is used to distinguish between the different text flows that can be displayed on the same page in version 1.2.

Converting coordinates between the two coordinate systems can be performed using the new **PM_CTLCARETTOWORLD** and **PM_CTLWORLDTOCARET** messages.

A third type of coordinate exists for tabular format classes (archive, database, spreadsheet) that is specified in terms of rows and columns. No operations explicitly exist that use these coordinates, as they can be performed in either of the previously described coordinate systems. A single message, **PM_CTLGETDIMS,** is available to determine the dimensions of a control in terms of rows and columns.

Structure Summary

Structure Name	Used by
struct PAN_BOOKMARK (Multi-Byte) and class CPanBookmark (Unicode)	PAN_CTLGETBOOKMARKS
struct PAN_BLOCK	PM_CTLGETBLOCKNAMES messagePAN_CTLGETBLOCKNAMES
struct PAN_CtlCaretPos	PAN_CtlSearchInfo structure
	PM_CARETTOWORLD message
	PM_WORLDTOCARET message
struct PAN_CtlClpbrdFmt	PAN_CtlClpbrdFmtList structure
struct PAN_CtlClpbrdFmtList	PM_CTLGETCLPBRDFMTS message
struct PAN_CtlDimensions	PM_CTLGETDIMS message
struct PAN_CtlEntityInfo	PM_CTLGETENTITY message
	PAN_CtlGetEntityInfo structure
struct PAN_CtlFileFmt	PAN_CtlFileFmtList structure
struct PAN_CtlFileFmtList	PM_CTLGETFILEFMTS message
struct PAN_CtlFileInfo	PM_CTLGETFILE message
struct PAN_CtlGetEntityInfo	PM_CTLGETENTITY message
struct PAN_CtlHandle	PM_CTLGETENTITY message

	PAN_CtlSearchInfo structure
	PAN_CtlEntity structure
	PAN_CtlGetEntityInfo structure
struct PAN_CtlInfo	PM_CTLGETINFO message
struct PAN_CtlPos	PAN_CtlFileInfo structure
	PAN_CtlPrintOptions
	PM_CTLSEARCH message
	PM_CTLCLIENTTOWORLD message
	PM_CTLWORLDTOCLIENT message
	PM_GETOFFSET message
	PM_SETOFFSET message
struct PAN_CtlPrintOptions	PM_CTLVALIDATEMARGINS message
	PM_CTLPRINTPREVIEW message
	PM_CTLPRINT message
struct PAN_CtlPrintPreview	PAN_CtlPrintOptions structure
struct PAN_CtlRange	PAN_CtlSel structure
	PM_CTLGETPAGESIZE message
	PM_CTLSETSEL message
	PM_CTLGETVIEWEXTENTS message
	PM_CTLSETVIEWEXTENTS message
	PM_XFRMRECT message
	PNM_CTLSETVIEWEXTENTS notification message
struct PAN_CtlRenderOptions	PM_CTLRENDERONTODC message
struct PAN_CtlSearchInfo (Multi-Byte) and CPanTextSearch / CPanTextSearchOptions (Unicode)	PM_CTLSEARCH message
struct PAN_CtlSel	PAN_CtlSelList structure
struct PAN_CtlSelList	PM_CTLGETSELS message
struct PAN_CtlShowEntity	PM_CTLSHOWENTITY message
struct PAN_LAYER	PM_CTLGETLAYERSTATE message PM_CTLSETLAYERSTATE message
class IResourceContextInfo	PM_CTLSETFILEEX message
class IPanResourceInfo	PM_CTLSETFILEEX message
function ResourceLocateProc	PM_CTLSETFILEEX message
struct PanUserCallback	PM_CTLSETFILEEX message
struct PAN_CtlSetFile	PM_CTLSETFILEEX message
struct PAN_XREF (Multi-Byte) and class CPanXRef (Unicode)	PM_CTLGETXREFSTATE and PM_CTLSETXREFSTATE messages
struct PAN_RESOURCEINFO (Multi-Byte) and struct ResourceInfo (Unicode)	PM_CTLGETRESOURCEINFOSTATE
class IGetResourceInfo	PM_CTLGETRESOURCEINFOSTATE
class IGetString	PM_CTLGETSTRING

Structure Descriptions

```
typedef struct PAN_BOOKMARK {          // Bookmark  information.
unsigned int ID;            // OUT: Any application defined ID (It depends on
                            // the formats).
int level;                  // OUT: Depth level of this bookmark tree (0 := root)
int fState;                 //OUT: State of the bookmark node in bookmark tree.
                            // (0:= closed, 1:= expanded, 2:= leaf)
wchar_t name[_Max_PATH]; //OUT: Descriptive name of bookmark node.
char target[1024];          // OUT: Target destination of this bookmark holding
                            // bookmark commands. We support many types of
                            // commands in the target destination of a bookmark
                            // structure. This is the complete list of them:

                            // Switch to another page PAGE=<index>
                            // <index> - Page index (start from 1)
                            // E.g.:   "PAGE=2"

                            //Link to external file      DOC=<Doc name>
                            // <Doc name> - file name, the application will launch // a new child frame for
                            this document.
                            // E.g.:   Local file: "DOC=file.dwg"
                            //         URL: "DOC=\"http://www.cimmetry.com/file.dwg\""
                            //            or   "DOC=http://www.cimmetry.com/file.dwg"
                            //                        (without quotes)

                            // Set offset        POS=<x>,<y>
                            // <x> - double value for x offset
                            // <y> - double value for y offset
                            // E.g.:   "POS=10.0,15.0"

                            // Set view extents          RANGE=<minX>,<minY>,<maxX>,<maxY>
                            // <minX> - min. X of view extents
                            // <minY> - min. Y of view extents
                            // <maxX> - max. X of view extents
                            // <maxY> - max. Y of view extents
                            E.g.:      "RANGE=-2.0,0.0,10.0,12.0"

                            // Set rotation     ROTATE=<rotDeg>
                            // <rotDeg> - rotation angle in degree
                            // E.g.:   "ROTATE=90"

                            // Set flip          FLIP=<type>
                            // <type> - type of flipping
                            // E.g.:   X flip: "FLIP=X"
                            // Y flip: "FLIP=Y"
                            // Flip both: "FLIP=XY"

                            // Set zoom          ZOOM=<value>
                            // <value> - zoom level
                            // E.g.:   "ZOOM=10.0"

                            // Execute an external command   APPLICATION=<CmdLine>
                            // <CmdLine> - command line to execute
```

```
                        // E.g.:   "APPLICATION=c:\dev\bin32s\avwin.exe"

                        // Do zoom fit      e.g. "FIT"

                        // Set Views        VIEW=<index>
                        // <index> - View index
                        // E.g.:   "VIEW=1"

                        // Set Views ( by name )   VIEWNAME=<name>
                        // <name> - name of the view ( listed in the views dialog )
                        // E.g.:   "VIEWNAME=topview"

                        // Set Page ( by title )       PAGETITLE=<title>
                        // <title> - title of the page ( listed in the bookmark tree )
                        // E.g.:   "PAGETITLE=Layout1"

                        // Display Note ( using tooltip )
                        // NOTE=Subject0x02<Subject>0x01
                        //          Author0x02<Author>0x01
                        //          Modified0x02<Time Date>0x01
                        //          Notes0x02<Notes>

                        // "seperator" is represented by 0x01
                        // "is equal to" is represented by 0x02


} PAN_BOOKMARK;


class CPanBookmark
{
public:
        operator PAN_BOOKMARK () const

         id_type GetId () const
                        // Get the bookmark identifier

        unsigned int GetLevel () const
                        // Get the bookmark level

        enum  State { S_CLOSED = 0, S_OPENED = 1, S_LEAF = 2 }
        State GetState () const
                        // Get the bookmark state

        const std::(1)wstring & GetName () const
                        // Get the bookmark name

        const std::wstring & GetTarget () const
                        // Get the bookmark target
                        // See detailed documentation in PAN_BOOKMARK::target
};
```

(1) Look at Overview section about usage of STL and Boost libraries

```
typedef struct PAN_BLOCK{              // Block information.
unsigned int          id;
char                  name[80];
} PAN_BLOCK;

typedef struct PAN_CtlPos {            // View coordinates.
double x, y, z;
} PAN_CtlPos;


typedef struct PAN_CtlRange {          // View range.
struct PAN_CtlPos     min;
struct PAN_CtlPos     max;
} PAN_CtlRange;


typedef struct PAN_CtlCaretPos {       // Caret position coordinate.
int                   page;
DWORD                 flow;
DWORD                 offset;
} PAN_CtlCaretPos;


typedef struct PAN_CtlCaretRange {     // Caret-based range.
struct PAN_CtlCaretPos  from;
struct PAN_CtlCaretPos  to;
} PAN_CtlCaretRange;


typedef struct PAN_CtlDimensions {     // Dimensions.of Spreadsheet,
Real                  DimWidth;        // Database, or Archive (rows &
Real                  DimHeight;       // columns)
Real                  DimDepth;
} PAN_CtlDimensions;


typedef struct PAN_CtlInfo {
PAN_FileType          type;      // OUT: type of control as a PAN_FileType value
WORD                  version;  // OUT: version number of control
} PAN_CtlInfo;


typedef struct PAN_CtlHandle {
DWORD                 low;   // The hadle is composed of two double words
DWORD                 high;  // allowing for a total of 2^64 different entities.
} PAN_CtlHandle;
```

```
typedef enum {
PAN_UnknownFile,        // Unsupported file format
PAN_RasterFile,         // File handled by the raster contol
PAN_VectorFile,         // File handled by the vector contol
PAN_DatabaseFile,       // File handled by the database contol
PAN_SpreadsheetFile,    // File handled by the spreadsheet contol
PAN_DocumentFile,       // File handled by the document contol
PAN_ArchiveFile         // File handled by the archive contol
} PAN_FileType;


typedef struct PAN_CtlFileInfo {
PAN_FileType            type;           // OUT: type of file currently
                                        // oppened
char                    name[PAN_MAX_PATH]; // OUT: full pathname
                                            // encoded in MultiByte
                                            // using the local system
                                            // code page
DWORD                   size;           // OUT: file size in bytes
DWORD                   date;           // OUT: file modification date
char                    desc[PAN_CTLMAXDESC];       // OUT:
                                        // format description
PAN_CtlRange            dimensions;     // OUT: view coordinate extents.
WORD                    colorDepth;     // OUT: color depth
int                     nPages;         // OUT: number of pages
int                     tilex, tiley;   // OUT: Width/height of tiles
                                        // 0 if not applicable.
DWORD                   dwHints;        // OUT: Hints, default 0x0000
struct {
                        PAN_Point       offset;         // Insertion point
                        PAN_Point       scale;          // Insertion scaling
                        PAN_Point       dpi;            //Dots-per-inch resolution
                        Real            rot;            // Insertion rotation (radians)
                        WORD            flip;           // Insertion flipping
                        WORD            units;          // Drawing units
                        int             CurPage;        // Last saved page
                        } ins;          // Insertion scaling + offsets.
                                        // Zero if not applicable.
unsigned long           ClipCount;      // OUT: number of clip regions
                                        // in the current page
BOOL                    loadedFromMetafile; // OUT: true if the file is loaded
                                            // from metafile.
} PAN_CtlFileInfo;


class IGetString // Iterator over the page strings returned by PM_CTLGETSTRING message
{
        virtual operator bool () const = 0                      // Check for iteration end

        virtual const std::(1)wstring & operator * () const = 0  // Return the current string

        virtual IGetString & operator++ () = 0                  // Advance to the next string
```

(1) Look at Overview section about usage of STL and Boost libraries

```
        virtual size_t RemainingCount () = 0                    // Return the number of remaining strings
};

typedef struct PAN_CtlSearchInfo {
BOOL                    fMulti;         // IN: Find all occurences, or
                                        // just first.
BOOL                    fDown;          // IN: search downward
BOOL                    fWrap;          // IN: wrap around end of file
BOOL                    fCase;          // IN: match case
BOOL                    fWord;          // IN: match whole word
LPCSTR                  string;         // IN: string to find
PAN_CtlCaretPos         startPos;       // IN: starting file position
WORD                    fFound;         // OUT: Number of found occurrences of the string
WORD                    foundCnt;       // OUT: if > 1 => Total number of entries in
                                        // hFoundHandle and hFoundBBox (single search only).
PAN_CtlCaretPos         foundPos;       // OUT: found file position
PAN_CtlHandle           foundHandle;    // OUT: Handle of found text entity
PAN_CtlPos              foundBBox[4];   // OUT: Bounding box's four vertices

HGLOBAL                 hFoundCnt;      // OUT: Each entry contains the number of hFoundHandle
                                        // and hFoundBBox entries corresponding to the particular
                                        // occurrence of the string (multi-search only)
HGLOBAL                 hFoundPos;      // OUT: Handle to global memory region containing
                                        // list of PAN_CtlCaretPos.
HGLOBAL                 hFoundHandle;   // OUT: Handle to array of PAN_CtlHandle
                                        // structs. The number of entries for each found
                                        // occurrence is stored in hFoundCnt
HGLOBAL                 hFoundBBox;     // OUT: Handle to array of PAN_CtlBoundingBox
                                        // structs. The number of entries for each found
                                        // occurrence is stored in hFoundCnt
} PAN_CtlSearchInfo;


struct CPanTextSearchOptions
{
        CPanTextSearchOptions ( bool bDownSearch=true,
                                bool bWrapSearch=false,
                                bool bMatchCaseSearch=false,
                                bool bMatchWordSearch=false)
        bool bDown;                     // Search direction: true → forward, false → backwards
        bool bWrap;                     // Wrap the search if reaching the first or last page
                                        // depending on the search direction
        bool bMatchCase;                // Match the case for the search string
        bool bMatchWord;                // Match only whole words (i.e. surrounded by spaces)
};

class CPanTextSearch
{
public:
        struct  ContourPoints
        {
                PAN_Point       points[4]; // The four contour points defining a text selection area
        };

        class  ITextSearch
```

```
{
public:
        virtual bool FindNext ( const std::(1)wstring & searchText,          // Search text
                                const CPanTextSearchOptions & options,       // Search options
                                PAN_CtlCaretPos & position,                  // in/out: position
                                std::vector< ContourPoints > *pContours=NULL,
                                                                             // out: resulting
                                                                             // contour areas
                                std::vector< PAN_Handle > *pHandles=NULL)=0
                                                                          // out: pointers to
                                                                          // the handle buffer
};

CPanTextSearch (const std::wstring &searchText,          // The text to be searched
                const CPanTextSearchOptions &options,    // Search options
                const PAN_CtlCaretPos &position)         // Search start position
```

// Find the next hit

```
bool FindNext (   PAN_CtlCaretPos &position,        // Receives position where the string
                                                    // was found
                  std::vector< ContourPoints > *pContours=NULL,
                                                    // Optional pointer to a vector receiving the
                                                    // found string highlight bounding boxes
                  std::vector< PAN_Handle > *pHandles=NULL)
                                                    // Options pointer to a vector receiving the
                                                    // handles corresponding to the records
                                                    // covered by the found string
```

// Find all occurences

```
bool FindAll (    std::vector< std::pair< PAN_CtlCaretPos, size_t > > &positions,
                                                    // A vector of pairs holding the caret position
                                                    // and number of countours & handles
                  std::vector< ContourPoints > *pContours=NULL,
                                                    // Optional pointer to a vector receiving the
                                                    // found string highlight bounding boxes
                  std::vector< PAN_Handle > *pHandles=NULL)
                                                    // Options pointer to a vector receiving the
                                                    // handles corresponding to the records
                                                    // covered by the found string

const std::wstring & GetText () const                       // Returns the current search text

void SetText (const std::wstring &text)                     // Set a new search text

const CPanTextSearchOptions & GetOptions () const           //Get the search options

void SetOptions (const CPanTextSearchOptions &options)      // Set new search options

void SetTextSearch (const boost:: (1)shared_ptr<ITextSearch> &pTextSearch) // Reserved
};
```

(1) Look at Overview section about usage of STL and Boost libraries

```
typedef struct PAN_CtlEntity {
int                 inslevel;       // Coresponds to the insertion level of entities.
                                    // It is zero for top-level entities. For entities defined
                                    // within blocks, it is the level of nesting (1, …).
PAN_CtlHandle       handle[1];      // A variable length array of size inslevel+1.
                                    // The zeroth entry is the handle of the entity itself.
                                    // If an entity is defined within blocks, the entries
                                    //1, … corespond to the handles of each insertion entity,
                                    // from the lowest to the top level.

} PAN_CtlEntity;


Typedef struct PAN_CtlGetEntityInfo {
PAN_CtlRange        bbox;           // IN: bound box of search in
                                    // world coordinates.
Int                 iThreshold;     // IN: Threshold for search in
                                    // screen coordinates, if
                                    // bbox.min == bbox.max
DWORD               nFound;         // OUT: Number of entities found.
HGLOBAL             hFound;         // OUT: Global buffer of found entities:
                                    // Information is stored as PAN_CtlEntry of
                                    // PAN_CtlEntryProps depending on wParam.
} PAN_CtlGetEntityInfo, *LPPAN_CtlGetEntityInfo;


typedef struct PAN_CtlFileFmtList {
WORD                nFmts;          // OUT: The number of formats stored in hFmts.
HGLOBAL             hFmts;          // OUT: Global handle to an array
                                    // of PAN_CtlFileFmt structures.
} PAN_CtlFileFmtList;


typedef struct PAN_CtlFileFmt {
char                desc[PAN_CTLMAXDESC];       // OUT: format description
char                exts[PAN_CTLMAXEXTS];       // OUT: file extensions,
                                                  // e.g., ".bmp.dib"
} PAN_CtlFileFmt;


typedef struct PAN_CtlClpbrdFmtList {
WORD                nFmts;          // OUT: number of formats
HGLOBAL             hFmts;          // OUT: global handle to array
                                    // of PAN_CtlClpbrdFmt structures
} PAN_CtlClpbrdFmtList;


typedef struct PAN_CtlClpbrdFmt {
WORD                fmt;            // OUT: clipboard format,
                                    // One of PAN_CTLCLPBRD
char                desc[PAN_CTLMAXDESC]; // OUT: format description
} PAN_CtlClpbrdFmt;
```

```
typedef struct PAN_CtlSelList {
WORD                nSels;          // OUT: number of selections
HGLOBAL             hSels;          // OUT: global handle to array of
                                    // PAN_CtlSel  structures
} PAN_CtlSelList;


typedef struct PAN_CtlSel {
int                 selType;        // Either PAN_SEL_CARET or
                                    // PAN_SEL_VIEW
union {
PAN_CtlRange        vwRange;        // Range specified in view
                                    // coordinates.
PAN_CtlCaretRange   ctRange;        // Caret based range.
PAN_CtlEntityID     EntityID;       // Specific entity ID.
                    } range;
} PAN_CtlSel;


typedef struct PAN_CtlPrintPreview {
int                 nPages;         // OUT: number of physical pages
WORD                nHorzPages;     // OUT: number of physical pages
                                    // horizontally across image
WORD                nVertPages;     // OUT: number of physical pages
                                    // vertically down the image
LONGRECT            allPagesRect;   // OUT: rectangle of all pages in device
                                    // units
RECT                deviceRect;     // OUT page rectangle of the device
HGLOBAL             imageRects;     // OUT: global handle to array
                                    // PAN_CtlRange structures: image
                                    // rectangles within page rectangles
HGLOBAL             clipRects;      // OUT: global handle to array of RECT
                                    // structures in device units: page
                                    // rectangles excluding marginsand
                                    // header/footers
int                 headerHeight;   // OUT: height of header in
                                    // device coordinates
int                 footerHeight;   // OUT: height of footer device
                                    //  coordinates
BOOL                systemFontUsed; // OUT: TRUE if system font is
                                    //   used
double              scale;          // OUT: relative scaling factor
} PAN_PrintPreview;
```

```
typedef struct PAN_CtlPrintOptions {
PRINTDLG            *printDlg;      // IN: common dialog options
WORD                units;          // IN: one of CTLUNIT_*
double              nImageUnits;    // IN: number of image units
double              nPaperUnits;    // IN: number of paper units
PAN_CtlRange        source;         // IN: area to be printed.
DWORD               mode;           // IN: rendering mode, combination
                                    // of PAN_CTLMODE* flags.  The
                                    // PAN_CTLMODEOPAQUE flag
                                    // is forced when printing.
PanCtlPrintAlignment alignment;     // IN: enumeration with the
                                    // following values:
                                    //       Custom
                                    //       TopLeft
                                    //       TopCenter
                                    //       TopRight
                                    //       MiddleLeft
                                    //       MiddleCenter
                                    //       MiddleRight
                                    //       BottomLeft
                                    //       BottomCenter
                                    //       BottomRight
PHYSPOINT           alignmentCustom;// IN: alignment position if
                                    // alignment is set to Custom.
                                    // PHYSPOINT is a struct with
                                    // two double members: x and y

        struct {
        WORD        units;          // IN: one of CTLUNIT_*
        double      top;
        double      left;
        double      bottom;
        double      right;
                    } margins;      // IN: margins

        struct {
// A header string can contain any of the following escape sequences.
        // %f   full pathname
        // %v   drive letter
        // %d   directory name
        // %b   file basename
        // %e   file extension
        // %p   current page
        // %n   number of pages
        // %%   percent sign

        LOGFONT     font;

        LPCSTR      topLeftText;
        LPCSTR      topCtrText;
        LPCSTR      topRightText;
        LPCSTR      botLeftText;
        LPCSTR      botCtrText;
        LPCSTR      botRightText;
                    } headers;      // IN: header strings
```

```
LPCSTR                  outputFileName;  // IN: output file name
PAN_CtlPrintPreview     printPreview;    // OUT: preview information

} PAN_CtlPrintOptions;
```

```
typedef struct PAN_CtlRenderOptions {

HDC             hdc;            // Display Context.

PAN_CtlRange    source;         // Region to be rendered.

DWORD           mode;
                // Combination of the following modes:
                // PAN_CTLMODEOPAQUE
                // (render in opaque mode)
                // PAN_CTLMODEANISOTROPIC
                // (no aspect ratio adjustment).
                // PAN_CTLMODESPREADSHEET_NOHEADERS
                // (do not render spreadsheet row/column headers).
                // PAN_CTLMODEMONOCHROME
                // (render all entities in black. Vector control only).
                // PAN_CTLMODEPRESERVECLIP
                // (do not reset DC's clip region).
                // PAN_CTLMODEPRESERVEPALETTE
                // (do not select control palette on the DC).
                // PAN_CTLMODERENDERTOPRINTER
                // (render as if to a printer, regardless of DC).
                //PAN_CTLMODERENDERSELECTED
                //(Render only the selected entities)
                //PAN_CTLMODE_HIGHLIGHT_DIMMED
                //(In selected rendering mode use the dimmed
                // highlight mode)


RECT            devRect;        // Rectangle in which to fit selection,
                                // in device units.
int             xDevRes;        // Desired resolution.  Set to zero for default
int             yDevRes;
/*
** Buffer to store scale values for fixed width thick lines.
** Used as an additional scale for the line width when rendering.
*/
WORD            numFixedWidthScale;
Real *          lpFixedWidthScale;

/*
** Buffer used to store pen thickness (pen index corresponds to entity
** color index).  Used to add thickness to entities when rendering.
*/
WORD            numPens;
LPWORD          lpPenThickness;

int             disableAntiAliasing; // Additional anti-aliasing options similar to the
                                     // AntiAliasing INI option in the [Disable] section.
} PAN_CtlRenderOptions;
```

```
typedef struct PAN_CtlObject {
char                ObjectID[16];     // Object Identifier
char                ShareID[16];      // Shared Identifier
int                 ObjectType;       // Embedded or linked object.
                                      // One of: OLE_OBJECT_*

char                filename[PAN_MAX_PATH];
                                      // File containing data.
WORD                flags;            // Object Flags. One of: OBJECTFLAG_*.

LPSTR               lpszHeader;       // Pointer to the file prefix.
int                 cbHeader;         // Size of the file prefix.

WORD                wPageNumber;      // Page number of the file to use
int                 zOrder;           // -1: underlay, 0: inline, 1: overlay.

size_t              ObjectOffset;     // Offset to data in file.
size_t              ObjectSize;       // Size of data in file.

PAN_CtlRange        source;           // Objects position in view coords.
PAN_Border *        borders;          // Not used.
PAN_Shading *       shading;          // Not used.

WORD                wRotate;          // Rotation angle in degrees (0, 90, 180 or 270).
WORD                wFlip;            // Flipping. One of: PAN_FLIP*.
BOOL                fTransform;       // Whether transformation should be applied.
BYTE                TransMethod;      // 0: Matrix, 1: TransformedRegion.
Hmatrix             hm;               // Transformation matrix.
PAN_Point           TransformedRegion[4];     // Transformed region: 4 points, clockwise,
                                              // starting by the top left corner.
color_type          fgColor;          // Foreground color.
BOOL                fUsefgColor       // Whether foreground color should be set in control.
color_type          bgColor;          // Background color.
BOOL                fUsebgColor       // Whether background color should be set in control.
color_type          trColor;          // Not used.
BOOL                fUsetrColor       // Not used.

void *              pLoadingAttributes; // Reserved.
int                 brightness;       // The brightness level [-100, 100], 0 means no change.
int                 contrast;         // The contrast level [-100, 100], 0 means no change.
} PAN_CtlObject;
```

```
typedef struct PAN_CtlShowEntity {
PAN_CtlHandle       handle;         // Entity Handle
DWORD               dwFlags;        // Bit flags: Combination of
                        PAN_CTLSHOWENTITYRESET,     // Use entity's own colors.
                        PAN_CTLSHOWENTITYSETCOLOR,// Use color specified in the struct.
                        PAN_CTLSHOWENTITYXOR,       // XOR color with background.
                        PAN_CTLSHOWENTITYBLOCK,     // SHOW all this block insertion.
                        PAN_CTLSHOWENTITYXREF       // SHOW all this xref insertion.
COLORREF            Color;          // RGB color used to highlight
} PAN_CtlShowEntity;

typedef struct PAN_LAYER {          //Layer information
unsigned int        id;
char                name[320];
COLORREF            color;
BOOL                bState;
BOOL                fThawed;
BOOL                bReadOnly
BOOL                fNotPrintable;
} PAN_LAYER;
```

```
class IResourceContextInfo // Class providing information about the
// context in which the resource is to be located
{
        public:
        virtual const std::(1)wstring & GetBaseDirectory();
                // Returns the directory of the current base file in UNICODE

        virtual const std::wstring & GetProfileName();
                // Returns path of the INI file currently used in UNICODE

        virtual void * GetUserData();
                // Returns the user_data passed in PM_CTLSETFILEEX
};

class IPanResourceInfo // Class providing info about the resource
{
public:
        class Location // Class encapsulating the info about a  location
        {
        public:
                class FileStatus // Class holding the file status information
                {
                public:
                        FileStatus(const std::wstring & originalPath =
         std::(1)wstring(),             // file path
                                        size_t fileSize = 0,          // file size (bytes)
                                        size_t modifiedTime = 0);// last modified

                        FileStatus( const std::wstring & originalPath,     //filepath
                                        const struct _stat & fileStat);          //_stat

                        virtual size_t GetSize() const;
                                // Returns the size of the file in bytes.

                        virtual size_t GetModifiedTime() const;
                                // Returns the last modified time of the file
                                // since midnight January 1st, 1970

                        virtual const std::wstring & GetOriginalPath() const;
                                // Returns the original
                                // file path

                        virtual bool CompareSignature(const FileStatus & rhs
                                ) const; // Comparator

                        virtual void Set(   const std::Error! Bookmark not defined.wstring &
originalPath,
                                                const struct _stat & fileStat);
                                // Set the original file name & status

                        virtual bool IsEmpty() const;
                                // Returns whether  the file status was
                                // initialized.
                }; // End of FileStatus
```

(1) Look at Overview section about usage of STL and Boost libraries

```
        Location(const std::(1)wstring & wsLocatedPath = std::Error! Bookmark not
defined.wstring(),
                                // file location
                        const std::wstring & wsKey = std::Error! Bookmark not defined.wstring(),
                                // unique key (optional)
                        const FileStatus & fileStatus = FileStatus());
                                // file status

        Location(const Location & location);
                 // Copy Constructor

        const std::wstring & GetLocatedPath() const;
                // Returns the absolute path of the located file

        const std::wstring & GetKey() const;
                 // Returns the key set by application during location

        const FileStatus & GetFileStatus() const;
                // Returns the file status

};// End of Location

class ISelection   // Holds the file path of the selected file that
                   // need to be downloaded to an accessible location
{
public:
        virtual const std::wstring & GetLocated() const = 0;
                        // Returns the selected resource's file path

        virtual void SetLocated(const std::wstring & located) = 0;
                        // Sets the downloaded file path
};// End of ISelection

class SelectionIterator // Selection Iterator
{
public:
        ISelection & operator*();
                        // Dereferenced the selection iterator to
                        // retrieve the current selection.

        ISelection * operator->();
                         // Dereferenced the selection iterator to
                        // retrieve the current selection.

        operator bool() const;
                        // Bool conversion operator indicating
                        // whether the iterator has a valid
                        // selection. This will return false after
                        // the iterator has reached the end.

        SelectionIterator & operator++();
                        // Pre-increment operator, which
                        // continues the iteration and moves
```

(1) Look at Overview section about usage of STL and Boost libraries

```
                                // to the next selection.

};// End of SelectionIterator


virtual ~IPanResourceInfo();                    // Destructor

virtual const std::(1)wstring & Name() const;
                                // Returns the resource name
                                //  in UNICODE

virtual unsigned int Type() const;
                        // Returns the type of the resource.
                        // Can be either of RI_TEXTFONT,
                        // RI_LINESTYLE, RI_SHAPE,
                        // RI_VECTOR_EXTERNAL_REFERENCE,
                        // RI_RASTER_EXTERNAL_REFERENCE.

virtual const std::wstring & Path() const;
                                // Returns the resource file path in
                                // UNICODE. The file path may
                                // contain wildcard characters
                                // such as * and ?

virtual const std::wstring & Pattern() const;
                                // Returns the regular expression
                                // pattern that matches the found
                                // file name

virtual const std::wstring & SearchPaths() const ;

/*
Returns the search paths in the following form:

    SearchPaths         := PathEntry [;Paths]
    PathEntry           := [$(Tag)[/|\]][Path]
    Path                := Any file path
    Tag                 := BASEFILE | MODULE | DIRECT | SUBSTITUTE | SpecialTag
                        SpecialTag              := Type=IniEntryKey
                        Type            := PATHS
    IniEntryKey         := XREFPATHS | XFONTPATHS | XREFFILES | …
    Tags:
  BASEFILE              The folder that the basefile is located in.
  MODULE                The folder that AutoVue is installed in.
  DIRECT                Search the filename directly.
  SUBSTITUTE            Use the provided file as an alternative to the missing file.
  SpecialTag            An INI entry under the options section what hold semicolon separated
                        search paths.

  E.g.: "C:\ samples\resources; $(Direct); $(BaseFile)\;        $(Paths=XRefPaths)\resources;
  $(Module)\Font;       $(SUBSTITUTE)C:\temp\substiute.shx".

  A search path may contain wildcard characters such as '*' and '?'. Another character sequence
  "**" has been added, which indicates that a recursive folder search is requested.
```

(1) Look at Overview section about usage of STL and Boost libraries

```
            Example:
                "C:\*\folder" would match
                        "C:\one\folder" and "C:\two\folder".

                "C:\**\folder" would match
                        "C:\one\folder" and "C:\two\three\folder".

                "C:\?ine\folder" would match
                        "C:\nine\folder" and "C:\Fine\folder".
            */

        virtual size_t MaxResults() const;
                                    // Returns the maximum number
                                    // of files that need to be returned as
                                    // located candidates

        virtual const std::(1)string & MagicString() const;
                                    // Returns the sequence of bytes
                                    // at the beginning of the located
                                    // file that must match.

        virtual void SetLocated(const std::vector<Location> & vLocations)
                                    // Sets the resulting located files
                                    // stored in a vector of locations

        virtual void GetLocated(std::vector<Location> & vLocations);
                                    // Fills the locations vector with
                                    // the located file locations

        virtual SelectionIterator GetSelected(); // Get the selected candidates.

        virtual bool Download() const;      // Returns whether the resource
                                            //  needs to be downloaded to
                                            // an accessible location

        virtual bool Search() const;        // Returns whether the resource
                                            //  needs to be located

        virtual std::wstring Identifier() const; // Get unique ID

};// End of IPanResourceInfo


struct PanUserCallback   // Encapsulates the user callback function
                         // pointer and the user data that will be passed to
                         // the callback function as argument
{
        typedef void ( * ResourceLocateProc )( std::vector<csi::IPanResourceInfo *> &,
                                               const IResourceContextInfo &, bool *);
                // User Callback function used in pre- and post-location

        ResourceLocateProc pResourceLocateProc;  // Callback function's pointer to
                                                 // user resource location procedure
```

(1) Look at Overview section about usage of STL and Boost libraries

```
        typedef int(* NotifyProc )(PanNotification notificationMsg, size_t n, void *ptr, void *user_data);
        NotifyProc pNotifyProc;   // Reserved for internal use.

        void * user_data;         // Any user data opaque to VCET.

        PanUserResourceLocate();          // Default constructor

        PanUserResourceLocate(  ResourceLocateProc resourceLocateProc,
                                NotifyProc notifyProc,
                                void * userData);
};

struct PAN_CtlSetFile
{
std::(1)wstring          file_name;          // File name in Unicode

PanUserCallback          user_ callback;     // user resource locate

PAN_CtlSetFile();                            // Default constructor

PAN_CtlSetFile(std::wstring filename, PanUserCallback usercallback);
};

typedef struct PAN_RESOURCEINFO {
UINT        resourceType;                    // Text font, linestyle, shape, raster
                                             // or vector external reference file
                                             // (one of RI_* types listed in the doc of
                                             // ResourceInfo::type below).
char        szResourceID [_MAX_PATH];        // Name of resource ID (Logical Name)
UINT        cbResourceID;                    // Number of characters in the
                                             // resource ID
char        szResource [_MAX_PATH];          // Name of resource (File Name)
UINT        cbResource;                      // Number of characters in the
                                             // name of the resource
char        szResolvedResource [_MAX_PATH];  // Full path to the resource or substituted res.
UINT        cbResolvedResource;              // Number of characters in the full path
char        szSearchResources [_MAX_SEARCHPATHS];
                                             // Paths to search in,
                                             // in order to locate the resources
UINT        cbSearchResources;               // Number of characters in the
                                             // szSearchResources
char        szSearchHints [_MAX_SEARCHHINTS]; // Hints used in search to locate resources
UINT        cbSearchHints;                   // Number of characters in the szSearchHints
BOOL        fOK;                             // Resource Status Flag. Possible values are:
                                             // 0 : Missing, 1 : Located, 2 : Substituted
                                             // These values are the same as the
                                             // ResourceInfo::Status enumeration.
} PAN_RESOURCEINFO;

struct ResourceInfo {
        enum Status {
                Missing,        // The resource was not found.
                Located,        // The resource was found or replaced
                                // by an equivalent substitute.
```

(1) Look at Overview section about usage of STL and Boost libraries

```
            Substituted         // The resource was not found but substituted.
        };

        int                     type;
                        // The type of the resource, possible values are:
                        // RI_TEXTFONT, RI_LINESTYLE, RI_SHAPE,
                        // RI_VECTOR_EXTERNAL_REFERENCE,
                        // RI_RASTER_EXTERNAL_REFERENCE,
                        // and RI_UNKNOWN.

        std::(1)wstring         name;
                        // The name of the resource in UNICODE

        std::wstring            note;           // The note of the resource

        std::wstring            identifier;     // Unique identifier

        std:: wstring           path;           // Resource file path

        Status                  eStatus;        // The location status
};

class IGetResourceInfo
{
public:
        virtual operator bool() const;          // Check for iteration end

        virtual const ResourceInfo & operator*() const;
                                // Returns the current resource info

        virtual IGetResourceInfo & operator++();
                                // Advance  to the next resource info
};

typedef struct {
id_type                 id;                     // Unique id
char                    name[_MAX_PATH];        // Logical Name char
                        fname[_MAX_PATH];       // Full filename
BOOL                    bState;                 // 1: On, 0: Off,
                                                //  -1: Disabled
} PAN_XREF;

class CPanXRef
{
        CPanXRef (id_type id=0, const std::wstring &name=std::wstring(),
                const std::wstring &path=std::wstring(), bool bState=true, bool bActive=true)
                        // Constructor

        id_type GetId () const
        void SetId (id_type id) const
                        // Get/Set XRef identifier

        std::wstring & GetName () const
        void SetName (const std::wstring &name)
```

(1) Look at Overview section about usage of STL and Boost libraries

```
                        // Get/Set XRef name

        const std::wstring & GetPath () const
        void SetPath (const std::(1)wstring &path)
                        // Get/Set XRef path

        bool IsOn () const
        void SetState (bool bOn=true)
                        // Whether this XRef is enabled

        void SetActive (bool bActive=true)
        bool IsActive () const
                        // Whether this XRef is activated
};
```

Command Messages

VCET, as an application programming interface (API), itself makes use of the Microsoft Windows API function SendMessage() in order to exploit its viewing and conversion capabilities. There is a certain advantage to this method, particularly in the case of a viewing and conversion tool, namely that the end user of your application does not directly access the current file, but rather sends his commands to the window or viewport in which the file is being rendered. This practically eliminates the risk of accidental data modification. This section takes an in-depth look at the command messages which form the core of VCET's functionality.

Command Message Summary

Control-Specific Command Messages

Message Name	Purpose
PM_CTLCLONECONTROL	Creates a duplicate of a control and returns its handle.
PM_CTLDESTROY	Prepares a control for destruction.
PM_CTLGETDIMS	Returns the dimensions of the control
PM_CTLGETCAPS	Returns the currently enabled capabilities of a control.
PM_CTLGETFILE	Returns information about the current file.
PM_CTLGETFILEFMTS	Returns file formats supported by a control.
PM_CTLGETFILETYPE	Returns the format class and an index into the list provided by PM_CTLGETFILEFMTS
PM_CTLGETINFO	Returns information about a control.
PM_CTLGETMODE	Returns the currently enabled modes of a control.
PM_CTLGETOPTION	Returns the control options.
PM_CTLGETSTATUS	Returns the current status of a control.
PM_CTLREGEN	Reread the file from disk.
PM_CTLSETCAPS	Modifies the capabilities of a control.
PM_CTLSETFILE	Renders the given file in the control window.
PM_CTLSETMODE	Modifies the modes of a control.
PM_CTLSETOPTION	Set the control options.

Presentation Command Messages

PM_CTLFLIP	Flips the contents of the control.
PM_CTLGETFLIP	Returns the current flipping state.
PM_CTLGETROTATION	Returns the current rotatation set in the control.
PM_CTLGETZOOM	Returns the current zoom factor.
PM_CTLHSCROLL	Similar to WM_HSCROLL.
PM_CTLPAINT	Similar to WM_PAINT.
PM_CTLROTATE	Rotates the contents of the control.
PM_CTLSETZOOM	Sets the zoom factor.
PM_CTLSIZE	Similar to WM_SIZE.
PM_CTLVSCROLL	Similar to WM_VSCROLL.

Conversion Command Message

PM_CTLCONVERT	Obtain handles to conversion functions.

Coordinates-related messages

PM_CTLCARETTOWORLD	Returns the view coordinate that corresponds to the specifiedcaret position.
PM_CTLGETCARETPOS	Returns the current position of the caret.
PM_CTLSETCARETPOS	Moves the caret to the specified position.
PM_CTLWORLDTOCARET	Returns the caret position that corresponds to the given world coordinates.
PM_CTLWORLDTOCLIENT	Returns the client area coordinates (i.e., relative to the top, left corner of the client area of the control window) corresponding to the given world coordinates.
PM_XFRMRECT	Applies/Removes current rotation and flipping to/from a world coordinate range.

Clipboard and selection-related messages

PM_CTLCLEARSELS	Clears all selections.
PM_CTLCOPY	Copy the current selections to the clipboard.
PM_CTLGETCLPBRDFMTS	Returns clipboard formats supported by a control.
PM_CTLGETNUMSELS	Returns the current number of selections.
PM_CTLGETSELS	Returns a list of the current selections.
PM_CTLSETSEL	Sets or removes a selection based on view coordinates.
PM_CTLSETSELCARET	Sets or removes a selection specified using caret positions.

Color-related messages

PM_CTLGETFGBGCOLOR	Returns the foreground or background color.
PM_CTLGETPALETTE	Returns palette information.
PM_CTLPALETTECHANGED	The calling application must send this message to the control whenever it receives a WM_PALETTECHANGED message.
PM_CTLQUERYNEWPALETTE	The calling application must send this message to the control whenever it receives a WM_QUERYNEWPALETTE message.
PM_CTLSETFGBGCOLOR	Sets the foreground or background color.
PM_CTLSETPALETTE	Sets the color palette.

Views-related message

PM_CTLGETVIEW	Returns the “active named view” in the control.
PM_CTLGETVIEWEXTENTS	Returns the current view extents of the current file in view coordinates.
PM_CTLGETVIEWNAMES	Returns a buffer of “view” names set by PANX_SetViews().
PM_CTLSETVIEW	Sets the view to use in the control.
PM_CTLSETVIEWEXTENTS	Sets the view extents of the current file in view coordinates.

Blocks-related messages

PM_CTLGETBLOCK	Gets the “active block” in the control.
PM_CTLGETBLOCKNAMES	Returns a buffer of block names set by PANX_SetBlocks().
PM_CTLSETBLOCK	Sets the “active block” in the control.

Entities-related

PM_CTLGETENTITY	Retrieves entity handles found within a specified region (vector files).
PM_CTLSHOWENTITY	Sets entity’s drawing color e.g. to highlight a selected entity (vector files).

Extended Image Data Command Messages

PM_CTLGETIMAGEEX	Retrieves the image’s current contrast/anti-aliasing setting.
PM_CTLSETIMAGEEX	Adjusts the image’s contrast and/or anti-aliasing.

Layers-Related Command Messages

PM_CTLGETLAYERSTATE	Returns a buffer of layer states set by PANX_SetLayers().
PM_CTLSETLAYERSTATE	Sets the internal buffer of layers in the control.

XRef-Related Command Messages

PM_CTLGETXREFSTATE	Returns a buffer of XRef states set by PANX_SetXRefs().
PM_CTLSETXREFSTATE	Sets the internal buffer of XRefs in the control.

Bookmark messages.

PM_CTLGETBOOKMARKS	Returns a buffer of PAN_BOOKMARK struct set by PANX_SetBookMarks.
PM_CTLSETBOOKMARKS	Not implemented

Resource-Related Command Messages

PM_CTLGETRESOURCEINFOSTATE	Returns a buffer of resource information set by PANX_ResourceInfo().

Offset-Related Command Messages

PM_CTLGETOFFSET	Returns the offset of the origin for the current file in view coordinates.
PM_CTLSETOFFSET	Sets the offset of the origin for the current file in view coordinates.

Font-Related Command Messages

PM_CTLGETBASEFONT	Returns the base font of the current file.
PM_CTLSETBASEFONT	Sets the base font of the current file.

Database and Spreadsheet-Related Command Messages

PM_CTLGETCOLWIDTH	Return the width of the specified field/column in the specified units.
PM_CTLSORT	Sorts a specified range in a database.
PM_CTLGETROWHEIGHT	Returns the height of the specified record/row in specified units.

Page-Related Command Messages

PM_CTLGETNUMPAGES	Returns the number of pages in the current file.
PM_CTLGETPAGE	Returns the current page number within the current file.
PM_CTLGETPAGESIZE	Returns the size of the specified page in the current file.
PM_CTLSETPAGE	Sets the current page number within the current file.

Printing-Related Command Messages

PM_CTLPRINT	Prints the specified extents.
PM_CTLPRINTPREVIEW	Returns the print preview of the specified extents.
PM_CTLVALIDATEMARGINS	Validates the printer margins.

Text-Specific Command Messages

PM_CTLGETSTRING	Retrieves a specified string from a control.
PM_CTLSEARCH	Search the current file for a text string.

Mouse-Specific Command Messages

PM_CTLGETLMBACTION	Returns the left mouse button control window behavior.
PM_CTLSETLMBACTION	Sets the left mouse button control window behavior.

Device Context Command Messages

PM_CTLRENDERONTODC	Renders the specified extents onto the given device context.

Formatting Device Command Messages

PM_CTLSETDEVICE	Set formatting device for document files.
PM_CTLGETDEVICE	Gets current formatting device for document files.

Command Messages dropped from Version 1.1 of VCET Controls

PM_CTLGETBOOKMARK	Returns the bookmark position within the current file.
PM_CTLGETDC	Returns a handle to the device context set by PM_CTLSETDC.
PM_CTLSETBOOKMARK	Positions the bookmark at a given file position.
PM_CTLSETDC	Sets the current device context handle to be used by the control when drawing or converting coordinates.

Command Message Descriptions

The following section presents a brief description along with the meaning of each parameter and the return value, for each command message, organized by functional category.

Controls Command Messages

PM_CTLCLONECONTROL

Purpose Creates a duplicate of a control and return its handle.

Description Create a duplicate of the control and return a handle to the window of the new control. The window of the new control can be a stand-alone window, in which case both *wParam* and *lParam* must be zero; or a child window if *lParam* initially points to a parent window, in which case, *wParam* must contain a unique child identifier greater than zero. The new control is created in a hidden state.

NOTE: While view state is preserved between the original and cloned control, the clone control does not inherit the selections that may have been present in the original control.

Parameters

wParam: 0 if new control window is to be stand-alone **(WORD)** child ID, if new control window has a parent

lParam: 0 if new control window is to be stand-alone **(HWND *)** parent window handle (in) /control window handle (out), if new control window has a parent

Returns: error code

Compatibility: all control types

Example:

```
HWND Clone(HWND ctrlHandle)
/* Create an independent clone of the control pointed to by ctrlHandle */
{
        HWND newHndl; // the new control has no parent
        int err;

        if (ctrlHandle == NULL) return ((HWND) 0);

        err = (int) SendMessage(ctrlHandle, PM_CTLCLONECONTROL, 0,
                        (LPARAM) &newHndl);
        if (err != PAN_CTLERRNONE) return((HWND) 0);
        return newHndl;
}
```

PM_CTLDESTROY

Purpose Prepares a control for destruction.

Parameters *wParam*: not used

lParam: not used

Returns: error code

Compatibility: all control types

Special Notes: This message must be sent before destroying the control window with the Windows **DestroyWindow()** function.

PM_CTLGETCAPS

Purpose Returns the capabilities of a control.

Description If *wParam* is zero, return a mask of the currently enabled capabilities of the control in the **DWORD** pointed to by *lParam*. If *wParam* is non-zero, return a mask of all the capabilities of the control. The mask consists of zero, one or more of the following constants ORed together.

PAN_CTLCAPSZOOM	can zoom
PAN_CTLCAPSCOPY	can copy to clipboard
PAN_CTLCAPSSEARCH	can search for string
PAN_CTLCAPSPAGE	can go to page
PAN_CTLCAPSSIZE	can handle resize events
PAN_CTLCAPSHSCROLL	can handle horizontal scroll events
PAN_CTLCAPSVSCROLL	can handle vertical scroll events
PAN_CTLCAPSMOUSE	can handle mouse events
PAN_CTLCAPSKEYBD	can handle keyboard events

Parameters

wParam:	***Zero***	Get all enabled capabilities of the control.
	Non-Zero	Get all possible capabilities of the control.
lParam:	***(wParam:Zero)***	**(LPDWORD)** mask of all currently enabled capabilities of the control
	(wParam:Non-Zero)	**(LPDWORD)** mask of all possible capabilities of the control

Returns: error code

Compatibility: all control types

Example:

```
void ShowCaps(HWND ctrlHandle)
//      Display the current capabilities of the control using 1 if enabled
//      and 0  if disabled.
{
        DWORD caps=0;

        if (ctrlHandle == NULL) return;

        SendMessage(ctrlHandle, PM_CTLGETCAPS, 0, (LPARAM) &caps);
        Output("Zoom     : %d\n",
                (caps&PAN_CTLCAPSZOOM) == PAN_CTLCAPSZOOM);
        Output("Copy     : %d\n",
                (caps&PAN_CTLCAPSCOPY) == PAN_CTLCAPSCOPY);
```

```
        Output("Search    : %d\n",
                (caps&PAN_CTLCAPSSEARCH) == PAN_CTLCAPSSEARCH);
        Output("Page      : %d\n",
                (caps&PAN_CTLCAPSPAGE) == PAN_CTLCAPSPAGE);
        Output("Size      : %d\n",
                (caps&PAN_CTLCAPSSIZE) == PAN_CTLCAPSSIZE);
        Output("HScroll   : %d\n",
                (caps&PAN_CTLCAPSHSCROLL) == PAN_CTLCAPSHSCROLL);
        Output("VScroll   : %d\n",
                (caps&PAN_CTLCAPSVSCROLL) == PAN_CTLCAPSVSCROLL);
        Output("Mouse     : %d\n",
                (caps&PAN_CTLCAPSMOUSE) == PAN_CTLCAPSMOUSE);
        Output("Keybd     : %d\n",
                (caps&PAN_CTLCAPSKEYBD) == PAN_CTLCAPSKEYBD);
}
```

PM_CTLGETDIMS

Purpose Returns the dimensions of the control, in terms of rows and columns, in the structure passed. For raster, vector, and document files it returns the dimensions in logical units.

Description This message is intended primarily for spreadsheet and database controls, which can be measured in terms of rows and columns as well as by view extents. This message returns these dimensions in the ***PAN_CtlDimensions*** structure pointed to by *lParam*.

Parameters *wParam*: not used

lParam: **(PAN_CtlDimensions *)** dimensions

Returns: error code

Compatibility: all control types

Special Notes: The dimensions that are returned from a document control do not include the 5mm interpage gaps that are added when the file is displayed.

Example:

```
int ShowDimensions(HWND ctrlHandle)
/* Output the dimensions of an archive, database, or spreadsheet control */
/* in terms of rows and columns. */
{
        int err;
        PAN_CtlDimensions dimensions;

        if (ctrlHandle == NULL)
                return PAN_CTLERRMISC;

        // Get control dimensions.
        err = (int)SendMessage(ctrlHandle, PM_CTLGETDIMS, 0, (LPARAM) &dimensions);

        if (err == PAN_CTLERRNONE) {
                Output("Control dimensions = (%d, %d, %d)\n",
                        dimensions.DimWidth,
                        dimensions.DimHeight,
                        dimensions.DimDepth);
        }

        return err;
}
```

PM_CTLGETFILE

Purpose Returns information about the current file.

Description Return information about the current file in the **PAN_CtlFileInfo** structure pointed to by *lParam*.

For vector files, if entity information is available the CTL_FILE_HINT_EDAT hint flag will be set in the dwHints field of the returned file information. All other bits of the dwHints field are reserved.

NOTE: The dimensions that are returned from a document control do not include the 5mm interpage gaps that are added when the file is displayed.

Parameters *wParam*: reserved, must be **0**

lParam: **(PAN_CtlFileInfo *)** file information

Returns: error code

Compatibility: all control types

Example:

```
        void ShowFileInfo(HWND ctrlHandle)
/*      display some file information on the output device. */
{
        PAN_CtlFileInfo fi;

        if (ctrlHandle == NULL) return;

        SendMessage(ctrlHandle, PM_CTLGETFILE, 0, (LPARAM) &fi);
        Output("name: %s\r", fi.name);
        Output("size: %d\r", fi.size);
        Output("date: %d\r", fi.date);
        Output("desc: %s\r", fi.desc);
}
```

PM_CTLGETFILEFMTS

Purpose Returns file formats supported by a control.

Description Return a description of each file format supported by the control in the **PAN_CtlFileFmtList** structure pointed to by *lParam*.

Parameters *wParam*: not used

lParam: **(PAN_CtlFileFmtList *)** structure to be filled with the formats information.

Returns: error code

Compatibility: all control types

Example:

```
        void ShowFileFmts(HWND ctrlHandle)
/*      Display the supported file formats on the output device. */
{
        PAN_CtlFileFmtList              fl;
        PAN_CtlFileFmt *                f;
        WORD                            i;
        char  desc[PAN_CTLMAXDESC];
        char  exts[PAN_CTLMAXEXTS];

        if (ctrlHandle == NULL) return;
        SendMessage(ctrlHandle, PM_CTLGETFILEFMTS, 0, (LPARAM) &fl);
        f = (PAN_CtlFileFmt *)GlobalLock(fl.hFmts);
        for (i=0; i<fl.nFmts; i++) {
                _fstrcpy(desc, f[i].desc);
                _fstrcpy(exts, f[i].exts);
                Output("%s (%s)\r", desc, exts);
                }
        GlobalUnlock(fl.hFmts);
}
```

PM_CTLGETFILETYPE

Purpose Returns the type of control associated with a given file.

Description The control types are listed below.

Parameters

wParam:
- 0 The file name encoded in MultiByte using the local system codepage.
- 1 The file name encoded in Unicode

lParam:
- ***(wParam = 0)*** **(LPCSTR)** filename
- ***(wParam = 1)*** **(wchar_t *)** filename

Returns A constant of type **PAN_FileType** is returned in the low word, and an index into the list provided by **PM_CTLGETFILEFMTS** is returned in the high word. If the format of the given file is not supported or an error occurred, **PAN_UnknownFile** is returned.

```
typedef enum {
        PAN_UnknownFile,      // The given file is not supported or
                              // an error has occurred.
        PAN_RasterFile,       // The given file is a raster format.
        PAN_VectorFile,       // The given file is a vector format.
        PAN_DatabaseFile,     // The given file is a database format.
        PAN_SpreadsheetFile,  // The given file is a spreadsheet format.
        PAN_DocumentFile,     // The given file is a document format.
        PAN_ArchiveFile       // The given file is a archive format.
} PAN_FileType;
```

Compatibility: all control types

Example:

```
void ShowFileType(HWND ctrlHandle, LPCSTR fileName)
/* Display the file type of the specified file on the output device. */
{
        WORD lowbyte;

        if (ctrlHandle==NULL) return;
        lowbyte = (WORD) LOWWORD(SendMessage(ctrlHandle, PM_CTLGETFILETYPE, 0,
                (LPARAM) filename));
        switch (lowbyte) {
        case PAN_RasterFile:        Output("File is of type : Raster");
                                    break;
        case PAN_VectorFile:        Output("File is of type: Vector");
                                    break;
        case PAN_DatabaseFile:      Output("File is of type:
                                    Database"); break;
        case PAN_SpreadsheetFile:   Output("File is of type:
                                    Spreadsheet"); break;
        case PAN_DocumentFile:      Output("File is of type:
                                    Document"); break;
        case PAN_ArchiveFile:       Output("File is of type: Archive");
                                    break;
        default:  Output("Unknown file type or an error                    occurred");
        } // end switch
}
```

PM_CTLGETINFO

Purpose Returns information about a control.

Description Returns type and version information about the control in the **PAN_CtlInfo** structure pointed to by *lParam*.

Parameters *wParam*: not used

lParam: **(PAN_CtlInfo *)** control information

Returns: error code

Compatibility: all control types

Example:

```
void ShowInfo(HWND ctrlHandle)
/*      Display the current version on the output device. */
{
        int err;
        PAN_CtlInfo i;    // info structure

        if (ctrlHandle == NULL) return;

        err = SendMessage(ctrlHandle, PM_CTLGETINFO, 0, (LPARAM) &i);
        if (err != PAN_CTLERRNONE) return;
        Output("Control version: %d", i.version);
}
```

PM_CTLGETMODE

Purpose Returns the currently enabled modes of a control.

Description If *wParam* is zero, return a mask of the currently enabled modes of the control in the **DWORD** pointed to by *lParam*. If *wParam* is non-zero, return a mask of all the modes of the control. The mask consists of zero, one or more of the following constants ORed together.

PAN_CTLMODEOPAQUE	Control is opaque
PAN_CTLMODENOREDRAW	Redraws are disabled
PAN_CTLMODEEXCESSSCROLL	Scrolling is not constrained
PAN_CTLMODEANISOTROPIC	Aspect ratio is not preserved
PAN_CTLMODEDRAGDROP	Control accepts dropped files
PAN_CTLMODEINTERRUPTIBLE	Control may be interrupted
PAN_CTLMODEMONOCHROME	Draw all entities in black. Only Supported by the vector control.
PAN_CTLMODEPRESERVECLIP	Preserve clip region of target device.
PAN_CTLMODEPRESERVEPALETTE	Preserve palette of target device.
PAN_CTLMODEIGNOREMINMARGINS	Ignore printing margins.
PAN_CTLMODERENDERSELECTED	Only render selected entities.
PAN_CTLMODERENDERTOPRINTER	Give the same render output as a printer even the DC is not a printer.
PAN_CTLMODELIMITTOONETILE	Limit to one tile in printing.
PAN_CTLMODE_HIGHLIGHT_DIMMED	Dimmed highlight mode.
PAN_CTLMODE_DISABLE_FORCETOBLACK_ONRASTER	Prevents the Force-To-Black from being applied to raster overlays(images) in vector/raster overlays.
PAN_CTLMODE_TILE	CMF tile mode.
PAN_CTLMODE_FORCE_BKG_PAINT	Paint in force-to-black mode.

Parameters *wParam*: zero or non-zero

lParam: **(LPDWORD)** mask of mode flags

Returns: error code

Compatibility: all control types

Example:

```
void ShowMode(HWND ctrlHandle)
/*      Display the current control mode settings on the output device. */
{
        WORD mode=0;

        if (ctrlHandle == NULL) return;
        SendMessage(ctrlHandle, PM_CTLGETMODE, 0, (LPARAM)&mode);
        Output("Opaque   : %d\r", (mode&PAN_CTLMODEOPAQUE)!=0);
        Output("NoRedraw : %d\r", (mode&PAN_CTLMODENOREDRAW)!=0);
        Output("Ex Scrl  : %d\r", (mode&PAN_CTLMODEEXCESSSCROLL)!=0);
        Output("Aniso    : %d\r", (mode&PAN_CTLMODEANISOTROPIC)!=0);
        Output("Drag&drop: %d\r", (mode&PAN_CTLMODEDRAGDROP)!=0);
        Output("Interrupt: %d\r", (mode&PAN_CTLMODEINTERRUPTIBLE)!=0);
```

```
}
```

PM_CTLGETOPTION

Purpose Gets information about different control options.

Description Depending on the control type it offers the possibility to read the current display values for different types of file.

PAN_ CTLCADOPTIONS Get the control CAD options. The CAD options are used to Enable/Disable. Some features (Text, Line-Styles, Filling…). This option is supported by Vector Control (2D/3D).

PAN_ CTLOPTDIMLEVEL Get the current set value for dim highlight level. Dim level, in case dim highlight mode has been selected, controls how much the rest of the drawing dimmed with respect to the highlighted items.

Parameters *wParam*: **PAN_ CTLCADOPTIONS** or **PAN_ CTLOPTDIMLEVEL**

lParam: (**wParam = PAN_CTLCADOPTIONS)**
Mask of the following CAD options flags:

PAN_CADOPTIONS_NOTEXT: Disable the text in the model.
PAN_CADOPTIONS_NODIMENSIONS Disable dimensional primitives in the model.
PAN_CADOPTIONS_NOXREFS: Disable XRefs in the model.
PAN_CADOPTIONS_NOFILLS: Instructs the snapping code to consider snapping filled areas if this flag is not set.
PAN_CADOPTIONS_NOLINESTYLES: Disable Line-Styles on all the primitives of the model. They will be drawn using solid lines.
PAN_CADOPTIONS_NOLINEWEIGHTS: Don't consider the line thickness for model primitives.
PAN_CADOPTIONS_FASTDISPLAY: Enable fast display rendering optimizations: whenever is possible the original model primitives will be displayed using approximation in order to speed up the rendering.
PAN_CADOPTIONS_FULLDISPLAY: Disable any kind of rendering optimizations. However, if both full display and fast display flags are specified the fast display will have priority (i.e. the full display flag will be ignored in this case.)

(wParam = PAN_OPTDIMLEVEL)
(double*) pointer to the value receiving the dim level.

Returns: error code

Compatibility: vector-2D

Example:

```
/* Get the current dim level*/

double dimLevel = 0.0;
WORD wParam = PAN_CTLOPTDIMLEVEL;
DWORD lParam = (DWORD)&dimLevel;
…
SendMessage (hwndCtl, PM_CTLGETOPTION, wParam, lParam);
…
```

PM_CTLGETSTATUS

Purpose Returns the current status of a control.

Description Return the current status of the control in the **DWORD** pointed to by *lParam*.

Parameters *wParam*: not used

lParam: a pointer to a **DWORD** specifying a mask of the following constants:

PAN_CTLSTATUSIDLE	control is idle
PAN_CTLSTATUSPROCESSING	control is processing the file
PAN_CTLSTATUSREFRESHING	control is refreshing the window

Returns: error code

Compatibility: all control types

Example:

```
void ShowStatus(HWND ctrlHandle)
/*      Display the status of the control on the output device. */
{
        DWORD stat;

        if (ctrlHandle == NULL) return;

        SendMessage(ctrlHandle, PM_CTLGETSTATUS, 0, (LPARAM) &stat);
        if (stat&PAN_CTLSTATUSIDLE)
                Output("GETSTATUS: Idle...\r");
        if (stat&PAN_CTLSTATUSPROCESSING)
                Output("GETSTATUS: Processing...\r");
        if (stat&PAN_CTLSTATUSREFRESHING)
                Output("GETSTATUS: Refreshing...\r");
}
```

PM_CTLREGEN

Purpose Redraws file using new parameters.

Description Causes a redraw of the file. This differs from repainting in that the file is re-read from disk. This message can be used if the contents of a file have changed, or if the user has changed any viewing options.

Note:that the following information is retained:
The page number currently viewed.
The layer states.
The block state.
The current "view" of the file (i.e., for graphics files, the offset and zoom factor, for doc files, the position in the document, for db/ss files the top left and current cell).

Parameters *wParam*: (**WORD**)Can be one of:

PAN_CLEARLAYERS clears layers from file before re-drawing
PAN_CLEARBLOCKS clears blocks from file
PAN_CLEARVIEWS clears views from file
PAN_CLEARVIEWEXTENTS clears view extents from file before re-drawing
PAN_FITVIEWEXTENTS re-draw the file to fit the current view extents

Note: If the user does not provide a *wParam*, the file will be re-drawn as is.

lParam: Not used

Returns: error code

Compatibility: all control types

Example:

```
void ShowAllBlocks(HWND ctrlHandle)
/* Force the complete vector drawing to be displayed.. */
{
        int err;

        if (ctrlHandle == NULL) return;

        err = (int) SendMessage(ctrlHandle, PM_CTLSETBLOCK, -1, 0);

        if (err != PAN_CTLERRNONE) {
                Output("Couldn't reset active block.\n");
                }

        // Force re-reading of input file.
        SendMessage(ctrlHandle, PM_CTLREGEN, 0, 0);
}
```

PM_CTLSETCAPS

Purpose Set the capabilities of a control.

Description If *wParam* is zero, enable the capabilities of the control specified in the ***DWORD*** pointed to by *lParam* (any capabilities not specified are disabled). If *wParam* is non-zero, enable allcapabilities supported by the control. The mask consists of zero, one or more of the following constants ORed together.

PAN_CTLCAPSZOOM	can zoom
PAN_CTLCAPSCOPY	can copy to clipboard
PAN_CTLCAPSSEARCH	can search for string
PAN_CTLCAPSPAGE	can go to page
PAN_CTLCAPSSIZE	can handle resize events
PAN_CTLCAPSHSCROLL	can handle horizontal scroll events
PAN_CTLCAPSVSCROLL	can handle vertical scroll events
PAN_CTLCAPSMOUSE	can handle mouse events
PAN_CTLCAPSKEYBD	can handle keyboard events

Parameters

wParam:	***Zero***		enable capabilities specified by the mask (disable those not specified)
	Non-Zero		enable all capabilities supported by the control
lParam:	(***wParam:Zero***)	**(DWORD)**	mask of capabilities
	(***wParam:Non-Zero***)		not used

Returns: error code

Compatibility: all control types

Example:

```
void EnableCaps(HWND ctrlHandle, DWORD caps)
/* Try to enable all the capabilities specified in the caps variable.
If one or more capabilities are not supported then none are set.*/
{
        DWORD allCaps;

        if (ctrlHandle == NULL) return;

        // get all caps of control
        SendMessage(ctrlHandle, PM_CTLGETCAPS, 1, (LPARAM) ((LPDWORD)                    &allCaps));

        // enable current cap only if control support it
        if ((allCaps & caps) == caps) {
                DWORD curCaps;
                // get current caps
                SendMessage(ctrlHandle, PM_CTLGETCAPS, 0, (LPARAM)
                        ((LPDWORD) &curCaps));
```

```
                curCaps |= caps;
                SendMessage(ctrlHandle, PM_CTLSETCAPS, 0, (LPARAM) curCaps);
        }
}
```

PM_CTLSETFILE

Purpose Renders the given file in the control window.

Description Render the given file in the control window. If *wParam* is not -1, it must be an index into the list of file formats returned by **PM_CTLGETFILEFMTS**. Otherwise, the first compatible format will be used.

Parameters *wParam* file format index or -1

lParam: **(LPCSTR)** filename

Returns: error code

Compatibility: all control types

Example:

```
void ShowFile(HWND ctrlHandle, LPCSTR fileName)
/* Display the named file in the specified control. */
{
        int err;

        if (ctrlHandle == NULL) return;

        err = SendMessage(ctrlHandle, PM_CTLSETFILE, -1, (LPARAM)filename);

        if (err != PAN_CTLERRNONE) {
                Output("Couldn't display specified file\n");
        }
}
```

PM_CTLSETFILEEX

Purpose Renders the given file in the control window.

Description This is an enhanced version of the message **PM_CTLSETFILE** that sets a file in the control window supporting **UNICODE** and application callback procedures.

Parameters

wParam The file format index from the list of file formats returned by **PM_CTLGETFILEFMTS**, otherwise -1

lParam: **(PAN_CTLSetFile *)** filename and callback

Returns: error code

Compatibility: all control types

Special Notes This message has been designed to support UNICODE file names. However, not all formats support UNICODE. For those, which do not, the UNICODE string will be converted to MULTIBYTE using the local system code page, then used to load the file. Despite this limitation, UNICODE resource names will be sent back to the application for user resource location. This allows the application to locate UNICODE resource names for all formats.

Example:

```
/*
  Setup a user resource location procedure with the file to open in VCET window control
*/

typedef std::(1)vector<IPanResourceInfo *>  ResourceVector;

/**      User Resource Locate Callback method */
void DMSLocateResource(ResourceVector & resources,
                                const IResourceContextInfo &    context,
                                bool * pContinue) {
        // Pre-Locate (before VCET tries to locate the files)
        if(pContinue){
                // Perform Localization search
                for (ResourceVector::iterator resIter =
                        resources.begin();
        resIter != resources.end(); ++resIter){

                        // Find the file and obtain the resolved path
                        std::vector<std::wstring> vResult =
                                DMSSearch(
                                        resIter->Path(),
                                        resIter->Pattern(),
                                        resIter->SearchPaths(),
                context.GetBaseDirectory(),
                                        context.GetProfileName(),
                context.GetUserData());
```

(1) Look at Overview section about usage of STL and Boost libraries

```
                        // Set the resolved path
                        resIter->SetLocated(vResult);

                        // Set the keys
                        std::(1)vector<std::wstring> vKeys =
                                                DMSGetID(vResults);
                        resIter->SetKeys(vKeys);
                }
                // Do not continue because we have found them.
                *pContinue = false;
        }
}

void OpenFile(const std::wstring & file_name) {
        // Define the user resource localization structure
        PanUserResourceLocate user_resource_locate =
                                        {DMSLocateResource , my_data};
        // Define the SetFile structure
        PAN_CtlSetFile set_file = {file_name,user_resource_locate};

        // Set the file
        SendMessage(hwndCtl, PM_CTLSETFILEEX, -1, &set_file);
        ...
        // Load/Render the file
        SendMessage(hwndCtl, PM_CTLPAINT, wParam, lParam);
}
```

(1) Look at Overview section about usage of STL and Boost libraries

Technical Note

As VCET loads the requested basefile, some external resource files may need to be located, which requires searching and possibly downloading. By default, VCET may search folders such as the basefile's directory, VCET's installation folder and XRefPaths provided by the user in the INI profile.

For applications that wish to extend VCET's resource location (beyond the INI XRefPaths customization), a callback mechanism is provided. This callback is invoked every time a resource is requested and allows the application to write their own file location algorithm that better suites their needs.

To activate the extended resource location mechanism, the application should provide a **PanUserCallback**, a members of **PAN_CtlSetFile** structure, with **pResourceLocateProc** filled when sending the **PM_CTLSETFILEEX** message.

The **PanUserCallback** holds two useful members: the callback function pointer and a void pointer that can be used to hold any user data. The following is the callback function's prototype:

```
typedef  bool ( * ResourceLocateProc )(     std::(1)vector<csi::IPanResourceInfo *> & vResourceInfos,
                                            const IResourceContextInfo & resourceContext,
                                            bool * pContinue);
```

ResourceInfos:
The **IPanResourceInfo** vector provides information about every resource that needs to be located. Calling the **Search()** and **Download()** methods will give clear indication to what needs to be done with the requested resource.

ResourceContext:
The resource context provides information about the environment that the resource is to be located in. This context includes that basefile's folder path, the INI profile path and the user data (void pointer) that was initiallyinitialy provided to the **PM_CTLSETFILEEX** message.

Continuation:
pContinue is a boolean pointer that can be used to indicate whether VCET should perform it's location. The location process has three stages:

- Pre-locate: the application may choose to locate the resource directly without VCET's assistance.
- Core-locate: VCET performs the file searching if pre-locate didn't yield any results.
- Post-locate: the application has the opportunity to find any resources that were not found in the pre- or core-locate stages.

Following these three steps, pContinue controls the the flow of the location process. In pre-locate, the boolean pointer is valid and may receive a value of **true** (default) indicating that VCET should continue and locate the remaining missing resources, effectively enabling the core- and post-locate stages. A value of **false** will stop the location process and will leave the resources' locations dependent on pre-locate's results. This could be used when all the external resources are known to be at a certain location.

Once the core-location procedure has run, post-locate will be invoked on the resources that were not found. This stage is indicated by a **NULL** pointer in place of pContinue. This is the last chance for the resource to be found as VCET will continue loading the basefile regardless of the resource location. In some cases, substitutions may be used to replace the missing resources, which may affect the display of the file.

(1) Look at Overview section about usage of STL and Boost libraries

Locating Resources

If the application chooses to locate the resources itself, the **IPanResourceInfo** interface can be used to get the required information and set the file locations.

The resource requirements can be read from **csi::IPanResourceInfo**'s **Search()** and **Download()** functions. More often than not, both search and download will be needed in the same callback invocation.

The following is the information needed to perform the search.

- The file path indicating the wildcard expression that must match the external file name without the folder structure. The file path can be retrieved by calling the **csi::IPanResourceInfo::Path()** method.
- The pattern accompanies the file name and acts as an added filter. It is a regular expression that needs to match the located file name. It can be retrieved by invoking the **csi::IPanResourceInfo::Pattern()** function.
- The magic string is a sequencedescription of bytes that must match the first characters inside the located file. The magic string is returned by **csi::IPanResourceInfo::MagicString()**.the interface methods.
- The search paths list the folders that need to be searched. They may contain wildcard characters with the addition of the double asterisk. The search paths can be obtained from the **csi::IPanResourceInfo::SearchPaths()** method.
  It should be noted that the searching should stop after every search path and return if there is at lease one file found. A wildcard or recursive search path should be fully searched before returning the results.
- The maximum number of results is used to limit the search. It should be used as an upper bound on the number of files that are found matching the above criteria. This limit can be read from the **IPanResourceInfo::MaxResults()** method.

After the search is complete and the requested files are found, the file locations need to be added through the **csi::IPanResourceInfo::SetLocated** method that receives a vector of **csi::IPanResourceInfo::Location**s.

If the files were found in a folder not accessible to VCET, the resources will need to be copied into a local folder for parsing. This downloading process is required only on the files that have been selected for parsing. The **csi::IPanResourceInfo::ISelection** provides a way to set the downloaded, local file path for the selected files. Its iterator can be obtained from the **csi::IPanResourceInfo::GetSelected()** method.

PM_CTLSETMODE

Purpose Modifies the modes of a control.

Description If *wParam* is zero, enable the modes of the control specified in the mask, i.e., disable any mode which is not specified in the mask. If *wParam* is non-zero, enable all the modes of the control. The mask consists of zero, one or more of the following constants ORed together:

PAN_CTLMODEOPAQUE	Control is opaque
PAN_CTLMODENOREDRAW	Redraws are disabled (same effect as **WM_SETREDRAW**)
PAN_CTLMODEEXCESSSCROLL	Scrolling is not constrained (i.e., limited to the extents of the raster or vector image), allowing the control to scroll past the image limits.
PAN_CTLMODEANISOTROPIC	Aspect ratio is not preserved
PAN_CTLMODEDRAGDROP	Control accepts dropped files
PAN_CTLMODEINTERRUPTIBLE	Enables background processing for the control, allowing display operations to be interrupted while rendering. Please refer to technical note below.
PAN_CTLMODEMONOCHROME	Draw all entities in black. Only supported by the vector control.
PAN_CTLMODEPRESERVECLIP	Preserve clip region of target device.
PAN_CTLMODEPRESERVEPALETTE	Preserve palette of target device.
PAN_CTLMODEIGNOREMINMARGINS	Ignore printing margins.
PAN_CTLMODERENDERSELECTED	Only render selected entities.
PAN_CTLMODERENDERTOPRINTER	Give the same render output as a printer even the DC is not a printer.
PAN_CTLMODELIMITTOONETILE	Limit to one tile in printing.
PAN_CTLMODE_HIGHLIGHT_DIMMED	Dimmed highlight mode.
PAN_CTLMODE_DISABLE_FORCETOBLACK_ONRASTER	Prevents the Force-To-Black from being applied to raster overlays(images) in vector/raster overlays.
PAN_CTLMODE_TILE	CMF tile mode.
PAN_CTLMODE_FORCE_BKG_PAINT	Paint in force-to-black mode.

NOTE: The current implementation of the raster control does not support transparent or anisotropic modes when displaying monochrome images.

Parameters *wParam*: zero or non-zero

lParam: **(DWORD)** mask of mode flags

Returns: error code

Compatibility: all control types

Example:

```
void ToggleMode(HWND ctrlHandle, DWORD flg)
{
        DWORD cflg;
```

```
        BOOL    m_aniso,m_drag,m_exscr,m_noredr,m_opaq;

        if (ctrlHandle == NULL) return;

        // Get current mode flags
        SendMessage(ctrlHandle, PM_CTLGETMODE, 0, (LPARAM) &cflg);
        m_aniso  = (cflg&PAN_CTLMODEANISOTROPIC) != 0;
        m_drag   = (cflg&PAN_CTLMODEDRAGDROP) != 0;
        m_exscr  = (cflg&PAN_CTLMODEEXCESSSCROLL) != 0;
        m_noredr = (cflg&PAN_CTLMODENOREDRAW) != 0;
        m_opaq   = (cflg&PAN_CTLMODEOPAQUE) != 0;

        // Toggle flags specified by user
        if (m_aniso) cflg |= PAN_CTLMODEANISOTROPIC;
        else cflg &= ~PAN_CTLMODEANISOTROPIC;

        if (m_drag) cflg |= PAN_CTLMODEDRAGDROP;
        else cflg &= ~PAN_CTLMODEDRAGDROP;

        if (m_exscr) cflg |= PAN_CTLMODEEXCESSSCROLL;
        else cflg &= ~PAN_CTLMODEEXCESSSCROLL;

        if (m_noredr) cflg |= PAN_CTLMODENOREDRAW;
        else cflg &= ~PAN_CTLMODENOREDRAW;

        if (m_opaq)   flg |= PAN_CTLMODEOPAQUE;
        else cflg &= ~PAN_CTLMODEOPAQUE;

        // Now set with the new mask
        SendMessage(ctrlHandle, PM_CTLSETMODE, 0, (LPARAM) cflg);
}
```

Technical Note

It is important that developers who enable the **PAN_CTLMODEINTERRUPTIBLE** flag on a control be aware of issues related to background processing under Windows. Windows 3.x operates under a single system-wide message queue. As a result, while one task is processing a message, all other applications "block", and do not recieve any messages until this task has finished processing its message and returns control to Windows. As a result messages that take a long time to process (e.g., *s*earches in a database application, reading and decompressing large graphics images, long mathematical calculations, etc.) will temporarily control the CPU, and other tasks will have to wait for the message to complete.

In Windows, background processing is normally implemented by inserting:

```
if (PeekMessage(...)) {
GetMessage(...);
DispatchMessage(...);
}
```

into the handling routines for messages that take a lot of time to process.

The Controls provide this as an option if the **PAN_CTLMODEINTERRUPTIBLE** mode is set.
This allows users, for example, to do operations such as:

1- zoom/pan/scroll an image while it is displaying, without having to wait for the display to finish.
2- The reading of a file (e.g., a JPEG or CAD drawing) can be interrupted before finished, then another file can be displayed using the **PM_CTLSETFILE** message.
3- The application can also destroy a control while it is reading a file.

The first case is handled, transparently to the application, by the control. The second and third cases *require certain modications on the application's side.* Note that these modifications are required *only* if the **PAN_CTLMODEINTERRUPTIBLE** mode is enabled.

To handle the second case (dropping a file while reading), the application must:

1- Check the status of the control
2- If the control is idle then set the file with **PM_CTLSETFILE**, and proceed normally.
3- If the control is not idle then
    3a- Set the file with **PM_CTLSETFILE**
    3b- Post yourself the message to set the file and return immediately.
        Steps 1..3 will be automatically repeated when Windows processes the message.

The code extract on the following page illustrates this:

```
#define  UM_SHOWFILE       WM_USER + 10
HWND        hwndCtl;          /* Handle of the active Control */

LPARAM     CALLBACK
WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
     switch (msg) {
          case WM_SIZE: // ....
               break;
          case WM_PAINT: // ....
               break;
          // ....etc...
       case UM_SHOWFILE:
          {{{
          /*
          **  UM_SHOWFILE is a private message used by the application.
          **  wParam:  Not used
          **  lParam:  (LPCSTR) fname:  String pointer to the file
          **           to display.
          **               If lParam is 0, then the message had been internally                    **
     generated when the control was busy.
          **  return:       0
          */

          DWORD                dwStatus = 0L;
          char            fname[_MAX_PATH];

          //  If the Control is busy, try later.
          SendMessage(     m_hPanCtl, PM_CTLGETSTATUS,
                           (WPARAM)-1, (LPARAM)&dwStatus);
          if (dwStatus && lParam) {
               /*
               **  Keep a static variable so that the value is not lost when
               **  the variable goes out of context.
               */
               static      char fname[_MAX_PATH];
               strcpy(fname, (LPSTR) lParam);
               /*
               **  Set the file
               */
               SendMessage(hwndCtl, PM_CTLSETFILE, (WPARAM)-1,
     (LPARAM)fname);
               PostMessage(hWnd, UM_SHOWFILE, 0, 0L);
               return(0);
          } else if (lParam) {
               /*
               **  Set the file
               */
               SendMessage(hwndCtl, PM_CTLSETFILE, (WPARAM)-1,
     (LPARAM)lParam);
          }
          /*
          **  Get the name of the file. If lParam is 0, then get the file
          **  name from the control.
          */
          if (lParam) {
```

```
            _fstrcpy(fname, (LPCSTR)lParam);
        } else {
        PAN_CtlFileInfo fileInfo;
            _fmemset(&fileInfo, 0, sizeof(fileInfo));
            SendMessage(hwndCtl, PM_CTLGETFILE, (WPARAM)0,
    (LPARAM)&fileInfo);
            _fstrcpy(fname, fileInfo.name);
        }

        /*
        ** Proceed normally:
        */
        // ....
        return (0);
        }}}  // case UM_SHOWFILE
    }     // switch (msg)
}
```

The handling of the third case (destroying a control while reading) is similar to that of setting a new file while reading, except the the modifications are made in the handling of the **WM_CLOSE** message. The principle of operation remains the same:

1- Check the status of the control
2- If the control is idle then send the control the **PM_CTLDESTROY** message and proceed normally in the destruction of the window.
3- If the control is not idle then
    3a- Send the **PM_CTLDESTROY** message
    3b- Post yourself the **WM_CLOSE** message and immediately return.
        Steps 1..3 will be automaitcally repeated when Windows processes the posted message.

The code extract on the following page illustrates:

```
HWND        hwndCtl;            /* Handle of the active Control */

LPARAM    CALLBACK
WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
     switch (msg) {
       case WM_SIZE:
            ....
            break;
       case WM_PAINT:
            ....
            break;
            ....etc...
       case WM_CLOSE:
            {{{
            DWORD                dwStatus = 0L;


            SendMessage(m_hPanCtl, PM_CTLGETSTATUS, 0, (LPARAM)&dwStatus);

            if (hwndCtl && IsWindow(hwndCtl) && GetWindowLong(hwndCtl, 0)) {
                 SendMessage(hwndCtl, PM_CTLGETSTATUS, 0, (LPARAM)&dwStatus);
                 SendMessage(hwndCtl, PM_CTLDESTROY, (WPARAM)hCtl, 0);
            }

            if (! dwStatus) {
                 /*
                 ** If (control is idle)
                 **          && (no control || control has been destroyed)
                 */
                 if (hwndCtl && IsWindow(hwndCtl)) {
                      /*
                      **  Let Windows destroy the control
                      */
                      DestroyWindow(hwndCtl);
                 }
                 return TRUE;
            } else {
                 /*
                 **  If the Control is busy, try later.
                 */
                 PostMessage(WM_CLOSE,0, 0L);
                 return FALSE;
            }

            }}}  // case WM_CLOSE
     }     // switch (msg)
}
```

PM_CTLSETOPTION

Purpose Sets different control options.

Description Allows setting of different control options. The value of *wParam* must be one of the following options:

PAN_ CTLCADOPTIONS Set file specific CAD options. The CAD options are used to Enable/Disable some features (Text, Line-Styles, Filling…). This option is supported by Vector Control (2D/3D).

PAN_ CTLOPTDIMLEVEL Dim level, in case dim highlight mode has been selected, controls how much the rest of the drawing dimmed with respect to the highlighted items.

NOTE: If the *lParam* is a NULL pointer, the default dim level value is set. The message is handled only when the file is opened and a page of the relevant control is active. **E.g.:** The dim level is settable only if a vector file is open in VCET and a 2D page is active.

Parameters *wParam*: **PAN_ CTLCADOPTIONS** or **PAN_ CTLOPTDIMLEVEL**

lParam: (**wParam = PAN_CTLCADOPTIONS)**
Mask of the following CAD options flags:

PAN_CADOPTIONS_NOTEXT: Disable the text in the model.
PAN_CADOPTIONS_NODIMENSIONS Disable dimensional primitives in the model.
PAN_CADOPTIONS_NOXREFS: Disable XRefs in the model.
PAN_CADOPTIONS_NOFILLS: Instructs the snapping code to consider snapping filled areas if this flag is not set.
PAN_CADOPTIONS_NOLINESTYLES: Disable Line-Styles on all the primitives of the model. They will be drawn using solid lines.
PAN_CADOPTIONS_NOLINEWEIGHTS: Don't consider the line thickness for model primitives.
PAN_CADOPTIONS_FASTDISPLAY: Enable fast display rendering optimizations: whenever is possible the original model primitives will be displayed using approximation in order to speed up the rendering.
PAN_CADOPTIONS_FULLDISPLAY: Disable any kind of rendering optimizations. However, if both full display and fast display flags are specified the fast display will have priority (i.e. the full display flag will be ignored in this case.)

(wParam = PAN_OPTDIMLEVEL)
(double*) pointer to the desired dim level.

Returns: error code

Compatibility: vector-2D

Example:

```
/* Set a new dim level */
double newDimLevel = 0.7;
WORD wParam = PAN_CTLOPTDIMLEVEL;
DWORD lParam = (DWORD)&newDimLevel;

…
SendMessage (hwndCtl, PM_CTLSETOPTION, wParam, lParam);
…
```

Image-Related Command Messages

PM_CTLCONVERT

Purpose Obtain handles to conversion related functions

Description The control facilitates conversion from one format to another. This message allows the application to obtain the functions related to conversion. For each allowed *wParam*, a different conversion-related function is assigned to *lParam*.

Parameters *wParam*: Handle specifier. One of:
GET_PAFSEXPORT_IDENTIFYIMAGE_HANDLE
GET_PAFSEXPORT_QUERYIMAGE_HANDLE
GET_PAFSEXPORT_IDENTIFY_HANDLE
GET_PAFSEXPORT_QUERYFORMAT_HANDLE
GET_PAFSEXPORT_QUERYSUBFORMAT_HANDLE
GET_PAFSEXPORT_CONVERTFILE_HANDLE
GET_PAFSEXPORT_CONVERT_HANDLE
GET_PAFSEXPORT_OPTIMIZEPALETTE_HANDLE
GET_PAFSEXPORT_REDUCECOLORS_HANDLE

lParam: **(FARPROC *)** receives pointer to specified function

Returns: error code

Compatibility: all control types

Comments:

The functions that are retrieved by this function can be used to convert a file to a raster or vector output file.

GET_PAFSEXPORT_IDENTIFYIMAGE_HANDLE
Returns a pointer to the following function:

BOOL PAFSExport_IdentifyImageDLL(LPWORD *lpNumDlls***);**

This function returns the number of encoder DLLs in the variable pointed to by lpNumDlls.

GET_PAFSEXPORT_QUERYIMAGE_HANDLE
Returns a pointer to the following function:

BOOL PAFSExport_QueryImageDLL(WORD *index***,**
LPWORD *lpType***, LPSTR** *lpDesc***);**

This function returns the type and description of the image processing DLL that is associated with the specified index.

GET_PAFSEXPORT_IDENTIFY_HANDLE
Returns a pointer to the following function:

BOOL PAFSExport_Identify(LPWORD *wFormats***,**
LPWORD *wSubFormats***, LPWORD** *wOutputType***);**

When *wFormats* is NULL, this function returns the list of sub-formats and/or the list of output types for all export filters. When *wFormats* is not NULL and both other parameters are NULL, then the number of exporters is returned in *wFormats*. When

wFormats is not NULL and at least one other parameter is not NULL, then the sub-format and/or output type for the specified format is returned in *wSubFormats* and/or *wOutputType* respectively.

GET_PAFSEXPORT_QUERYFORMAT_HANDLE
Returns a pointer to the following function:

BOOL PAFSExport_QueryFormat(LONG *ID*, **LPCVTPANX** *cvcbs*, **DWORD** *dwFormat*, **LPSTR** *szModule*, **LPSTR** *szDesc*);

This function returns the name of the export filter and the description of the output format in *szModule* and *szDesc*, respectively, for the format indexed by *dwFormat*.

GET_PAFSEXPORT_QUERYSUBFORMAT_HANDLE
Returns a pointer to the following function:

BOOL PAFSExport_QuerySubFormat(LONG *ID*,
LPCVTPANX *cvcbs*, **DWORD** *dwFormat*, **DWORD** *dwSubFormat*, **LPSTR** *szDesc*,
LPSTR *szFilter*, **LPSTR** *szExt*, **LPWORD** *numDepths*, **LPWORD ****wColorDepths*,
LPDWORD *dwOutputCaps*, **LPWORD** *wUnits*, **REAL ****Width*, **REAL ****Height*);

This function returns the characteristics associated with the specified format, *dwFormat*, and sub-format, *dwSubformat*.

GET_PAFSEXPORT_CONVERTFILE_HANDLE
Returns a pointer to the following function:

BOOL PAFSExport_ConvertFile(LONG *ID*, **LPCVTPANX** *cvcbs*, **DWORD** *dwFormat*, **DWORD** *dwSubFormat*, **LPCSTR** *szInFileName*, **LPCSTR** *szOutFileName*);

This function converts the input file, *szInFileName*, to the specified format, *dwFormat* and *dwSubformat*, and outputs to the specified output file, *szOutFileName*.

GET_PAFSEXPORT_CONVERT_HANDLE
Returns a pointer to the following function:

BOOL PAFSExport_Convert(LPCVTPANX *cvcbs*, **DWORD** *dwFormat*, **DWORD** *dwSubFormat*, **LPCSTR** *szInFileName*, **LPCSTR** *szOutFileName*);

This function calls **PAFSExport_ConvertFile**, but will first render the file if needed.

GET_PAFSEXPORT_OPTIMIZEPALETTE_HANDLE
Returns a pointer to the following function:

BOOL PAFSExport_OptimizePalette(LONG *index*,
HANDLE *hDIB*, **LPRGBQUAD** *lpColors*, **WORD** *nColors*);

This function constructs a color table with at most *nColors* based on the technique specified by *index* and the colormap associated with *hDIB*. The resulting color-map is returned in the array pointed to by *lpColors*.

GET_PAFSEXPORT_REDUCECOLORS_HANDLE
Returns a pointer to the following function:

BOOL PAFSExport_ReduceColors(LONG *index*,
HANDLE *hDIBIn*, HANDLE *hDIBOut*);

This function maps the pixels stored in *hDIBIn* to correspond with the colors associated with *hDIBOut*. The method used is specified by *index*.

PM_CTLFLIP

Purpose Flips the contents of the control.

Description Flip the contents of the control depending on the given flag.

Parameters

wParam	**(WORD)** one of: **PAN_CTLFLIPNONE** **PAN_CTLFLIPX** **PAN_CTLFLIPY** **PAN_CTLFLIPXY**
lParam	not used

Returns: error code

Compatibility: raster and vector-2D and controls

Examples:

```
int FlipInX(HWND ctrlHandle)
/* This function will flip the contents of the control along the X axis */
{
        if (ctrlHandle == NULL) return PAN_CTLERRMISC;
        return (int) SendMessage (ctrlHandle, PM_CTLFLIP, PAN_CTLFLIPX, 0);
}

int FlipInY(HWND ctrlHandle)
/* This function will flip the contents of the control along the Y axis */
{
        if (ctrlHandle == NULL) return PAN_CTLERRMISC;
        return (int) SendMessage(ctrlHandle, PM_CTLFLIP, PAN_CTLFLIPY, 0);
}

int FlipInXY(HWND ctrlHandle)
{
        if (ctrlHandle == NULL) return PAN_CTLERRMISC;
         return (int)SendMessage(ctrlHandle, PM_CTLFLIP, PAN_CTLFLIPXY, 0);
}

int FlipResetHWND ctrlHandle)
/* This function will reset the contents of the control to its original orientation */
{
        if (ctrlHandle == NULL) return PAN_CTLERRMISC;
        return (int)SendMessage(ctrlHandle, PM_CTLFLIP, PAN_CTLFLIPNONE, 0);
}
```

PM_CTLGETFLIP

Purpose Retrieves the current flipping state.

Description Returns the flipping mode that is currently set in the control

Parameters

wParam	not used	
lParam	**(int *)**	flip mode, one of: **PAN_CTLFLIPNONE** **PAN_CTLFLIPX** **PAN_CTLFLIPY** **PAN_CTLFLIPXY**

Returns: error code

Compatibility: raster and vector-2D and controls

Examples:

```
int GetFlip(HWND ctrlHandle)
/* Sample code that retrieves the flipping mode */
{
        int flip = 0;
        SendMessage(ctrlHandle, PM_CTLGETFLIP, 0, &flip);
        return flip;
}
```

PM_CTLGETROTATION

Purpose Retrieves the current rotation state.

Description Returns the current rotation angle in degrees set in the control.

Parameters

wParam	not used	
lParam	**(int *)**	rotation angle. Possible values are 0, 90, 180 and 270.

Returns: error code

Compatibility: raster and vector-2D and controls

Examples:

```
int GetRotation(HWND ctrlHandle)
/* Sample code that retrieves the rotation angle */
{
        int rotation  = 0;
        SendMessage(ctrlHandle, PM_CTLGETROTATION, 0, & rotation  );
        return rotation;
}
```

PM_CTLGETZOOM

Purpose Returns the current zoom factor.

Parameters *wParam*: **(WORD)** zoom flag, one of:
PAN_CTLZOOMX
PAN_CTLZOOMY
PAN_CTLZOOMBOTH

lParam: **(LPDWORD)** zoom factor in 1000th's of a percent

Returns: error code

Compatibility: all control types

Example:

```
void ShowZoomFactors(HWND ctrlHandle)
/*      Display the current zoom factor of the control
        on the output device. */
{
        DWORD fact[2];

        if (ctrlHandle == NULL) return;
        SendMessage(ctrlHandle, PM_CTLGETZOOM,
                PAN_CTLZOOMX, (LPARAM) &fact[0]);
        SendMessage(ctrlHandle, PM_CTLGETZOOM,
                PAN_CTLZOOMY, (LPARAM) &fact[1]);

        Output("zoom X  : (%f)%%\n", (float)(fact[0]/1000.0));
        Output("zoom Y  : (%f)%%\n", (float)(fact[1]/1000.0));
}
```

PM_CTLPAINT, PM_CTLSIZE, PM_CTLHSCROLL, PM_CTLVSCROLL

Compatibility: all control types

Identical to the Windows messages **WM_PAINT, WM_SIZE, WM_HSCROLL,** and **WM_VSCROLL,** respectively. Unlike the Windows messages, however, the control-specific messages are always processed, whether or not the control is visible or manages resize and scroll events directly.

The most common situations in which these messages would be used are when hidden controls are being used to format the contents of a file for printing or copying to the clipboard.

PM_CTLROTATE

Purpose Rotates the contents of the control.

Description Rotate the contents of the control by the given angle in degrees.

Parameters

wParam: **(WORD)** rotation angle in degrees.
Possible values are 0, 90, 180 and 270.

lParam: not used

Returns: error code

Compatibility: vector 2D and raster controls

Example:

```
void Rotate(HWND ctrlHandle, WORD deg)
{
	if (ctrlHandle == NULL) return;
	SendMessage(ctrlHandle, PM_CTLROTATE, deg, 0);
}
```

PM_CTLSETZOOM

Purpose Sets the zoom factor.

Parameters

wParam: **(WORD)** zoom flag, one of:
PAN_CTLZOOMX
PAN_CTLZOOMY
PAN_CTLZOOMBOTH

lParam: **(DWORD)** zoom factor in 1000th's of a percent

Returns: error code

Compatibility: All

Special Notes When the control is in isotropic mode, **PAN_CTLZOOMX** and **PAN_CTLZOOMY** flags force the other zoom direction to adjust preserving the X-Y ratio. To change the Raster and Vector-2D controls' isotropic mode, send the **PAN_CTLMODEANISOTROPIC** flag using the **PM_CTLSETMODE** message.

Example:

```
void SetZoom(HWND ctrlHandle, double zoom, WORD flag)
/* Set the zoom factor. Zoom is in % so a value of 250 corresponds to a 250% zoom.  Flag = one of
PAN_CTLZOOMX, PAN_CTLZOOMY, PAN_CTLZOOMBOTH.*/
{
        DWORD fct;

        if (ctrlHandle == NULL) return;

        // The message is sent with the zoom percentage multiplied by 1000.
        fct = (DWORD) (zoom * 1000);
        SendMessage(ctrlHandle, PM_CTLSETZOOM, flag, (LPARAM)fct);
}
```

Coordinates-Related Command Messages

PM_CTLCARETTOWORLD

Purpose Converts a caret based position to a view position.

Description Text based operations (searches in particular) tend to return their results in caret based coordinates (page, flow, character offset). To perform view manipulations, world coordinates are required. This message allows the conversion of caret positions to view positions. The return value is a pointer to a static buffer in the control, from which the coordinates can be copied; or **NULL** if the translation could not be done (non-existent caret position).

Parameters *wParam*: not used

lParam: **(struct PAN_CtlCaretPos *)** lpCaretPos;

Returns: **(struct PAN_CtlPos *)** lpViewPos

Compatibility: archive, database, document, spreadsheet.

Example:

```
int GetViewPos(HWND ctrlHandle, struct PAN_CtlCaretPos *lpCaretPos,
                struct PAN_CtlPos *lpPos;)
/* Convert the caret position into a view coordinate. */
{
        int err;
        PAN_CtlPos *lpTmpPos;

        if (ctrlHandle == NULL)
                return PAN_CTLERRMISC;

        lpTmpPos = SendMessage(ctrlHandle, PM_CTLCARETTOWORLD, 0,
                                                (LPARAM) lpCaretPos);

        if (lpTmpPos != (struct PAN_CtlPos *) NULL) {
                err = PAN_CTLERRNONE;
                *lpPos = *lpTmpPos;
        } else {
                err = PAN_GetCtlErrorCode();}

        return (err);
}
```

PM_CTLCLEARSELS

Purpose Clears all selections.

Description Clear all selections in the current file.

Parameters *wParam*: not used

lParam: not used

Returns: error code

Compatibility: all control types

Example:

```
int ClearSelections(HWND ctrlHandle)
/* Clear all the selections in the control */
{
        int err;

        if (ctrlHandle == NULL)
                return PAN_CTLERRMISC;

        err = (int) SendMessage(ctrlHandle, PM_CTLCLEARSELS, 0, 0);
        return err;
}
```

PM_CTLCLIENTTOWORLD

Purpose Performs a client to world coordinates conversion.

Description Returns the world coordinates corresponding to the given client area coordinates (i.e., relative to the top, left corner of the client area of the control window).

Parameters *wParam*: not used

lParam: **(PAN_CtlPos *)** client coordinates

Returns: error code

Compatibility: all control types

Example:

```
int ClientToWorld(HWND ctrlHandle)
/*      display the world coordinates on the output device. */
{
        PAN_CtlPos cc;
                // structure to hold the returned value,
                // this must be initialized with the values to convert
        int     err;

        if (ctrlHandle == NULL) return PAN_CTLERRMISC;

        err = SendMessage(ctrlHandle, PM_CTLCLIENTTOWORLD, 0, (LPARAM) &cc);
        if (err != PAN_CTLERRNONE)
                return err;

        Output("CLIENTTOWORLD: pt1 = (%f, %f, %f), pt2 = (%f, %f, %f)\n",
                        cc.pt1.x, cc.pt1.y, cc.pt1.z,
                        cc.pt2.x, cc.pt2.y, cc.pt2.z);

        return PAN_CTLERRNONE;
}
```

PM_CTLGETCARETPOS

Purpose Returns the current position of the caret within a text stream.

Description This message allows an application to obtain the current position of the caret in the control.

Parameters *wParam*: not used

lParam: **(PAN_CtlCaretPos *)** caret position

Returns: error code

Compatibility: archive, database, document, spreadsheet.

Example:

```
void ShowCaretOffset(HWND ctrlHandle)
/*      Display the offset of the file on the output device. */
{
        int             err;
        PAN_CtlCaretPos CaretPos;

        if (ctrlHandle == NULL) return;

        err = (int) SendMessage(PM_CTLGETCARETPOS, 0,
                (LPARAM) ((PAN_CtlCaretPos *) &CaretPos));

        if (err != PAN_CTLERRNONE) {
                Output("Couldn't get caret position\r");
        } else {
                Output("CaretPos = (%d, %ld, %ld)\n",
                CaretPos.page, CaretPos.flow, CaretPos.offset);
        }
        return;
}
```

PM_CTLSETCARETPOS

Purpose Sets the caret to the specified position

Description The position is specified as an offset within a flow on a specified page.

Parameters *wParam*: not used.

lParam: **(PAN_CtlCaretPos *)** lpCaretPos;

Returns: error code

Compatibility: archive, database, document, spreadsheet.

Example:

```
void HomeCursor(HWND ctrlHandle)
/* Move the caret to the start of the information stream in the current */
/* information stream and page. */
{
        int err;
        PAN_CtlCaretPos curPos;

        if (ctrlHandle == NULL) return;

        err = SendMessage(ctrlHandle, PM_CTLGETCARETPOS, 0,
                (LPARAM) ((PAN_CtlCaretPos *) &curPos));

            if (err != PAN_CTLERRNONE) {
                Output("Couldn't get current caret position.\n");
                return;
                }

                curPos.offset = 0;

        SendMessage(ctrlHandle, PM_CTLSETCARETPOS, 0,
                (LPARAM) ((PAN_CtlCaretPos *) &curPos));
}
```

PM_CTLWORLDTOCARET

Purpose Converts a view based position to a caret position.

Description This message allows the caller to convert a view based coordinate into a caret based position.

Parameters *wParam*: not used

lParam: **(struct PAN_CtlPos *)** lpViewPos

Returns: **(struct PAN_CtlCaretPos *)** lpCaretPos;

Compatibility: archive, database, document, spreadsheet.

Example:

```
void ShowCaretPos(HWND ctrlHandle, struct PAN_CtlPos *lpPos)
/* Display the caret position that corresponds to the specified view coordinate. */
{
        PAN_CtlCaretPos *lpCtPos;
        int err;

        if (ctrlHandle == NULL)
        return;

        *lpCtPos = SendMessage(ctrlHandle,
        PM_CTLWORLDTOCARET, 0, (LPARAM) lpPos);

        if (*lpPos != (struct PAN_CtlPos *) NULL) {
                Output("Coordinate = page %d, flow %ld, offset %ld\n",
                        lpCtPos->page, lpCtPos->flow, lpCtPos->offset);
                }
}
```

PM_CTLWORLDTOCLIENT

Purpose Performs a world to client coordinates conversion.

Description Returns the client area coordinates (i.e., relative to the top, left corner of the client area of the control window) corresponding to the given world coordinates.

Parameters *wParam*: not used

lParam: **(PAN_CtlPos *)** world coordinates

Returns: error code

Compatibility: all control types

Example:

```
void ShowWorldToClient(HWND ctrlHandle)
/* Display on the output device (see the Output() function). */
{
        PAN_CtlPos cc;
        Send(PM_CTLCLIENTTOWORLD, 0, (LONG) &cc);
        Output("CLIENT COORD x=%d y=%d\r", cc.x, cc.y);
}

//-------------------------------------------------------

// Function used by several code examples in this manual.
void Output(char* fmt, ...)
{
        va_list args;
        static  char buff[1025];

        strcpy(buff,""); /* Format message string. */
        va_start(args, fmt);
        vsprintf(buff, fmt, args);
        va_end(args);

        /* Write code to send the buffer to the desired output device*/
}
```

PM_CTLXFRMRECT

Purpose Apply the current control rotation and flipping transform.

Description Transforms a world coordinate range by applying current rotation/flipping. Can also perform the inverse operation i.e. Undo rotation/flip transformation..

Parameters

wParam: Controls if we are applying a direct or inverse (undo) transformation.

XFRM_FNC	Apply transformation
XFRM_FNC_RECIPROCAL	Undo transformation.

lParam: **(PAN_CtlRange *)** world coordinates.

Returns: error code

Compatibility: vector and raster controls.

Example:

```
/* Apply rotation and filling to a world coordinates rectangle */

PAN_CtlRange rect = …
SendMessage (hwndCtl, PM_CTLXFRMRECT, XFRM_FNC, (LPARAM)&rect);
```

PM_CTLCOPY

Purpose Copy the current selections to the clipboard.

Description Copy the current selections to the clipboard using the clipboard formats specified in the given mask of **PAN_CtlClpbrd** constants.

Parameters *wParam*: not used

lParam: **(DWORD)** format(s)

Returns: error code

Compatibility: all control types

Example:

```
void CopyToClpBoard(HWND ctrlHandle, DWORD clpbFlgs)
/*
Copy the control's contents to the clipboard using the clipboard formats specified by clpbFlgs. First we clear all the
        selections, then get the current page size and send the message to copy.*/
{
        PAN_CtlRange  rg;       // structure to hold the page extents
        DWORD                   pg;     // page number

        if (ctrlHandle == NULL) return;

        SendMessage(ctrlHandle, PM_CTLGETPAGE, 0, (LPARAM) &pg);
        SendMessage(ctrlHandle, PM_CTLGETPAGESIZE, pg, (LPARAM) &rg);
        SendMessage(ctrlHandle, PM_CTLCLEARSEL, 0, 0);
        SendMessage(ctrlHandle, PM_CTLSETSEL, TRUE, (LPARAM) &rg);
}
```

PM_CTLGETCLPBRDFMTS

Purpose Returns clipboard formats supported by a control.

Description Return a description of each clipboard format supported by the control in the **PAN_CtlClpbrdFmtList** structure pointed to by *lParam*.

Parameters

wParam:	not used		
lParam: formats	**(PAN_CtlClpbrdFmtList*)**	supported clipboard	

Returns: error code

Compatibility: all control types

Example:

```
void ShowClpbrdFmts(HWND ctrlHandle)
/*      display the list of clipboard formats supported by the control. */
{
        PAN_CtlClpbrdFmtList fl;// pointer to the fmt list
        PAN_CtlClpbrdFmt * cf;   // pointer to the format structure
        WORD                            i;
        char                            desc[PAN_CTLMAXDESC];

        if (ctrlHandle == NULL) return;
        SendMessage(ctrlHandle, PM_CTLGETCLPBRDFMTS, 0, (LPARAM) &fl);
        cf = (PAN_CtlClpbrdFmt *) GlobalLock(fl.hFmts);
        for (i=0; i<fl.nFmts; i++) {
                Output("%s", cf[i]. desc);          // display format description
                }
        GlobalUnlock(fl.hFmts);
}
```

PM_CTLGETNUMSELS

Purpose Returns the current number of selections

Description Returns the current number of selections in the **WORD** pointed to by *lParam*. The number of selections, if any, is always one for vector controls, one for raster and one per page for document. Archive, database and spreadsheet controls can have more than one selection.

Parameters

	wParam:	not used
	lParam:	**(LPWORD)** number of selections

Returns: error code

Compatibility: all control types

Example:

```
void ShowNumSelection(HWND ctrlHandle)
/*                  Display the current number of selections on the output device. */
{
        DWORD nsels;

        if (ctrlHandle == NULL) return;

        SendMessage(ctrlHandle, PM_CTLGETNUMSELS, 0, (LPARAM) &nsels);
        Output("Number of selections=%d\n", nsels);
}
```

PM_CTLGETSELS

Purpose Returns a list of the current selections.

Description Return a list of the current selections for the current file in the **PAN_CtlSelList** structure pointed to by *lParam*.

Parameters

	wParam:	not used
	lParam:	**(PAN_CtlSelList *)** pointer to selections list

Returns: error code

Compatibility: all control types

Example:

```
void ShowSelections(HWND ctrlHandle)
/*      Display the list of current selections on the output device. */
{
        PAN_CtlSelList sl;
        PAN_CtlSel *    s;

        if (ctrlHandle == NULL) return;

        SendMessage(ctrlHandle, PM_CTLGETSELS, 0, (LPARAM) &sl);
        s = (PAN_CtlSel *) GlobalLock(sl.hSels);
        for (WORD i = 0; i < sl.nSels; i++) {
                switch (s[i].type) {
                case CTLSEL_VIEW:
                        Output(" (VIEW COORDS) = (%f, %f, %f) - (%f, %f, %f)\n",
                                s[i].range.vwRange.pt1.x,
                                s[i].range.vwRange.pt1.y,
                                s[i].range.vwRange.pt1.z,
                                s[i].range.vwRange.pt2.x,
                                s[i].range.vwRange.pt2.y,
                                s[i].range.vwRange.pt2.z);
                        break;

                case CTLSEL_CARET:
                        Output(" (CARET COORD) ");
                        Output("from = (page %d, flow %d, offset %ld), ",
                                s[i].range.ctRange.from.page,
                                s[i].range.ctRange.from.flow,
                                s[i].range.ctRange.from.offset);
                        Output("to = (page %d, flow %d, offset %ld)\n",
                                s[i].range.ctRange.to.page,
                                s[i].range.ctRange.to.flow,
                                s[i].range.ctRange.to.offset);
                        break;
                } /* switch */
        } // end for
}
```

PM_CTLSETSEL

Purpose Sets or removes a selection based on view coordinates.

Description If the selection flag is **TRUE**, set the given selection in the current file, otherwise, remove the given selection if it exists.

Parameters

wParam **(BOOL)** selection flag

lParam: **(const PAN_CtlRange *)** selection

Returns: error code

Compatibility: all control types

Example:

```
void SetSelectionRaster(HWND ctrlHandle, WORD x, WORD y, WORD w,
                                        WORD h)
{
        PAN_CtlRange rg;

        if (ctrlHandle == NULL)
                return;

        rg.min.x = x;
        rg.min.y = y;
        rg.min.z = 0.0;
        rg.max.x = x + w;
        rg.max.y = y + h;
        rg.max.z = 0.0;
        SendMessage(ctrlHandle, PM_CTLSETSEL, TRUE, (LPARAM)&rg);
}
```

PM_CTLSETSELCARET

Purpose Sets or removes a selection based on caret coordinates.

Description If the selection flag is **TRUE**, set the given selection in the current file; otherwise, remove the given selection if it exists.

Caret based selection within tabular data works as follows: The from and to flows are converted to (row, column) pairs. The selection is made of all cells between the two specified rows and two columns. Page numbers in the caret range are ignored when used for creating selections in tabular data. By using this technique, all table selections are rectangular regions.

Parameters *wParam*: **(BOOL)** selection flag

lParam: **(const PAN_CtlCaretRange *)** selection

Returns: error code

Compatibility: archive, database, document, spreadsheet.

Example:

```
void SetSelection(HWND ctrlHandle, PAN_CtlCaretPos pt1,
        PAN_CtlCaretPos pt2)
{
        PAN_CtlCaretRange rg;

        if (ctrlHandle == NULL)
                return;

        rg.from = pt1;
        rg.to = pt2;

        SendMessage(ctrlHandle, PM_CTLSETSELCARET, TRUE,
                        (LPARAM) &rg);
}
```

Color-Related Command Messages

PM_CTLGETFGBGCOLOR

Purpose Returns the foreground or background color.

Description If *wParam* is zero, return the current foreground color. If *wParam* is non-zero, return the current background color.

Parameters

wParam:	**0**	Foreground Request
	≠0	Background Request
lParam:		**(LPCOLORREF)** color

Returns: error code

Compatibility: all control types

Example:

```
typedef struct { char R,G,B; } RGB;

int GetFgBgColor(HWND ctrlHandle, RGB* fg, RGB* bg)
/* Return the foreground and background colors of the control */
{
        int err;
        COLORREF c;

        if (ctrlHandle == NULL || fg == NULL || bg == NULL)  return PAN_CTLERRMISC;

        // Get foreground color
        err=(int) SendMessage(ctrlHandle, PM_CTLGETFGBGCOLOR, 0, (LPARAM)&c);
        if (err != PAN_CTLERRNONE) return err;
        fg->R = GetRValue(c);
        fg->G = GetGValue(c);
        fg->B = GetBValue(c);

        // Get background color
        err=(int) SendMessage(ctrlHandle, PM_CTLGETFGBGCOLOR, 1, (LPARAM) &c);
        if (err != PAN_CTLERRNONE) return err;
        bg->R = GetRValue(c);
        bg->G = GetGValue(c);
        bg->B = GetBValue(c);

        return PAN_CTLERRNONE;
}
```

PM_CTLGETPALETTE

Purpose Returns palette information.

Description Return the color depth of the current file. If *lParam* points to a **LOGPALETTE** structure, the size of which corresponds to the number of colors, it is filled with the file's palette, if any.

Parameters *wParam*: not used

lParam: **(LOGPALETTE *)** logical palette

Returns: **(WORD)** color depth on success or -1 on failure

Compatibility: raster and vector-2D controls

Example:

```
void ShowPalette(HWND ctrlHandle, int ncol)
/*      Display the palette used by the displayed file. */
{
        LPLOGPALETTE palette;
        int err, indx;

        if (ctrlHandle == NULL)   return;

        palette = fmalloc(sizeof(LOGPALETTE + (255) * sizeof(PALETTEENTRY));
        if (palette == NULL) {
                Output("Couldn't allocate storage for palette\n");
                return;
                }

        err = SendMessage(ctrlHandle, PM_CTLGETPALETTE, 0, (LPARAM) palette);

        if (err == -1) {
                Output("Couldn't get palette\n");
                } else {
                for (indx = 0; indx <= palette->palNumEntries; indx++) {
                        Output("PAL[%d] = (%d, %d, %d)\n", indx,
                                palette->palPalEntry[indx].peRed,
                                palette->palPalEntry[indx].peGreen,
                                palette->palPalEntry[indx].peBlue);
                }
        }
}
```

PM_CTLPALETTECHANGED

Purpose The calling application must send this message to the control whenever it receives a **WM_PALETTECHANGED** message.

Description The window handle specified in *wParam* is the same window handle as that specified in *wParam* for the **WM_PALETTECHANGED** message. Handled identically as a **WM_PALETTECHANGED** message.

Parameters

wParam:	**(HWND)** Handle of the window which changed the palette.	
lParam:	not used	

Returns: error code

Compatibility: all control types

PM_CTLQUERYNEWPALETTE

Purpose The calling application must send this message to the control whenever it receives a **WM_QUERYNEWPALETTE** message. Handled identically to **WM_QUERYNEWPALETTE** message. Useful for debugging purposes.

Parameters

wParam: **(HWND)** Handle of window for which to realize the new palette.

lParam: not used

Returns: error code

Compatibility: all control types

PM_CTLSETFGBGCOLOR

Purpose Sets the foreground or background color.

Description Sets the foreground/background color. Can be done with or without updating the control window.

Parameters *wParam*:
0 Set foreground color and update window.
1 Set background color and update window.
2 Set foreground color without updating.
3 Set background color without updating.

lParam: **(const COLORREF *)** color

Returns: error code

Compatibility: all control types

Example:

```
// use these defines to initialize which for SetColor below
#define SET_FOREGROUND 0
#define SET_BACKGROUND 1

void SetColor(HWND ctrlHandle, char  r, char g, char b, int which)
{
	COLORREF color = RGB(r, g, b);

	if (ctrlHandle == NULL) return;
	SendMessage(ctrlHandle, (WPARAM) which, (LPARAM) &color);
}
```

PM_CTLSETPALETTE

Purpose Sets the color palette.

Description Set the color palette of the current file.

Parameters *wParam*: not used

lParam: **(LOGPALETTE *)** logical palette

Returns: error code

Compatibility: raster and vector-2D controls

Example:

```
extern LOGPALETTE *lp;

void Reset8Palette(HWND ctrlHandle)
{
	if (ctrlHandle == NULL) return;

	SendMessage(ctrlHandle, PM_CTLSETPALETTE, 0, lp);
}
```

Views-Related Command Messages

PM_CTLGETVIEW

Purpose Gets the "active named-view" being displayed by the control.

Parameters *wParam*: Not used

lParam: **(PAN_VIEW *)** current view

Returns: (**BOOL) FALSE** means no view defined, **TRUE** indicates valid view being used.

Compatibility: vector.

Example:

```
void ShowViewName(HWND ctrlHandle)
/*      Display the name of the view currently being viewed. */
{
        BOOL ViewStatus;
        PAN_VIEW curView;

        if (ctrlHandle == NULL) return;

        ViewStatus = SendMessage(ctrlHandle, PM_CTLGETVIEW, 0,
                        (LPARAM) (PAN_VIEW *) &curView);
        if (ViewStatus) {
                Output("Current view name = %s\r", curView.name);
        } else {
                Output("No named view currently defined\n");
        }
}
```

PM_CTLGETVIEWEXTENTS

Purpose Get the current view extents.

Description Returns the current view extents of the current file in view coordinates.

NOTE: In order to get the same values as were set with **PM_CTLSETVIEWEXTENTS**, the control's mode has to be set to anisotropic.

Parameters *wParam*: not used

lParam: **(PAN_CtlRange *)** extents

Returns: error code

Compatibility: all control types

Example:

```
/* Display the view extents of the current page on the output device */
void ShowViewExtents(HWND ctrlHandle)
{
        PAN_CtlRange rg;

        if (ctrlHandle == NULL) return;
        SendMessage(ctrlHandle, PM_CTLGETVIEWEXTENTS, 0, (LPARAM)                    &rg);
        Output("VIEWEXTENTS: min = (%f, %f, %f)\n", rg.min.x, rg.min.y,          rg.min.z);
        Output("VIEWEXTENTS: max = (%f, %f, %f)\n", rg.max.x, rg.max.y,          rg.max.z);
}
```

PM_CTLGETVIEWNAMES

Purpose Returns the buffer of "view" names set by **PANX_SetViews().**

Description Vector files can define view names which the decoder passes to the core using the **PANX_SetViews** callback. This message provides a means for the application of obtaining this information from the control.

Parameters *wParam*: **(int)** number of views to return

lParam: **(PAN_VIEW *)** view buffer

Returns: Number of views returned in buffer pointed to by *lParam*, 0 for now views, -1 on failure.

Compatibility: vector.

Example:

```
void ShowAvailViews(HWND ctrlHandle)
/*      Display a list of up to the first 64 available named views. */
{
        int numAvail;
        PAN_VIEW ViewBuf[64];

        if (ctrlHandle == NULL) return;

        numAvail = SendMessage(ctrlHandle, PM_CTLGETVIEWNAMES, 64,
                        (LPARAM) ((PAN_VIEW *) ViewBuf));
        if (numAvail <= 0) {
                Output("No named views available.\r");
        } else {
                int indx;
                for (indx = 0; indx < numAvail; indx++) {
                        Output("View %d = %s\r", indx, ViewBuf[indx].name);
                }
        }
}
```

PM_CTLSETVIEW

Purpose Sets the view to use in the control.

Description If a vector decoder supports different views, it can determine which view to render through the **PANX_GetView** callback. This message allows the application to choose the view that will be decoded.

If view number -2 is specified, the application is specifying a user-defined view that may be set through the view info parameter. View number -1 indicates that the default view should be used. Other view numbers should specify views that are defined within the file.

Parameters *wParam*: **(WORD)** view number

lParam: **(PAN_VIEW *)** view info

Returns: error code

Compatibility: vector.

Example:

```
void ShowView(HWND ctrlHandle, int newView, PAN_VIEW *nViewSpecs)
/*      Display the specified view. */
{
        int err;
        PAN_VIEW *oldViews;

        if (ctrlHandle == NULL) return;

        if (newView >= 0) {
                oldViews = fmalloc((newView + 1) * sizeof(PAN_VIEW));
                if (oldViews == (PAN_VIEW *) NULL) {
                        Output("Couldn't allocate space for views list\n");
                        return;
                }

        err = SendMessage(ctrlHandle, PM_CTLGETVIEWS, newView + 1,
        (LPARAM) oldViews);

        if (err != PAN_CTLERRNONE) {
                Output("Couldn't get specified view parameters\n");
                return;
                }

                *nViewSpecs = *(oldViews + newView);
        }

        err = (int) SendMessage(ctrHandle, PM_CTLSETVIEW, newView,
        nViewSpecs);

        if (err != PAN_CTLERRNONE) {
                Output("Couldn't set specified view\n");
                }
}
```

PM_CTLSETVIEWEXTENTS

Purpose Sets the view extents of the current file in view coordinates.

Description Set the view extents of the current file in view coordinates. If the width member is zero, then only the x, y (, z) members are used to position the top-left corner of the control window. This message can also be used to zoom in or out for controls with zooming capability.

Parameters *wParam*: not used

lParam: **(const PAN_CtlRange *)** extents

Returns: error code

Compatibility: all control types

Example:

```
typedef struct {
double m_x, m_y, m_width, m_height;
} EXTENTS;

void SetViewExtent(HWND ctrlHandle, EXTENTS dlg)
{
        PAN_CtlRange   rg;

        rg.min.x = dlg.m_x;
        rg.min.y = dlg.m_y;
        rg.min.z = 0.0;

        rg.max.x = dlg.m_x + dlg.m_width;
        rg.max.y = dlg.m_y + dlg.m_height;
        rg.max.z = 0.0;

        SendMessage(ctrlHandle, PM_CTLSETVIEWEXTENTS, 0,
                                (LPARAM) &rg);
}
```

Blocks-Related Command Messages

PM_CTLGETBLOCK

Purpose Gets the "active block" from the control.

Description Get the current "active block" for the current page and set it in the **long** pointed to by *lParam*.

Parameters

wParam: Not used

lParam: **(long *)** current block. Set to zero if the page contains no blocks or if the whole page is active.

Returns: error code

Compatibility: vector

Example:

```
void OutputCurBlock(HWND ctrlHandle)
/* Display the ID of the currently active block */
{
        long    activeBlock;

        if (ctrlHandle == NULL) return;

        int err = SendMessage(ctrlHandle, PM_CTLGETBLOCK, 0,
                        (LPARAM) &activeBlock);

        if (err != PAN_CTLERRNONE) {
                Output("Incompatible control\n");
        } else {
                Output("Current block = %ld\n", activeBlock);
        }
        return;
}
```

PM_CTLGETBLOCKNAMES

Purpose Returns the buffer of block names set by the decoder.

Description Get the names of the number of blocks specified in *wParam* and set them in the location specified by *lParam*.

Parameters *wParam*: **(WORD)** number of block names to retrieve.

lParam: **(PAN_BLOCK *)** block structure

Returns: Number of blocks returned
Number of total blocks if *lParam* is NULL
0 if no blocks exist
-1 if error

Compatibility: vector

Restrictions: All blocks will be returned if more than the total is requested.

Example:

```
void ShowBlockNames(HWND ctrlHandle)
/* Outputs a list of block names. */
{
        LONG        NumBlocks;
        PAN_BLOCK   Blocks[64];

        if (ctrlHandle == NULL) return;

        NumBlocks = SendMessage(ctrlHandle, PM_CTLGETBLOCKNAMES, 64, (LPARAM) Blocks);

        switch (NumBlocks) {
        case -1:
                Output("Incompatible control type\n");
                break;

        case 0:
                Output("No blocks defined\n");
                break;

        default:
                Output("%ld Blocks:\n", NumBlocks);
                for (LONG indx = 0; indx < NumBlocks; indx++) {
                        Output("Block %ld = %s\n", indx, Blocks[indx].name);
                }
        }

        return;
}
```

PM_CTLSETBLOCK

Purpose Sets the current "active block" of the page.

Description Vector decoders may offer the capability of decoding blocks of a file. This message allows the application to set the block number that will be returned to the decoder when it calls the **PANX_GetBlock** callback.

Parameters *wParam*: **(WORD)** block number, 0 for the whole page

lParam: not used

Returns: error code

Compatibility: vector

Special Notes: This message will force a repaint,
If an invalid block is specified, the last block will be set.

Example:

```
void ShowBlock(HWND ctrlHandle, WORD blockNo)
/* Display the specified block */
{
        if (ctrlHandle == NULL) return;

        int err = SendMessage(ctrlHandle, PM_CTLSETBLOCK, blockNo, 0);

        if (err != PAN_CTLERRNONE) {
                Output("Couldn't set active block\n");
                return;
        }

        SendMessage(ctrlHandle, PM_CTLREGEN, 0, 0);
}
```

Entity-Related Command Messages

PM_CTLGETENTITY

Purpose Used with CAD decoders to obtain identifiers to objects within a specified region of the viewed drawing.

Description When displaying a vector file that supports entities, this message provides a means of obtaining all handles within a specified region.

The region may be defined as a bounding box in world coordinates, or by specifying a single point in world coordinates (by setting the minimum equal to the maximum point of the bounding box) and a radius in terms of screen coordinates.

Parameters *wParam*: reserved, must be **0**

lParam: **(PAN_CtlGetEntityInfo *)** lpGetEntity

Returns: error code

Compatibility: Vector controls when viewing files that support entity handles. This may be determined by checking the CTL_FILE_HINT_EDAT bit within the hints flag obtained using PM_CTLGETFILE.

Example:

```
int ShowEntities(HWND ctrlHandle, PAN_CtlRange where)
/* Output the dimensions of an archive, database, or spreadsheet control */
/* in terms of rows and columns. */
{
        int err;
        struct PAN_CtlEntity myEntities;

        if (ctrlHandle == NULL)
                return (PAN_CTLERRMISC);

        myEntities.bbox = where;
        myEntities.iThreshold = 0;

        // Get control dimensions.
        err = (int) SendMessage(ctrlHandle, PM_CTLGETENTITY, 0,
                (LPARAM) (struct PAN_CtlEntity *) &dimensions);

        if (err == PAN_CTLERRNONE) {
                Output("Number of entities in region = %d\n", myEntities.nFound);
                GlobalFree(myEntities.hFound);
        }
        return (err);
}
```

PM_CTLSHOWENTITY

Purpose Used with CAD decoders to set the drawing color of a given entity e.g. to highlight a selected entity.

Description The message can be used to modify the drawing color/mode of an entity. It can also be used to restore the entity's original color.

Parameters

wParam: Negative of the number of elements pointed to by *lParam*.
0 or -1 indicate 1 element in the list.

lParam: **(PAN_CtlShowEntity *)** pShowEntity
NULL indicates that all current selections should be removed.

Returns: error code

Compatibility: vector-2D

Restrictions: This functionality is only available when viewing files that support entity handles. This may be determined by checking the CTL_FILE_HINT_EDAT bit within the hints flag obtained using PM_CTLGETFILE.

Extended Image Data-Related Command Messages

PM_CTLGETIMAGEEX

Purpose Retrieves the current contrast/anti-aliasing setting of an image.

Description This message only applies to monochrome raster images. All other image classes return **PAN_CTLERRNOTCOMPATIBLE**.

Parameters

wParam: **(WORD)** input, one of:
PAN_IMAGE_CONTRAST,
PAN_IMAGE_ANTIALIAS,
PAN_IMAGE_INVERT or
PAN_IMAGE_BRIGHTNESS.

lParam: **(wParam = PAN_IMAGE_CONTRAST)**
(int*) output receiving a contrast value between -100 to 100.
(wParam = PAN_IMAGE_ANTIALIAS)
(BOOL*) output receiving **TRUE** or **FALSE** indicating the current anti-aliasing mode.
(wParam = PAN_IMAGE_INVERT)
(BOOL*) output receiving **TRUE** or **FALSE** indicating the current invertion mode.
(wParam = PAN_IMAGE_BRIGHTNESS)
(int*) output receiving a brightness value between -100 to 100.

Returns error code

Compatibility: raster and vector-2D controls

Restrictions **PAN_IMAGE_INVERT** is only used for monochrome raster images and **PAN_IMAGE_BRIGHTNESS** is only used for colored raster images either in the raster control or overlaid onto a vector-2D format.

Example:

```
void ShowMonoOptions(HWND ctrlHandle)
/* Show the options being used to display a monochrome raster image. */
{
        int err=0;
        int contrast;
        BOOL antialias;

        if (ctrlHandle == NULL)
                return;

        err = (int)SendMessage(ctrlHandle, PM_CTLGETIMAGEEX, PAN_IMAGE_CONTRAST, (LPARAM)
                        ((int _far *) &contrast));

        if (err != PAN_CTLERRNONE) return;
        Output("Contrast for current image = %d\n", contrast);

        err = (int)SendMessage(ctrlHandle, PM_CTLGETIMAGEEX,
                        PAN_IMAGE_ANTIALIAS, (LPARAM) ((int _far *) &antialias);
```

```
if (err != PAN_CTLERRNONE) return;
Output("Antialiasing is %s\r", antialias ? "enabled" : "disabled");}
```

PM_CTLSETIMAGEEX

Purpose Adjusts the contrast/anti-aliasing setting on an image.

Description This message only applies to monochrome raster images. All other image classes return **PAN_CTLERRNOTCOMPATIBLE**.

Parameters

wParam: **(WORD)** input, one of:
PAN_IMAGE_CONTRAST,
PAN_IMAGE_ANTIALIAS,
PAN_IMAGE_INVERT or
PAN_IMAGE_BRIGHTNESS.

lParam: **(wParam = PAN_IMAGE_CONTRAST)**
(int) input, contrast setting -100 (fully saturated) to 100 (least saturated), default:0 (unchanged).
(wParam = PAN_IMAGE_ANTIALIAS)
(BOOL) input, anti-aliasing, **TRUE** (on) or **FALSE** (off), default: off.
(wParam = PAN_IMAGE_INVERT)
(BOOL) input, color inversion, **TRUE** (on) or **FALSE** (off), default: off.
(wParam = PAN_IMAGE_ BRIGHTNESS)
(int) input, brightness setting -100 (darkest (black)) to 100 (lightest (white)), default:0 (unchanged).

Returns error code

Compatibility raster and vector-2D controls

Restrictions **PAN_IMAGE_INVERT** is only used for monochrome raster images and **PAN_IMAGE_BRIGHTNESS** is only used for colored raster images either in the raster control or overlaid onto a vector-2D format.

Example

```
void ResetMonoOpts(HWND ctrlHandle)
/* Resets the monochrome rendering options to a contrast of 0, and */
/* disables anti-aliasing. */
{
        int err;

        if (ctrlHandle == NULL) return;

        err = SendMessage(ctrlHandle, PM_CTLSETIMAGEEX,
                PAN_IMAGE_CONTRAST, 0);
        if (err != PAN_CTLERRNONE) {
                Output("Couldn't reset contrast\n");
                return;
        }

        SendMessage(ctrlHandle, PM_CTLSETIMAGEEX,
                PAN_IMAGE_ANTIALIAS, FALSE);
}
```

Layers-Related Command Messages

PM_CTLGETLAYERSTATE

Purpose Returns the buffer of layer states set by the decoder.

Description As a vector decoder interprets a file, it may report different layers to the core. This message provides a means for the application to obtain a copy of the layer information that has been accumulated.

Parameters

wParam: (**LOWORD**) number of layer structures to retrieve
(**HIWORD**) reserved, must be set to 0

lParam: **(PAN_LAYER *)** layers

Returns: Number of layers copied to *lParam*,
current layer count if lParam is NULL,
0 if no layers,
-1 on failure.

Compatibility: vector

Restrictions: All layers will be returned if more than the total is requested.

Example:

```
void ShowLayerStates(HWND ctrlHandle)
/*      Display the states of up to the first 64 layers. */
{
        int numLayers;
        PAN_LAYER layers[64];

        if (ctrlHandle == NULL) return;

        numLayers = SendMessage(ctrlHandle, PM_CTLGETLAYERSTATE, 64,
                (LPARAM) layers);

        if (numLayers < 0) {
                Output("Error occurred when retrieving layer states\n");
                return;
        }

        if (numLayers == 0) {
                Output("No layers defined\n");
                return;
        }

        for (int indx = 0; indx < numLayers; indx++) {
                Output("Layer %d: ", indx);
                Output("%s %s\n", layers[indx].name,
                        (layers[indx].bState ? "On" : "Off"));
        }
}
```

PM_CTLSETLAYERSTATE

Purpose Set the parameters of the layer state buffer.

Description Sets the internal buffer of layers in the control. Any following **PANX_GetLayer** call references this buffer.

Parameters

wParam:	**(LOWORD)** number of layers
	(HIWORD) reserved, must be set to 0
lParam:	**(PAN_LAYER *)** layer information.
	If NULL, the default layers are restored.

Returns: error code

Compatibility: vector

Restrictions: If the number of layers to be set is larger than the number of existing layers, then only the number of existing layers is set.

Special Notes: This message will force a repaint.

Example:

```
void ShowEvenLayers(HWND ctrlHandle)
// Display half of the layers in a vector drawing. Assumes 256 or less layers in drawing.
{
        int numLayers;
        PAN_LAYER layers[256];

        if (ctrlHandle == NULL) return;

        numLayers = SendMessage(ctrlHandle, PM_CTLGETLAYERSTATE, 256, (LPARAM) layers);

        if (numLayers <= 0) {
                Output("No layers defined to manipulate\n");
                return;
        }

        for (int indx = 0; indx < numLayers; indx++) {
                layers[indx].bState = (indx & 1) ? TRUE : FALSE;
        }

        // Reset states of layers.
        SendMessage(ctrlHandle, PM_CTLSETLAYERSTATE, numLayers,
                (LPARAM) layers);

        // Force update of drawing.
        SendMessage(ctrlHandle, PM_CTLREGEN, 0, 0);
}
```

XREF-Related Command Messages

PM_CTLGETXREFSTATE

Purpose Returns the buffer of XRef states set by the decoder.

Description As a vector decoder interprets a file, it may report different XRefs to the core. This message provides a means for the application to obtain a copy of the Xref information that has been accumulated.

Parameters

wParam:	**< 0**	(MultiByte) Negative of number of XRef structures to retrieve.
	1	(Unicode) Flag indicating the usage of the Unicode interface.
lParam:	**(wParam: < 0)**	**(PAN_XREF *)** lpXRefs
	(wParam: 1)	**(std::[1]vector<CPanXRef> *)** pvXRefs

Returns: Number of XRefs or CPanXRefs copied to *lParam*, XRef count if lParam == NULL, 0 if no XRefs, -1 on failure.

Compatibility: vector-2D

Restrictions All XRefs will be returned if more than the total is requested.

Example:

```
void ShowMultiByteXRefStates(HWND ctrlHandle)
/*      Display the states of up to the first 64 XRefs. */
{
        int numXRefs, indx;
        PAN_XREF xrefs[64];

        if (ctrlHandle == NULL) return;

        numXRefs = SendMessage(ctrlHandle, PM_CTLGETXREFSTATE, -64,
                (LPARAM) ((PAN_XREF *) &xrefs));

        if (numXRefs < 0) {
                Output("Probably not a vector file\n");
                return;
        }

        if (numXRefs == 0) {
                Output("No XRefs defined\n");
                return;
        }

        for (indx = 0; indx < numXRefs; indx++) {
                Output("XRef %d: ", indx);
                Output("%s %s\n", xrefs[indx].name,
                        (xrefs[indx].bState == 1? "On" : "Off"));
        }
}
```

[1] Look at Overview section about usage of STL and Boost libraries

```
void ShowUnicodeXRefStates(HWND ctrlHandle)
{
        std::(1)vector<CPanXRef> vXRefrs;

        if (ctrlHandle == NULL) return;

        int numXRefs = SendMessage(ctrlHandle, PM_CTLGETXREFSTATE, 1, (LPARAM) (&vXRefs));

        if (numXRefs == 0) {
                Output("No XRefs defined\n");
                return;
        }

        for (int i = 0; i < numXRefs; i++) {
                Output("XRef %d: ", i);
                Output("%s %s\n", vXRefs[i].GetName(), (vXRefs[i].IsOn() ? "On" : "Off"));
        }
}
```

(1) Look at Overview section about usage of STL and Boost libraries

PM_CTLSETXREFSTATE

Purpose Set the parameters of the XRef state buffer.

Description Sets the internal buffer of XRefs in the control. Any following **PANX_GetXRefs** call references this buffer.

Parameters

wParam:	**(WORD) < 0**	(MultiByte) Negative of number of XRef structures.
	(WORD) 1	(Unicode) Indicating Unicode version.
lParam:	**(wParam: < 0)**	**(PAN_XREF *)** lpXRefs
	(wParam: 1)	**(std::[1]vector<CPanXRef> *)** pvXRefs

Returns: error code

Compatibility: vector-2D

Restrictions If *wParam* is larger than the number of existing XRefs, then only the number of existing XRefs is set.

Special Notes This message will force a repaint.

Example:

```
void ShowEvenMultiByteXRefs(HWND ctrlHandle)
// Display half of the XRefs in a vector drawing. Assumes 20 or less XRefs in drawing.
{
	int numXRefs;
	PAN_XREF xrefs[20];

	if (ctrlHandle == NULL) return;

	numXRefs = SendMessage(ctrlHandle, PM_CTLGETXREFSTATE, -20,
		(LPARAM) ((PAN_XREF *) xrefs));

	if (numXRefs <= 0) {
		Output("No XRefs defined to manipulate\n");
		return;
	}

	for (indx = 0; indx < numXRefs; indx++) {
		xrefs[indx].bState = (indx & 1) ? TRUE : FALSE;
	}

	// Reset states of XRefs.
	SendMessage(ctrlHandle, PM_CTLSETXREFSTATE, -numXRefs,
		(LPARAM) ((PAN_XREF *) xrefs));

	// Force update of drawing.
	SendMessage(ctrlHandle, PM_CTLREGEN, 0, 0);
}

void ShowEvenUnicodeXRefs(HWND ctrlHandle)
```

[1] Look at Overview section about usage of STL and Boost libraries

```
{
        if (ctrlHandle == NULL) return;

        std::(1)vector<CPanXRef> vXRefs;
        int numXRefs = SendMessage(ctrlHandle, PM_CTLGETXREFSTATE, 1, (LPARAM) )(&vXRefs));
        if (numXRefs <= 0) {
                Output("No XRefs defined to manipulate\n");
                return;
        }
        for (int i = 0; i < numXRefs; i++) {
                vXRefs[i].SetState( (i & 1)  ? true : false);
        }
        // Set the new XRef states
        SendMessage(ctrlHandle, PM_CTLSETXREFSTATE, 1, (LPARAM) (&vXRefs));
}
```

(1) Look at Overview section about usage of STL and Boost libraries

Bookmark-Related Command Messages

PM_CTLGETBOOKMARKS

Purpose Returns the buffer of PAN_BOOKMARK struct set by the decoder.

Description Any decoder can store a tree of bookmarks in the control. This tree is stored in a list of bookmarks. This message provides a means for the application to obtain a copy of the bookmark information that has been accumulated.

Parameters

wParam:
- **(WORD) < 0** Negative of the number of Bookmark structures to retrieve.
- **(WORD) 1** Flag indicating the usage of the Unicode interface.

lParam:
- **(wParam: < 0)** **(PAN_BOOKMARK *)** lpBMarks
- **(wParam: 1)** **(std::[1]vector<CPanBookmark> *)** pvBookmarks

Returns: Number of Bookmarks copied to *lParam*, Bookmark count if lParam == NULL, 0 if no Bookmarks, -1 on failure.

Compatibility: all

Example:

```
void OutputMultiByteBookMarks(HWND ctrlHandle)
/*      Output the number of leaf bookmarks stored in the control */
{
        // Get the total number of bookmarks stored in the control
        long nBookmarks = SendMessage(ctrlHandle, PM_CTLGETBOOKMARKS, 0, NULL);
        // Allocate the buffer that will receive the bookmarks
        PAN_BOOKMARK * pBookmarks = new PAN_BOOKMARK[nBookmarks];

        // Request the bookmarks stored in the control
        nBookmarks = SendMessage(ctrHandle , PM_CTLGETBOOKMARKS, - nBookmarks, pBookmarks);

        // Process the bookmark buffer
        if( nBookmarks < 0 ){
                Output("An error occurred in the control\r");
                return;
        }
        int nLeafBookmarks = 0;
        for( long iBookmark = 0; iBookmark < nBookmarks; iBookmark++ ){
                if( pBookmarks[iBookmark].fState == 0 ) nLeafBookmarks ++;
        }
        Output("%d leafs in the bookmark tree\r", nLeafBookmarks);

        // Clear the bookmark buffer
        delete [] pBookmarks;
}

void OutputUnicodeBookMarks(HWND ctrlHandle)
{
        // Request the bookmarks stored in the control
```

[1] Look at Overview section about usage of STL and Boost libraries

```
        std::(1)vector<CPanBookmark> vBookmarks;
        long nBookmarks = SendMessage(  ctrlHandle, PM_CTLGETBOOKMARKS, 1, &vBookmarks);

        int  nLeafBookmarks = 0;
        for( long iBookmark = 0; iBookmark < nBookmarks; iBookmark++ ){
                if(vBookmarks[iBookmark].fState == 0 ) {
                        nLeafBookmarks ++;
                }
        }

        Output("%d leafs in the bookmark tree\r", nLeafBookmarks);
}
```

(1) Look at Overview section about usage of STL and Boost libraries

Resource-Related Command Messages

PM_CTLGETRESOURCEINFOSTATE

Purpose Returns the resource information for the current page for vector control, for the current sheet for spreadsheet control and the whole control for document control.

Description As a decoder interprets a file, it may report different resources used to the core. This message provides a means for the application to obtain a copy of the resource information that has been accumulated. This information includes the resource type, identifier, resolved location, search paths and search hints.

Parameters

wParam: **>= 0** number of resource info structures to retrieve.
-1 the resource iterator is requested.

lParam: **(wParam >= 0) (PAN_RESOURCEINFO *)** lpResourceInfo
(wParam = -1)
(boost::[1]shared_ptr<IGetResourceInfo> *) resource iterator

Returns: Number of entries copied to *lParam*, resource count if lParam == NULL, 0 if no resources, -1 on failure.

Compatibility: document, spreadsheet and vector controls

```
void ShowResources(HWND ctrlHandle)
/*      Get the resource iterator    */
{
        std::[1]auto_ptr<csi::IGetResourceInfo> pResIter;

        if (ctrlHandle == NULL)
                return;

        SendMessage(PM_CTLGETRESOURCEINFOSTATE, -1, (LPARAM) & pResIter);

        if (!pResIter.get())
                return;

        csi::IGetResourceInfo & resourceIterator = *pResIter;
        while (resourceIterator) {
                const csi::ResourceInfo & resourceInfo = *resourceIterator;
                DisplayResource(resourceInfo);
                ++resourceIterator;
        }
}
```

[1] Look at Overview section about usage of STL and Boost libraries

Offset-Related Command Messages

PM_CTLGETOFFSET

Purpose Get the origin offset.

Description Returns the offset of the origin for the current file in world coordinates.

Parameters *wParam*: not used

lParam: **(PAN_CtlPos *)** origin offset

Returns: error code

Compatibility: all control types

Example:

```
void ShowOffset(HWND ctrlHandle)
/*      Display the offset of the file on the output device. */
{
        PAN_CtlPos fp;

        if (ctrlHandle == NULL)
                return;

        SendMessage(PM_CTLGETOFFSET, 0, (LPARAM) &fp);

        Output("Offset = (%f, %f, %f)\n", fp.x, fp.y, fp.z);

}
```

PM_CTLSETOFFSET

Purpose Set the origin offset.

Description Sets the offset of the origin for the current file in world coordinates.

Parameters *wParam*: not used

lParam: **(PAN_CtlPos *)** origin offset

Returns: error code

Compatibility: all control types

Example:

```
void SetDocumentRasterOffset(HWND ctrlHandle, double x, double y)
/* Set the current position at (x, y). */
{
        PAN_CtlPos p;

        if (ctrlHandle == NULL) return;

        p.x = x;
        p.y = y;
        p.z = 0.0;
        SendMessage(ctrlHandle, PM_CTLSETOFFSET, 0, (LPARAM) &p);
}
```

Font-Related Command Messages

PM_CTLGETBASEFONT

Purpose Returns the base font of the current file.

Parameters *wParam*: not used

lParam: **(LPLOGFONT)** base font

Returns: error code

Compatibility: document, database, spreadsheet, and archive controls

Example:

```
char* GetBaseFontName(HWND ctrlHandle)
/* Return a pointer to the font name */
{
        // keep the structure valid so we can use the returned pointer
        static LOGFONT      bf;
        int                 err;

        if (ctrlHandle == NULL) return NULL;

        err = (int) SendMessage(ctrlHandle, PM_CTLGETBASEFONT, 0, (LPARAM)&bf);

        if (err != PAN_CTLERRNONE) return NULL;
        return bf.lfFaceName;
}
```

PM_CTLSETBASEFONT

Purpose Sets the base font of the current file.

Parameters *wParam*: not used

lParam: **(const LOGFONT *)** base font

Returns: error code

Compatibility: document, database, spreadsheet, and archive controls

Example:

```
void UserChangeFont(HWND ctrlHandle)
/* Allow the user to change the font being used in a specified control. */
{
        CHOOSEFONT  cf;
        LOGFONT             lf;

        if (ctrlHandle == NULL) return;

        /* Set all structure fields to zero. */
        memset(&cf, 0, sizeof(CHOOSEFONT));

        cf.lStructSize = sizeof(CHOOSEFONT);
        cf.hwndOwner = ctrHandle;
        cf.lpLogFont = &lf;
        cf.Flags = CF_SCREENFONTS | CF_EFFECTS;
        cf.nFontType = SCREEN_FONTTYPE;

        ChooseFont(&cf);

        if (!SendMessage(ctrlHandle, PM_SETBASEFONT, 0,
                (LPARAM) ((LOGFONT *) lf))){
                Output("Couldn't change font\r");
        } else {
                Output("Base font changed\r");
        }
}
```

Database- and Spreadsheet-Related Messages

PM_CTLGETCOLWIDTH

Purpose Return the width of the specified field/column.

Description Return the width, in the specified units, of the given field/column in the current page. Specifying the zero'th column returns the width of the row header.

Parameters *wParam* **(WORD)** unit type to use.

lParam **(DWORD)** field/column

Returns: **(LRESULT)** width on success or -1 on failure

Compatibility: database and spreadsheet controls

Example:

```
WORD GetColWidthPixels(HWND ctrlHandle, DWORD who)
/* Return the width in pixels of the given field/column specified by who.
   Assume that ctrlHandle refers to a database or spreadsheet file
*/
{
        if (ctrlHandle == NULL) return 0;
        return (WORD) SendMessage(ctrlHandle, PM_CTLGETCOLWIDTH, CTLUNIT_PIXELS, (LPARAM)
                who);
}
```

PM_CTLGETROWHEIGHT

Purpose Returns the height of the given record/row in specified units.

Description Return the height, in the specified units, of the given record/row in the current page. Specifying the zero'th row returns the height of the column headers.

Parameters *wParam*: **(WORD)** unit type to use

lParam: **(DWORD)** record/row

Returns: **(LONG)** height on success or -1 on failure

Compatibility: database and spreadsheet controls

Example:

```
LONG GetRowHeightTwips(HWND ctrlHandle, DWORD who)
/* Return the height in TWIPS of the given field/column specified by who.
  Assume that ctrlHandle refers to a database or spreadsheet file.*/
{
        if (ctrlHandle == NULL) return 0;
        return (LONG) SendMessage(ctrlHandle,
                PM_CTLGETROWHEIGHT, CTLUNITS_TWIPS, (LPARAM) who);
}
```

PM_CTLSORT

No longer supported.

Page-Related Command Messages

PM_CTLGETNUMPAGES

Purpose Returns the number of pages in the current file.

Description Return the number of pages in the current file in the **int** pointed to by *lParam*.

Parameters *wParam*: not used

lParam: **(LPINT)** number of pages, signed integer giving the number of pages in the file. If the number is positive, this is the total number of pages in the file. If the number is negative, the file may be multi-page, but it contains at least this many pages (in absolute value).

This is necessary, since certain file formats (e.g. CGM) do not report the number of pages in the file (for reasons of efficiency). Once the control has determined the number of pages in the file, the **PM_CTLGETNUMPAGES** message will return a positive number.

Returns: error code

Compatibility: all control types

Example:

```
void ShowNumPages(HWND ctrlHandle)
/*      Display the current number of pages on the output device. */
{
        DWORD pages;

        if (ctrlHandle == NULL) return;

        SendMessage(ctrlHandle, PM_CTLGETNUMPAGES, 0, (LPARAM) &pages);
        if (pages > 0) {
                Output("The file contains %d pages\n", pages);
        } else {
                Output("The file contains at least %d pages\n", -pages);
        }
}
```

PM_CTLGETPAGE

Purpose Returns the current page number within the current file

Description Return the current page number in the current file in the **WORD** pointed to by *lParam*.

Parameters *wParam* not used

lParam: **(LPWORD)** current page number

Returns: error code

Compatibility: all control types

Example:

```
void DisplayCurrentPage(HWND ctrlHandle)
/*      Display the number of the current page within the file being viewed on the
        output device. */
{
        int err;
        WORD curPage;

        if (ctrlHandle == NULL) return;

        err = SendMessage(ctrlHandle, PM_CTLGETPAGE, 0,
                (LPARAM) ((LPWORD) &curPage));

        if (err != PAN_CTLERRNONE) {
                Output("Couldn't get current page\n");
        } else {
                Output("Page %d is the current page\n", curPage);
        }
}
```

PM_CTLGETPAGESIZE

Purpose Returns the size of the specified page in the current file.

Description Return the maximum extents for the current page of the current file. The **PAN_CtlRange** structure returns the extents as a pair of points that bound the page.

For vector, raster, archive, database and spreadsheet controls, the returned value always represents the maximum extents of the current page.

Parameters

wParam: **(WORD)** page number for document control.
Not used by other controls.

lParam: **(PAN_CtlRange *)** page size

Returns: error code

Compatibility: all control types

Example:

```
void ShowPageSize(HWND ctrlHandle, WORD pagenum)
/* Display the specified page size in view coordinates on the output device. */
{
        PAN_CtlRange rg;

        if (ctrlHandle == NULL) return;

        SendMessage(ctrlHandle, PM_CTLGETPAGESIZE, pagenum, (LPARAM)&rg);
        Output("Page Size = (%f, %f, %f)\n",
                abs(rg.max.x - rg.min.x),
                abs(rg.max.y - rg.min.y),
                abs(rg.max.z - rg.min.z));
}
```

PM_CTLSETPAGE

Purpose Sets the current page number within the current file.

Description Set the current page number for the current file based on the value specified by the **WORD** passed as the *wParam*.

Parameters *wParam*: **(WORD)** current page number

lParam: not used

Returns: error code

Compatibility: all control types

Example:

```
void SetFirstPage(HWND ctrlHandle)
/* Display the first page of the file. */
{
        int err;

        if (ctrlHandle == NULL) return;

        err = SendMessage(ctrlHandle, PM_CTLSETPAGE, 1, 0);

        if (err != PAN_CTLERRNONE) {
                Output("Couldn't change to page 1.\n");
        }
}
```

Printing Command Messages

PM_CTLPRINT

Purpose Prints the specified extents.

Description Print the specified extents using the options specified in the **PAN_CtlPrintOptions** structure pointed to by *lParam*.

Parameters *wParam*: Page number specifying the page to print, **(WORD)-1** to print all pages

lParam: **(const PAN_CtlPrintOptions *)** printing options

Returns: error code

Compatibility: all control types except for Vector-3D

Example:

```
int Print(HWND ctrlHandle, PRINTDLG * prnDlg)
/*      Print the control using default values */
{
        int             err=0;
        PAN_CtlPrintOptions printOptions;

        if (ctrlHandle == NULL) return 0;

        memset(&printOptions, 0, sizeof(PAN_CtlPrintOptions));
        printOptions.printDlg = prnDlg;

        // fill structure with default values
        printOptions.units = CTLUNIT_INCH;

        printOptions.margins.units = CTLUNIT_INCH;
        printOptions.margins.top   = 0.75;
        printOptions.margins.left  = 0.75;
        printOptions.margins.bottom= 0.75;
        printOptions.margins.right = 0.75;

        // all the header default to null strings */

        err= (int) SendMessage(ctrlHandle, PM_CTLPRINT, (WORD) -1,
                                                        (LPARAM) &printOptions);
        return err;
}
```

PM_CTLPRINTPREVIEW

Purpose Returns the print preview of the specified extents.

Description Return the printing preview of the specified extents using the options specified in the **PAN_CtlPrintOptions** structure pointed to by *lParam*. All member variables and structures (except **printPreview**) of the given **PAN_CtlPrintOptions** structure must be properly initialized.

Parameters *wParam*: not used

lParam: **(PAN_CtlPrintOptions *)** preview information

Returns: error code

Compatibility: all control types

Example:

```
int PrintPreview(HWND ctrlHandle, PRINTDLG * prnDlg)
/* Generates the print preview rectangles for the control */
/* using default values */
{
        int         err;
        PAN_CtlPrintOptions printOptions;

        if (ctrlHandle == NULL) return 0;

        memset(&printOptions, 0, sizeof(PAN_CtlPrintOptions));
        printOptions.printDlg = prnDlg;

        // fill structure with default values
        printOptions.units = CTLUNIT_INCH;
        printOptions.margins.units = CTLUNIT_INCH;
        printOptions.margins.top   = 0.75;
        printOptions.margins.left  = 0.75;
        printOptions.margins.bottom= 0.75;
        printOptions.margins.right = 0.75;

        // all the header default to null strings */
        err = (int) SendMessage(ctrlHeader, PM_CTLPRINTPREVIEW, 0,
                        (LPARAM) &printOptions);

        if (err == PAN_CTLERRNONE) {
                Output("%d pages needed arranged (%d horz, %d vert)\n",
                        printOptions.printPreview.nPages,
                        printOptions.printPreview.nHorzPages,
                        printOptions.printPreview.nVertPages);

                // ... other information may be displayed.
        }

        return err;
}
```

PM_CTLVALIDATEMARGINS

Purpose Validates the printer margins.

Description This message is used to allow the control to ensure that the printer margins in the given **PAN_CtlPrintOptions** structure are at least the minimum size supported by the printer. The margins specified in the options are overwritten if they do not meet the minimum requirements of the printer.

Parameters *wParam*: not used

lParam: **(PAN_CtlPrintOptions *)** print options

Returns: error code

Compatibility: all control types

Restrictions: This message requires that the **hDC** member of the **printDlg** structure contained in the **PAN_CtlPrintOptions** structure be a valid printer device context handle.

The margins specified in the print options will not be adjusted if the **mode** member of the PAN_CtlPrintOptions structure has the **PAN_CTLMODEIGNOREMINMARGINS** flag set.

Example:

```
void SetupPrintMargins(HWND ctrlHandle
        PAN_CtlPrintOptions *printOptions, UINT units,
        double top, double bottom, double left, double right)
/* Set up the margins for a print operation */
{
        printOptions->margins.units = units;
        printOptions->margins.top = top;
        printOptions->margins.left = left;
        printOptions->margins.bottom = bottom;
        printOptions->margins.right = right;

        // Make sure print margins are at least the minimum required
        // by printer.
        SendMessage(ctrlHandle, PM_CTLVALIDATEMARGINS, 0,
                (LPARAM) printOptions);
}
```

Text-Related Command Messages

PM_CTLGETSTRING

Purpose Returns the number of retrievable strings from a control, or a specified string itself.

Description PM_CTLGETSTRING allows an application to retrieve strings that were decoded from the displayed file.

Parameters

wParam: -1: To return the total number of strings.
-2: Use the Unicode interface.
0..numstrings-1: To return the specified string.

lParam: if wParam == -1 then (LPDWORD) nStrings.
if wParam == -2 then (boost::[1]shared_ptr<IGetString> *) pspGetString
if wParam > 0 then (LPSTR) lpBuffer.

Returns: error code

Compatibility: vector files with entity support.

Example:

```
/* Display the status of the control on the output device. */
void ShowMultiByteStrings(HWND ctrlHandle)
{
        DWORD numstrings;
        DWORD ii;
        char    szBuffer[PAN_MAX_STR];

        if (ctrlHandle == NULL) return;

        err = SendMessage(ctrlHandle, PM_CTLGETSTRING, -1,
                        (LPARAM) &numstrings);

        if (PAN_CTLERRNONE == err) {
                for (ii = 0; ii < numstrings; ii++) {
                        SendMessage(ctrlHandle, PM_CTLGETSTRING, (WPARAM) ii,
                                (LPARAM) &szBuffer);
                        Output("String %d: %s\n", ii, szBuffer);
                }
        }
}

void ShowUnicodeStrings(HWND ctrlHandle)
{
        int i=0;
        boost::shared_ptr<IGetString> spGetString;
        SendMessage (ctrlHandle, PM_CTLGETSTRING, 0, &spGetString);
        for (IGetString & iter = *spGetString ; iter ; ++iter) {
                Output("String %d: %s\n", ++i, *iter);
        }
}
```

[1] Look at Overview section about usage of STL and Boost libraries

PM_CTLSEARCH

Purpose Search the current file for a text string.

Description Search for the string given in the **PAN_CtlSearchInfo** structure pointed to by *lParam* in the current file. The string in the current file is returned in the structure if the search is successful.

A major change from version 1.1 of the controls is the handling of a NULL search string. In version 1.1, this would result in a search dialog being presented to the user to allow the search string to be specified. This facility has been removed in version 1.2.

Searches can now be performed on vector files which support entity operations (indicated by the CTL_FILE_HINT_EDAT bit in the dwHints field of the PAN_CtlFileInfo structure from PM_CTLGETFILE). The results of these searches include the bounding vertices of matching strings (if the found occurrence is a sub-string, the bounding box of the full string is returned), as well as the entity handle. If all occurences of a string are requested, the second and following matches will be stored in the memory regions specified by the hFoundBBox and hFoundHandle of the PAN_CtlSearchInfo structure.

Parameters

wParam:
- **0** MultiByte text search
- **≠ 0** Unicode text search

lParam:
- **(wParam: 0) (PAN_CtlSearchInfo *)** MultiByte search information.
- **(wParam: ≠ 0) (CPanTextSearch *)** Unicode search interface that will be initialized with a TextSearch interface.

Returns: error code

Compatibility: vector, document, database, spreadsheet, and archive controls

Example:

```
/* Search the file for the specified string, starting at the beginning of the first page of the document. NOTE:
ctrlHandle is of type Document, Database, Spreadsheet or Archive */
void SearchMultiByteString(HWND ctrlHandle, LPCSTR string)
{
        PAN_CtlSearchInfo si;

        if (ctrlHandle == NULL) return;

        si.startPos.page = 1;
        si.startPos.flow = FLOW_MAIN;
        si.startPos.offset = 0;
        si.fDown = TRUE;                // search downward
        si.fWrap = TRUE;                // wrap around file
        si.fCase = FALSE;               // case sensitive
        si.fWord = FALSE;               // whole word
        si.string = string;
        // Find only the first occurence.
        SendMessage(ctrlHandle, PM_CTLSEARCH, FALSE, (LPARAM) &si);
}
```

```
void SearchUnicodeString(HWND ctrlHandle, std::(1)wstring searchText)
{
	PAN_CtlCaretPos pos = {1, 0, 0};
	CPanTextSearchOptions searchOptions(true, true, false, false);
	CPanTextSearch search(searchString, searchOptions, pos);

	SendMessage (ctrlHandle, PM_CTLSEARCH, 1, (LPARAM)&search);

	std::vector<CPanTextSearch:: ContourPoints> vContours;
	while (search.FindNext(pos, &vCountours)) {
		…
	}
}
```

(1) Look at Overview section about usage of STL and Boost libraries

Mouse Command Messages

PM_CTLGETLMBACTION

Purpose Returns the left/right mouse button control window behavior.

Description Return, in the **WORD** pointed to by *lParam*, the action taken when clicking and dragging with the left mouse button in the control window. The action can be one of the following constants.

PAN_CTLLMBNONE:	does nothing
PAN_CTLLMBSELECT:	selects a portion of the current file
PAN_CTLLMBZOOM:	zooms in on a portion of the current file

Parameters

wParam 0 => LMB action
1 = > RMB action

lParam: **(LPARAM)** LMB action

Returns: error code

Compatibility: all control types

Example:

```
void ShowLMBAction(HWND ctrlHandle)
/*      Display the current left mouse button action on the output device. */
{
        WORD lmb;

        if (ctrlHandle == NULL) return;
        SendMessage(ctrlHandle, PM_CTLGETLMBACTION, 0, (LPARAM) &lmb);
        switch (lmb) {
            case PAN_CTLLMBNONE: Output("Left Mouse Button action= NONE"); break;
            case PAN_CTLLMBSELECT: Output("Left Mouse Button action= SELECT"); break;
            case PAN_CTLLMBZOOM: Output("Left Mouse Button action= ZOOM"); break;
        }
}
```

PM_CTLSETLMBACTION

Purpose Sets the left/right mouse button control window behavior.

Description Set the action taken when clicking and dragging with the left mouse button in the control window. The action can be one of the following constants:

PAN_CTLLMBNONE	does nothing
PAN_CTLLMBSELECT	selects a portion of the current file
PAN_CTLLMBZOOM	zooms in on a portion of the current file

Parameters *wParam*: **(WORD)** MB action

lParam: 0 => LMB action.
1 => RMB action.

Returns: error code

Compatibility: all control types

Examples:

```
void DisableLeftMouseButton(HWND ctrlHandle)
{
	if (ctrlHandle == NULL) return;

	SendMessage(ctrlHandle, PM_CTLSETLMBACTION, (WPARAM)PAN_CTLLMBNONE, 0L);
}

void ChangeLeftMouseButtonState(HWND ctrlHandle, WORD action)
{
	if (ctrlHandle == NULL) return;

	SendMessage(ctrlHandle, PM_CTLSETLMBACTION, action, 0L);
}
```

Device Context-Related Message

PM_CTLRENDERONTODC

Purpose Renders the specified extents onto the given device context.

Description This message may be used to implement printing or for combining the contents of a control with graphical elements from other sources.

Parameters *wParam*: Not used

lParam: **(PAN_CtlRenderOptions *)** options

Returns: error code

Compatibility: all control types

Example:

```
void RenderOnToDc(HWND ctrlHandle, HDC dc, RECT mapRect)
/*      Render the current view in the control to the supplied device context
                                (dc), mapping it so that it will fit the mapRect rectangle.
*/
{
        PAN_CtlRange curView;
        PAN_CtlRenderOptions ro;

        if (ctrlHandle == NULL) return;

        SendMessage(ctrlHandle, PM_CTLGETVIEWEXTENTS, 0, (LPARAM) &curView);

        memset(&ro, 0, sizeof(PAN_CtlRenderOptions));
        ro.source = curView;
        ro.mode = PAN_CTLMODEOPAQUE;
        ro.hdc = dc;        // device context handle

        // Destination range.
        ro.devRect.left = mapRect.left;
        ro.devRect.top = mapRect.top;
        ro.devRect.right = mapRect.right
        ro.devRect.bottom = mapRect.bottom

        // Source range.
        ro.source = curView;

        SendMessage(ctrlHandle, PM_CTLRENDERONTODC, 0, (LPARAM) &ro);
}
```

Formatting Device Related Messages

PM_CTLSETDEVICE

Purpose Sets the formatting device for document files.

Description Sets the device used by the document control for text formatting. Fonts not supported by the device are adapted. wParam specifies the type of device: screen, printer with description given in lParam, or the standard device specified in WIN.INI.

Parameters

wParam: Device type which can be one of the following: PAN_CTLDEVICESCREEN, PAN_CTLDEVICEPRINTER, PAN_CTLDEVICESTANDARD.

lParam: if wParam == PAN_CTLDEVICEPRINTER, lParam must point to a string containing the device description specified in the same format used in WIN.INI: “Name,Driver,Port”. Otherwise, lParam must be set to NULL.

Returns: error code

Compatibility: Document control.

PM_CTLGETDEVICE

Purpose Gets the formatting device for document files.

Description Returns the device used by the document control for text formatting.

Parameters

wParam: Size of buffer passed in lParam to hold device description if current device type is PAN_CTLDEVICEPRINTER.

lParam: Buffer to hold device description if current device type is PAN_CTLDEVICEPRINTER. If NULL description is not returned

Returns: Type of current device: PAN_CTLDEVICESCREEN, PAN_CTLDEVICEPRINTER, or PAN_CTLDEVICESTANDARD.

Compatibility: Document control.

Notification Messages Summary

This section presents a summary of the notification messages sent by the controls, as well as a brief description, parameters and return values.

(Messages names prefixed with "*" are new to version 1.2 of the controls.)

Controls-Related Notification Messages

Message Name	Purpose
PNM_CTLDESTROY	A control sends this message prior to its destruction.
PNM_CTLDROPFILE	Sent when a file has been dropped on the control window.
PNM_CTLHELPSTRING	This message returns a help string describing the current state of the control, e.g., reading file... The calling application can use such help strings to provide feedback to the user, for example.
PNM_CTLSIZE	Sent when the control window has changed size.
PNM_CTLSTATUS	Sent when a controls' status changes.

Image-Manipulation Related

PNM_CTLPAINT	Sent when the control window has been repainted.
PNM_CTLREGEN	Sent when the control has been regenerated.
PNM_CTLSETFOCUS	Sent when a control obtains the focus.
PNM_CTLHSCROLL	Sent when the control window has been horizontally scrolled
PNM_CTLVSCROLL	Sent when the control window has been vertically scrolled.

Printing Notification Messages

PNM_CTLPRINT	A control sends this message when it has completed a print job initiated with PM_CTLPRINT.
PNM_CTLPRINTINGPAGE	A control sends this message when printing a given page during a print job.
PNM_CTLPRINTPROCESSINGPAGE	A control sends this message when processing (reading) a given page during a print job.

Clipboard-Related Message

PNM_CTLSETSEL	A control sends this message whenever the selection has changed.
PNM_CTLRENDERSEL	Sent while selection is being copied to the clipboad to allow the application to append its own data.

Archive/Database/Spreadsheet specific

PNM_CTLARCFILE	An archive control sends this message when an entry has been double-clicked in the control window.
***PNM_CTLCOLWIDTH**	When the width of a column in a archive / database / spreadsheet control has changed, this message is sent to inform the parent of the range of changed columns.
***PNM_CTLROWHEIGHT**	When the height of a row in a archive / database / spreadsheet control, this message is sent to inform the parent of the range of rows changed.

OLE-Related Notification Message

PNM_CTLOBJECT	Sent when an OLE object marker has been double clicked.

Link Notification Message

PNM_CTLLINK	Sent when the mouse is moved on top of a hot link or when a hot link is double clicked

Views Notification Message

PNM_CTLSETVIEWEXTENTS	Sent when the view extents have changed.

Control Notification Messages

PNM_CTLDESTROY

Purpose A control sends this message prior to its destruction.

Parameters *wParam*: not used

lParam: not used

PNM_CTLDROPFILE

Purpose A control send this message whenever a file has been dropped on the control window.

Parameters

wParam: **(WORD)** file index in sequence of files dropped

lParam: **(LPCSTR)** filename

PNM_CTLHELPSTRING

Purpose

This message returns a help string describing the current state of the control, e.g., reading file... The calling application can use such help strings to provide feedback to the user, for example.

Parameters

wParam: not used

lParam: **(LPCSTR)** help string

PNM_CTLSIZE

Purpose A control sends this message when the control window has changed size.

Parameters *wParam*: not used

lParam: **(const RECT *)** new size

PNM_CTLSTATUS

Purpose A control sends this message whenever its status changes.

Parameters *wParam*: not used

lParam: **(DWORD)** status (see **PM_CTLGETSTATUS**)

Image Notification Messages

PNM_CTLPAINT

Purpose A control sends this message whenever the control window has been repainted.

Parameters *wParam*: not used

lParam: **(const RECT *)** area repainted in client coordinates

PNM_CTLREGEN

Purpose A control sends this message whenever the control has been regenerated.

Parameters *wParam*: not used

lParam: not used

PNM_CTLSETFOCUS

Purpose A control sends this message whenever it obtains the focus.

Parameters *wParam*: **(HWND)** handle of window losing focus

lParam: not used

PNM_CTLHSCROLL, PNM_CTLVSCROLL

Purpose A control sends these messages whenever the control window has been scrolled.

Parameters *wParam*: **(WORD)** scroll amount in pixels

lParam: not used

Printing Notification Messages

PNM_CTLPRINT

Purpose A control sends this message when it has completed a print job initiated with **PM_CTLPRINT**.

Parameters *wParam*: not used

lParam: not used

PNM_CTLPRINTINGPAGE

Purpose

A control sends this message when printing a given page during a print job. The contents of the array are the same as for **PNM_CTLPRINTPROCESSINGPAGE**.

Parameters

wParam: **(WORD)** number of page number entries in array (always 4)

lParam: **(LPWORD)** array of page numbers

The array contains the following information:

offset 0 current physical page number
offset 1 number of physical pages
offset 2 current logical page number
offset 3 number of logical pages

PNM_CTLPRINTPROCESSINGPAGE

Purpose

A control sends this message when processing (reading) a given page during a print job.

Parameters

wParam: **(WORD)** number of page number entries in array (always 4)

lParam: **(LPWORD)** array of page numbers

The array contains the following information.

offset 0 current physical page number
offset 1 number of physical pages
offset 2 current logical page number
offset 3 number of logical pages

Clipboard-Related Notification Message

PNM_CTLSETSEL

Purpose A control sends this message whenever the selection has changed.

Parameters *wParam*: not used

lParam: **(const PAN_CtlSel *)** new selection

PNM_CTLRENDERSEL

Purpose A control sends this message while the selection is being copied to the clipborad to allow the application to append its own data.

Parameters *wParam*: not used

lParam: **(PAN_CtlRenderOptions *)** options

Compatibility Raster and Vector controls.

Archive-, Database- and Spreadsheet-Specific Notification Messages

PNM_CTLARCFILE

Purpose

An archive control sends this message when an entry has been double-clicked in the control window.

Parameters

wParam:	not used	
lParam: of extracted file are placed	**(LPCSTR)** name of temporary file in which followed by name of extracted file at offset **PAN_MAX_PATH**	contents

PNM_CTLCOLWIDTH

Purpose Informs the controls parent that the width of a range of columns has been changed.

Parameters

wParam: **(int)** starting changed column number

lParam: **(int)** ending changed column number

PNM_CTLROWHEIGHT

Purpose Informs the control's parent that the row height has been changed.

Parameters *wParam*: **(int)** first changed row number

lParam: **(int)** last changed row number

OLE-Related Notification Message

PNM_CTLOBJECT

Purpose A control sends this message when an OLE Object marker is double clicked.

Parameters *wParam:* not used

lParam: (PAN_CtlObject *) object desciptor.

Link-Related Notification Message

PNM_CTLLINK

Purpose A control sends this message when the mouse cursor is moved on top of a hot-link or when a hot link is double clicked.

Parameters *wParam:* PAN_CTLLINKSETCURSOR (mouse moved) or PAN_CTLDBLCLICKED (left mouse button was double clicked).

lParam: (PAN_Link *) link desciptor.

Views-Related Notification Message

PNM_CTLSETVIEWEXTENTS

Purpose A control sends this message when the view extents have changed.

Parameters

wParam	not used
lParam	**(const PAN_CtlRange *)** new view extents

Appendix A: Configuration Options

The default behaviour of the C.S.I. VCET Controls can be affected by setting options in an initialization file, i.e., INI file. This INI file is normally named "PCTL.INI" and is located in the Windows directory. The name and location of this file can be changed by specifying it to the function PAN_LoadControls(). Refer to the VCET document for further information.

All the following options may be specified in the “Options” section of the initialization file.

Vector Viewing Options

SHOWTEXT=<0|1>

Show text entities.

Default*:*1

SHOWLINESTYLE=<0|1>

Show linestyle patterns.

Default*:*0

SHOWDIMENSION=<0|1>

Show dimension entities.

Default*:*1

SHOWFILL=<0|1>

If *ON*, display filled entities (solids, fat polylines, etc).
If *OFF*, show the outline of these entities.

Default*:*0

SHOWXREFS=<0|1>

Show external reference files.

Default:1

VectorCustomMeta=(0|1)

If VectorCusomMeta is set to 1, a high resolution metafile is used for manipulating images. If 0, a Windows metafile is used for backing storage.

Default:1

Compatibility: Vector formats

Raster Viewing Options

CONTRAST=*contrast_level*

contrast_level can be between -100 to 100 to specify low to high contrasts.

Default:0

Compatibility: Monochrome Raster formats

ANTIALIAS=<0|1>

If *antialias* is 1 then the image is anti-aliased using a scale-to-gray algorithm.

Default:0

Compatibility: Monochrome Raster formats

Path Settings

XREFPATHS=*path*s

paths specifies a semicolon-delimited list of directories to search for external references, for AutoCAD and Intergraph/Microstation drawings:
e.g., XREFPATHS=D:\acad13\blocks;E:\ustation\cells

Default:none

Compatibility: Vector formats

XFONTPATHS=*path*s

paths specifies a semicolon delimited list of directories to search for AutoCAD shx fonts,
e.g., XFONTPATHS=C:\acad13\fonts;C:\acad12\support

Default:none

Compatibility: Vector formats

Document Viewing Options

USESTORAGE=*<0|1>*

Use page storage management or not. Enables viewing of long documents.

Default*:1*

INMEMPAGETOL=*n_pages*

n_pages specifies the number of pages on each side of the visible range that page storage management should attempt to keep in memory.

Default*:* 4

Spreadsheet/Database/Archive Options

ONEBPECALL=*<0|1>*

Determines strategy used to read files. Should be left at default value for initial release of VCET 1.2.

Default*:0*

Miscellaneous Options

RasterMemLimit=*n_kbytes*

n_kbytes is a numeric value indicating a memory threshold. If Windows' Global memory heap falls below this amount AutoVue will begin swapping raster data to disk.

Default*:6000*

Compatibility*:* Raster formats

VectorMemLimit=*n_kbytes*

n_kbytes is a numeric value indicating a memory threshold. If Windows' Global memory heap falls below this amount AutoVue will begin swapping vector data to disk.

Default*:*4096

Compatibility*:* Vector formats

Microstation/DGN Viewing Options

DGNCOLORTBL=*color_table_filename*

color_table_filename specifies the default color table file to be used by the Microstation/DGN decoder.

Default*:*color.tbl, located in the same directory as PFVC_DGN.DLL

Compatibility*:* PFVC_DGN.DLL

DGNLSTYLERSC=*linestyle_resource_filename*

color_table_filename specifies the linestyle resource file to be used by the Microstation/DGN decoder. This file is used to render extended linestyle patterns for lines, multi-lines *etc.*

Default:lstyle.rsc, located in the same directory as PFVC_DGN.DLL

Compatibility*:* PFVC_DGN.DLL

Postscript Viewing Options

PSMinDPI=*nDPI*

nDPI is a numeric value indicating the *minimum* resolution (in dots-per-inch) to use in rendering Postscript files. The VCET controls automatically calculate the rendering resolution, based on the resolution of the output device. If the user finds this to be too low (for example, details seem jagged on the output), this resolution can be increased. Note that if the rendering resolution calculated by the VCET controls is greater than that specified by this option, then the greater resolution (i.e., VCET's) will be used.

Default*:0*

Compatibility*:* PFVC_PS.DLL

Index

Index

—R—

—S—

—T—

—V—

—W—

—Z—

Feedback

We would appreciate your feedback, comments or suggestions. Contact us by e-mail or telephone. Let us know what you think.

General Inquiries:

Telephone: +1.514.905.8400 or +1.800.363.5805
E-mail: autovuesales_ww@oracle.com
Web Site: http://www.oracle.com/autovue/index.html

Sales Inquiries:

Telephone: +1.514.905. 8400 or +1.800.363.5805
E-mail: autovuesales_ww@oracle.com

Customer Support:

Web Site: http://www.oracle.com/support/index.html