

# VCET API Manual

Viewing and Conversion Enabling Technology

**Copyright © 1999, 2012, Oracle and/or its affiliates. All rights reserved.**

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software-Restricted Rights (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

# **Contents**

<b>INTRODUCTION.....</b>	<b>7</b>
<b>Introduction .....</b>	<b>8</b>
About this Document.....	9
<b>OVERVIEW.....</b>	<b>14</b>
Overview.....	15
<b>FUNCTIONS .....</b>	<b>18</b>
Function Summary .....	19
Function Descriptions.....	20
PAN_LoadControls() .....	21
PAN_CreateControl().....	22
PAN_FreeControls().....	24
PAN_GetCtlErrorCode() .....	25
Error codes .....	25
<b>MESSAGES.....</b>	<b>30</b>
<b>Structures and Unions used by Messages.....</b>	<b>31</b>
Structure Summary .....	31
Structure Descriptions .....	33
Command Message Summary .....	57
Purpose .....	57
Command Message Descriptions .....	61
Controls Command Messages .....	61
PM_CTLCLONECONTROL.....	61
PM_CTLDESTROY .....	62
PM_CTLGETCAPS .....	63
PM_CTLGETDIMS .....	65
PM_CTLGETFILE.....	66
PM_CTLGETFILEFMTS .....	67
PM_CTLGETFILETYPE.....	68
PM_CTLGETINFO.....	69
PM_CTLGETMODE .....	70
}PM_CTLGETOPTION.....	71
PM_CTLGETOPTION .....	72
PM_CTLGETSTATUS .....	73
PM_CTLREGEN.....	74
PM_CTLSETCAPS.....	75
PM_CTLSETFILE .....	77
PM_CTLSETFILEEX .....	78
PM_CTLSETMODE.....	82
PM_CTLSETOPTION .....	88
Image-Related Command Messages.....	90
PM_CTLCONVERT .....	90
PM_CTLFLIP.....	92
PM_CTLGETFLIP .....	93

PM_CTLGETROTATION.....	94
PM_CTLGETZOOM .....	95
PM_CTLPAINT, PM_CTLSIZE, PM_CTLHSCROLL, PM_CTLVSCROLL.....	96
PM_CTLROTATE .....	97
PM_CTLSETZOOM .....	98
Coordinates-Related Command Messages .....	99
PM_CTLCARETTOWORLD .....	99
PM_CTLCLEARSELS .....	100
PM_CTLCLIENTTOWORLD.....	101
PM_CTLGETCARETPOS .....	102
PM_CTLSETCARETPOS .....	103
PM_CTLWORLDTOCARET .....	104
PM_CTLWORLDTOCLIENT.....	105
PM_CTLXFRMRECT .....	106
PM_CTLCOPY .....	107
PM_CTLGETCLPBRDFMTS .....	108
PM_CTLGETNUMSELS .....	109
PM_CTLGETSELS.....	110
PM_CTLSETSEL.....	111
PM_CTLSETSELCARET.....	112
Color-Related Command Messages.....	113
PM_CTLGETFGBGCOLOR .....	113
PM_CTLGETPALETTE.....	114
PM_CTLPALETTECHANGED .....	115
PM_CTLQUERYNEWPALETTE.....	116
PM_CTLSETFGBGCOLOR.....	117
PM_CTLSETPALETTE .....	118
Views-Related Command Messages .....	119
PM_CTLGETVIEW.....	119
PM_CTLGETVIEWEXTENTS.....	120
PM_CTLGETVIEWNAMES.....	121
PM_CTLSETVIEW .....	123
PM_CTLSETVIEWEXTENTS.....	125
Blocks-Related Command Messages.....	126
PM_CTLGETBLOCK.....	126
PM_CTLGETBLOCKNAMES.....	127
PM_CTLSETBLOCK .....	129
Entity-Related Command Messages .....	130
PM_CTLGETENTITY.....	130
PM_CTLSHOWENTITY.....	131
Extended Image Data-Related Command Messages .....	132
PM_CTLGETIMAGEEX.....	132
PM_CTLSETIMAGEEX .....	134
Layers-Related Command Messages.....	135
PM_CTLGETLAYERSTATE .....	135
PM_CTLSETLAYERSTATE .....	137
XREF-Related Command Messages .....	139
PM_CTLGETXREFSTATE.....	139
PM_CTLSETXREFSTATE .....	141
Bookmark-Related Command Messages.....	143
PM_CTLGETBOOKMARKS.....	143
PM_CTLGETRESOURCEINFOSTATE.....	145
Offset-Related Command Messages.....	146
PM_CTLGETOFFSET.....	146
PM_CTLSETOFFSET .....	147
Font-Related Command Messages .....	148

PM_CTLGETBASEFONT .....	148
PM_CTLSETBASEFONT .....	149
Database- and Spreadsheet-Related Messages .....	150
PM_CTLGETCOLWIDTH.....	150
PM_CTLGETROWHEIGHT .....	151
PM_CTLSORT.....	152
Page-Related Command Messages .....	153
PM_CTLGETNUMPAGES .....	153
PM_CTLGETPAGE.....	154
PM_CTLGETPAGESIZE .....	155
PM_CTLSETPAGE .....	156
Printing Command Messages .....	157
PM_CTLPRINT .....	157
PM_CTLPRINTPREVIEW.....	158
PM_CTLVALIDATEMARGINS .....	159
Text-Related Command Messages .....	160
PM_CTLGETSTRING.....	160
PM_CTLSEARCH .....	162
Mouse Command Messages .....	164
PM_CTLGETLMBACTION .....	164
PM_CTLSETLMBACTION .....	165
Device Context-Related Message.....	166
PM_CTLRENDERONTODC .....	166
Formatting Device Related Messages.....	167
PM_CTLSETDEVICE .....	167
PM_CTLGETDEVICE .....	168
Notification Messages Summary.....	169
Control Notification Messages .....	171
PNM_CTLDESTROY.....	171
PNM_CTLDROPFILE.....	172
PNM_CTLHELPSTRING.....	173
PNM_CTLSIZE.....	174
PNM_CTLSTATUS .....	175
Image Notification Messages .....	176
PNM_CTLPAINT .....	176
PNM_CTLREGEN.....	177
PNM_CTLSETFOCUS .....	178
PNM_CTLHSCROLL, PNM_CTLVSCROLL.....	179
Printing Notification Messages.....	180
PNM_CTLPRINT .....	180
PNM_CTLPRINTINGPAGE.....	181
PNM_CTLPRINTPROCESSINGPAGE.....	182
Clipboard-Related Notification Message.....	183
PNM_CTLSETSEL.....	183
PNM_CTLRENDERSEL .....	184
Archive-, Database- and Spreadsheet-Specific Notification Messages.....	185
PNM_CTLARCFILE .....	185
PNM_CTLCOLWIDTH.....	186
PNM_CTLROWHEIGHT .....	187
OLE-Related Notification Message.....	188
PNM_CTLOBJECT .....	188
Link-Related Notification Message .....	189
PNM_CTLLINK .....	189
Views-Related Notification Message .....	190
PNM_CTLSETVIEWEXTENTS.....	190

---

<b>Appendix A: Configuration Options .....</b>	<b>191</b>
<b>INDEX .....</b>	<b>195</b>
<b>Index .....</b>	<b>196</b>
<b>Feedback.....</b>	<b>199</b>

# 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
- ☐ Raster Graphics
- ☐ Spreadsheets
- ☐ Vector Graphics
- ☐ Databases
- ☐ 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.

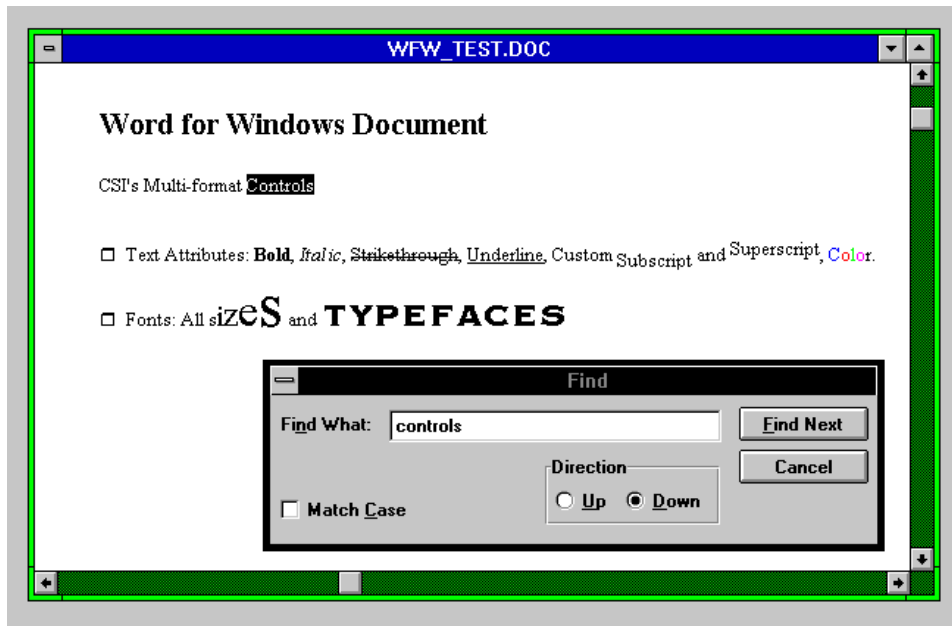


## About this Document

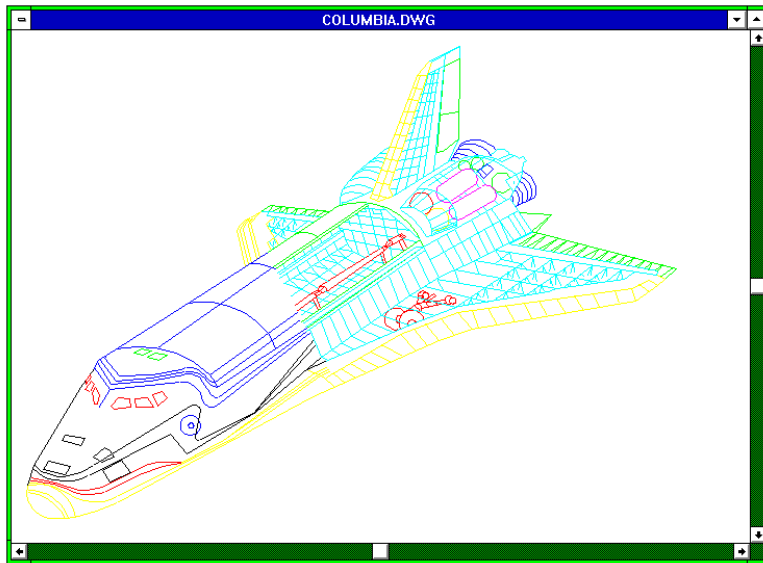
---

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.

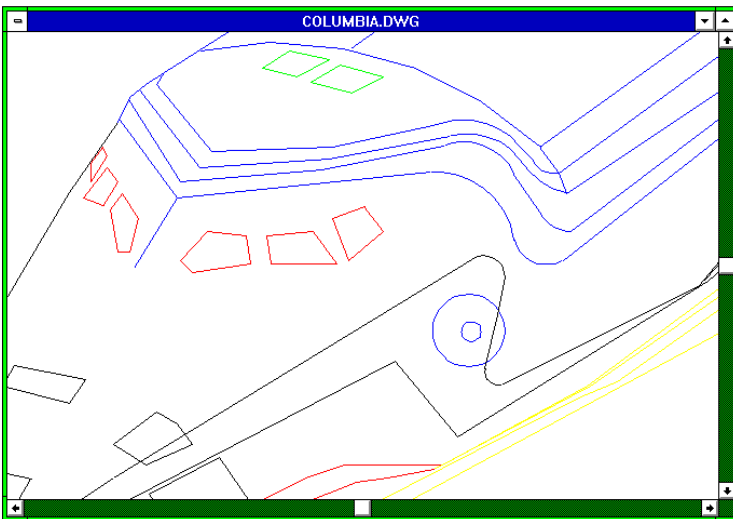
*Here are some of the things that you can do with the controls.*



Perform a text search in a word processor file.



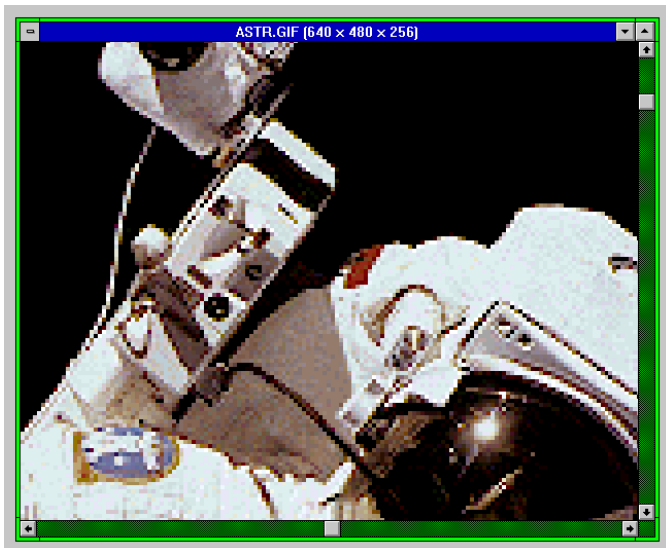
Display CAD files normally...



... or zoom in or out.



Display bitmaps normally...



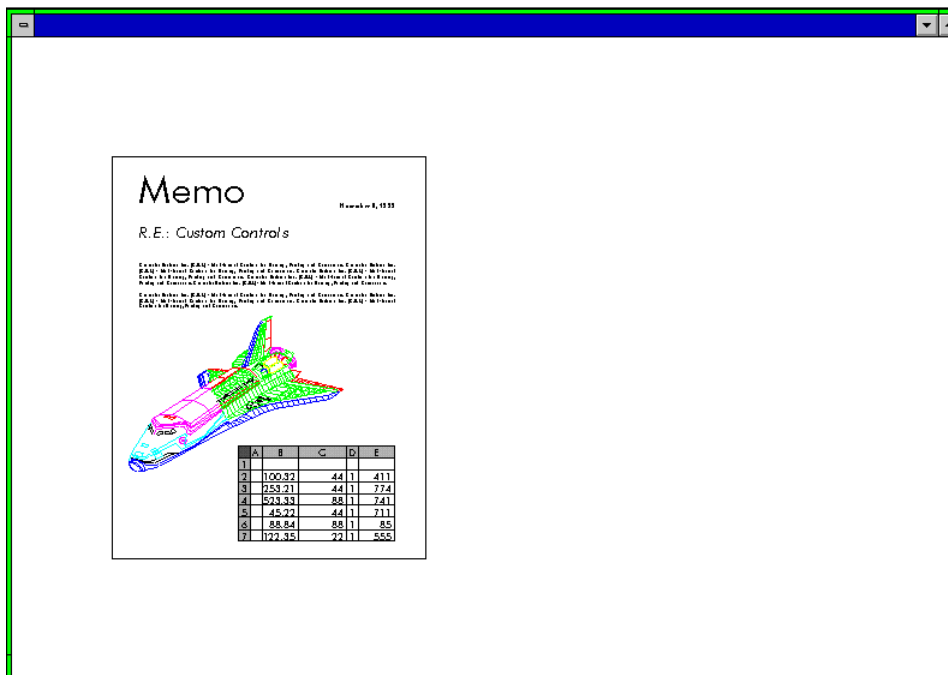
... or zoom in or out.

INVEST.WKS						
	A	B	C	D	E	F
1	Investment Model					
2						
3		Before	After	Annual		
4		tax	tax	asset	Amount	Percent
5	Asset	yield	yield	apprec.	invested	invested
6						
7	Stocks	3.20%	2.27%	9.00%	\$16,000	20.00%
8	Taxable bonds	8.80%	6.44%	3.76%	\$16,000	20.00%
9	Tax-exempt bonds	7.40%	7.80%	2.66%	\$16,000	20.00%
10	Money market	8.80%	6.23%	.00%	\$40,000	40.00%
11						
12	Total				\$96,000	100.00%
13						
14	Total return (we				10.80%	
15						
16						
17	Comments					

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

TEST.ZIP						
32020	Implode	20471	37%	11-23-87	08:50	COLUMBIA.DWG
5896	Implode	2183	63%	06-17-93	10:01	INVEST.WKS
132355	Stored	132355	0%	11-28-91	03:10	ASTR.GIF
51200	Implode	15193	71%	01-07-93	04:15	CEL.SCR

View the contents of archives.



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

# Overview

## Overview

---

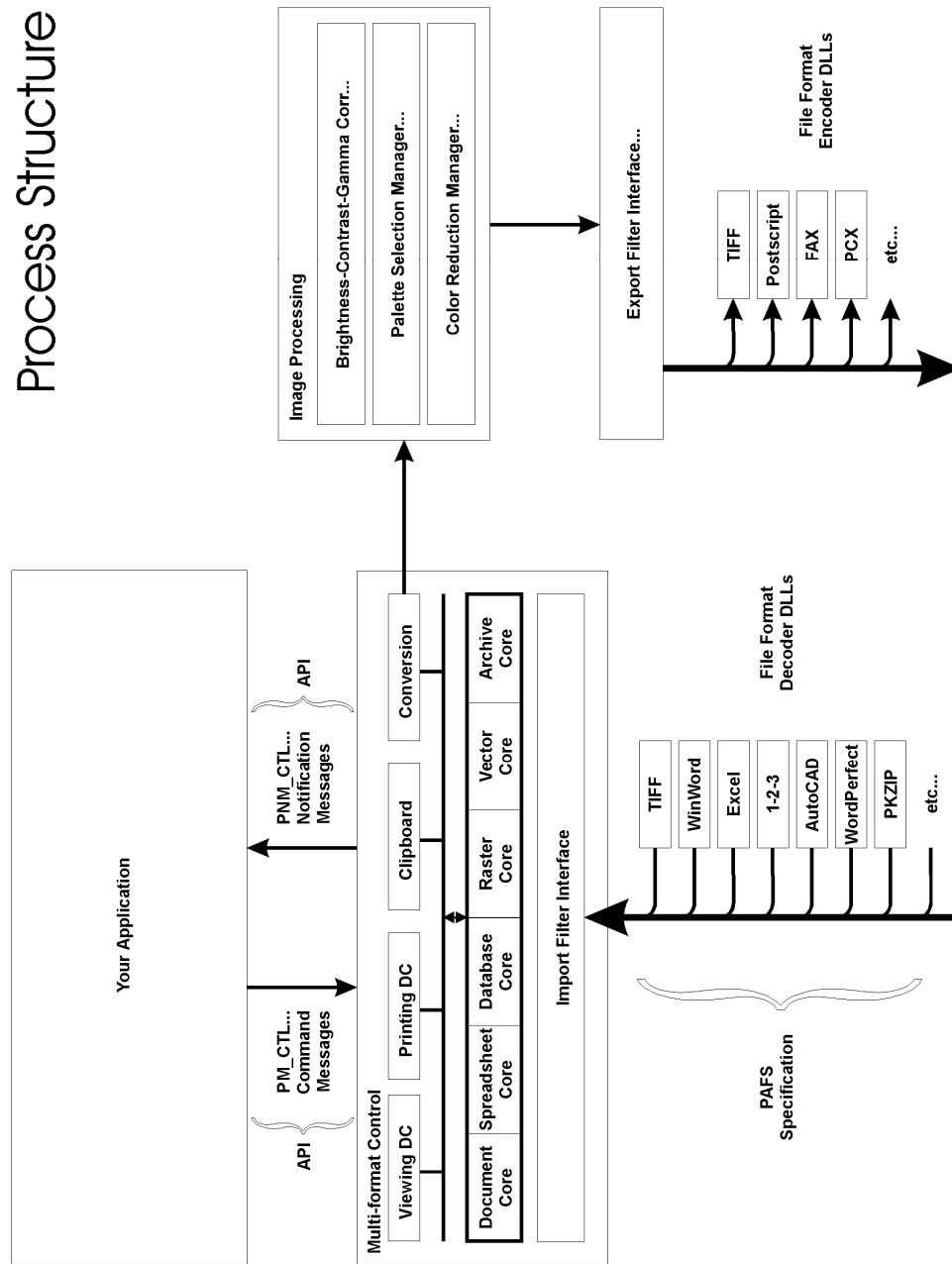
**CSI's Multi-format Controls** enable an application to easily display, print or convert the contents of a given file. There are six types of controls: raster, vector, document, database, spreadsheet, and archive, each capable of handling a specific class of file formats. Depending on how a control 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.

## Process Structure



**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

---

<i>Function Name</i>	<i>Purpose</i>
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()**

<b>Purpose</b>	Loads the CSI Multi-format Control DLLs.
<b>Syntax</b>	<b>int PAN_LoadControls(LPCSTR <i>szInifile</i>                           DWORD <i>dwReserved</i>);</b>
<b>Description</b>	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.
<b>Parameters</b>	<i>szInifile</i> specifies the initialization file which the controls will use to store and retrieve private information. If <b>NULL</b> , the controls will use pctl.ini.  <i>dwReserved</i> is a special variable which must be set to zero (0).
<b>Returns</b>	This function returns <b>TRUE</b> if the libraries were successfully loaded, <b>FALSE</b> otherwise..
<b>Example</b>	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.
}
```

<b>PAN_CreateControl()</b>
----------------------------

**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_SHOWNORMAL);
```

**PAN\_FreeControls()**

<b>Purpose</b>	Unloads the CSI Multi-format Control DLLs.
<b>Syntax</b>	<b>int PAN_FreeControls(void);</b>
<b>Description</b>	Removes the CSI Multi-format Controls from memory. Called by the application before it terminates.
<b>Returns</b>	This function returns <b>TRUE</b> if the libraries were unloaded, <b>FALSE</b> 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()**

<b>Purpose</b>	Returns the error code of the last error.
<b>Syntax</b>	<b>int PAN_GetCtlErrorCode(void);</b>
<b>Description</b>	All control-specific error codes begin with the prefix <b>PAN_CTLERR</b> .
<b>Parameters</b>	None.
<b>Returns</b>	Error code of last error (see list below).
<b>Example</b>	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 encountered 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 message PAN_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::wstring & GetName () const;
        // Get the bookmark name

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

```

---

<sup>(1)</sup> 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;

class CPanBlock
{
public:
    CPanBlock(      id_type      id      = 0,
                  const std::(1)wstring & wsName= std::wstring());
    // Constructor
    id_type GetId() const;
    // Get the block id
    void SetId(id_type id);
    // Set the block id
    const std::wstring & GetName() const;
    // Get the block name
    void SetName(const std::wstring & wsName);
    // Set the block name
};

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;

```

---

<sup>(1)</sup> Look at Overview section about usage of STL and Boost libraries

[illegible]

```

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

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

typedef struct PAN_CtlSearchInfo {
    BOOL                fMulti;                // IN: Find all occurrences, 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)
};

```

---

<sup>(1)</sup> Look at Overview section about usage of STL and Boost libraries

```

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::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 occurrences
        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

```

---

<sup>(1)</sup> Look at Overview section about usage of STL and Boost libraries

```

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
};

typedef struct PAN_CtlEntity {
int                inslevel;        // Corresponds 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, ... correspond 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      imageRecls;      // OUT: global handle to array
                                // PAN_CtlRange structures: image
                                // rectangles within page rectangles
    HGLOBAL      clipRecls;       // OUT: global handle to array of RECT
                                // structures in device units: page
                                // rectangles excluding margins and
                                // 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 CPanLayer
{
public:
    enum Polarity {
        NegativePolarity,
        PositivePolarity
    };

    CPanLayer(      id_type      id          = 0,
                   const std::wstring & wsName = std::wstring(),
                   color_type color         = 0,
                   bool bVisible            = false,
                   bool bThawed             = false,
                   bool bReadOnly            = false,
                   bool bPrintable           = true,
                   Polarity ePolarity        = PositivePolarity);

    // Constructor
    id_type GetId() const;
    // Get the layer id
    void SetId(id_type id);
    // Set the layer id
    const std::wstring & GetName() const;
    // Get the layer name
    void SetName(const std::wstring & wsName);
    // Set the layer name
    color_type GetColor() const;
    // Get the layer color
    void SetColor(color_type color);
    // Set the layer color
    bool IsVisible() const;
    // Whether the layer is ON or OFF
    void SetVisible(bool bVisible);
    // Change the layer visibility

```

---

<sup>(1)</sup> Look at Overview section about usage of STL and Boost libraries

```

    bool IsThawed() const;
        // Whether the layer is thawed
    void SetThawed(bool bThawed);
        // Thaw this layer
    bool IsReadOnly() const;
        // Whether this layer is read only (not edibale)
    void SetReadOnly(bool bReadOnly);
        // Make this layer read only (not editable)
    bool IsPrintable() const;
        // Whether this layer can be printed
    void SetPrintable(bool bPrintable);
        // Set this layer to printable or not
    Polarity GetPolarity() const;
        // Whether this layer has negative polarity and should be inverted
    void SetPolarity(Polarity ePolarity);
        // Set the polarity of this layer
};

typedef struct PAN_VIEW {
    id_type      id;                // unique identifier
    char         name[STRLEN80];    // name of view
    PAN_Point    vmin, vmax;        // Window coordinates;
    Hmatrix      view;              // transformation matrix
    DWORD        dwFlags;           // Additional flags (see PAN_VIEWFLAG_*)
} PAN_VIEW;

class CPanView
{
public:
    CPanView();
        // Default constructor
    CPanView(      id_type      id,
                  const std::wstring & wsName,
                  const PAN_Range & extents,
                  const Hmatrix & transform,
                  DWORD        dwFlags);
        // Constructor
    id_type GetId() const;
        // Get the view id
    void SetId(id_type id);
        // Set the view id
    const std::wstring & GetName() const;
        // Get the view name
    void SetName(const std::wstring & wsName);
        // Set the view name
    const PAN_Range & GetExtents() const;
        // Get the view extents
    void SetExtents(const PAN_Range & extents);
        // Set the view extents
    const Hmatrix & GetTransform() const;
        // Get the view transform
    void SetTransform(const Hmatrix & transform);
        // Set the view transform

```

---

<sup>(1)</sup> Look at Overview section about usage of STL and Boost libraries

```
    DWORD GetFlags() const;
        // Get the view flags (see PAN_VIEWFLAG_*)
    void SetFlags(DWORD flags);
        // Set the view flags (see PAN_VIEWFLAG_*)
};
```



```

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
        }
};

```

---

<sup>(1)</sup> 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

```

---

<sup>(1)</sup> Look at Overview section about usage of STL and Boost libraries

```

        // to the next selection.

}; // End of SelectionIterator

virtual ~IPanResourceInfo();           // Destructor

virtual const std::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::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
}
```

---

<sup>(1)</sup> 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::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
    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)

```

---

<sup>(1)</sup> 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
<b>PM_CTLCLONECONTROL</b>	Creates a duplicate of a control and returns its handle.
<b>PM_CTLDESTROY</b>	Prepares a control for destruction.
<b>PM_CTLGETDIMS</b>	Returns the dimensions of the control
<b>PM_CTLGETCAPS</b>	Returns the currently enabled capabilities of a control.
<b>PM_CTLGETFILE</b>	Returns information about the current file.
<b>PM_CTLGETFILEFMTS</b>	Returns file formats supported by a control.
<b>PM_CTLGETFILETYPE</b>	Returns the format class and an index into the list provided by PM_CTLGETFILEFMTS
<b>PM_CTLGETINFO</b>	Returns information about a control.
<b>PM_CTLGETMODE</b>	Returns the currently enabled modes of a control.
<b>PM_CTLGETOPTION</b>	Returns the control options.
<b>PM_CTLGETSTATUS</b>	Returns the current status of a control.
<b>PM_CTLREGEN</b>	Reread the file from disk.
<b>PM_CTLSETCAPS</b>	Modifies the capabilities of a control.
<b>PM_CTLSETFILE</b>	Renders the given file in the control window.
<b>PM_CTLSETMODE</b>	Modifies the modes of a control.
<b>PM_CTLSETOPTION</b>	Set the control options.

### Presentation Command Messages

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

### Conversion Command Message

<b>PM_CTLCONVERT</b>	Obtain handles to conversion functions.
----------------------	---

### Coordinates-related messages

<b>PM_CTLCARETTOORLD</b>	Returns the view coordinate that corresponds to the specified caret position.
<b>PM_CTLGETCARETPOS</b>	Returns the current position of the caret.
<b>PM_CTLSETCARETPOS</b>	Moves the caret to the specified position.
<b>PM_CTLWORLDTOCARET</b>	Returns the caret position that corresponds to the given world coordinates.
<b>PM_CTLWORLDTOCLIENT</b>	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.
<b>PM_XFRMRECT</b>	Applies/Removes current rotation and flipping to/from a world coordinate range.

## Clipboard and selection-related messages

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

## Color-related messages

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

## Views-related message

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

## Blocks-related messages

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

## Entities-related

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

## Extended Image Data Command Messages

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

## Layers-Related Command Messages

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

## XRef-Related Command Messages

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

## Bookmark messages.

<b>PM_CTLGETBOOKMARKS</b>	Returns a buffer of PAN_BOOKMARK struct set by PANX_SetBookMarks.
<b>PM_CTLSETBOOKMARKS</b>	Not implemented

## Resource-Related Command Messages

<b>PM_CTLGETRESOURCEINFOSTATE</b>	Returns a buffer of resource information set by PANX_ResourceInfo().
-----------------------------------	--

## Offset-Related Command Messages

<b>PM_CTLGETOFFSET</b>	Returns the offset of the origin for the current file in view coordinates.
<b>PM_CTLSETOFFSET</b>	Sets the offset of the origin for the current file in view coordinates.

## Font-Related Command Messages

<b>PM_CTLGETBASEFONT</b>	Returns the base font of the current file.
<b>PM_CTLSETBASEFONT</b>	Sets the base font of the current file.

## Database and Spreadsheet-Related Command Messages

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

#### Page-Related Command Messages

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

#### Printing-Related Command Messages

<b>PM_CTLPRINT</b>	Prints the specified extents.
<b>PM_CTLPRINTPREVIEW</b>	Returns the print preview of the specified extents.
<b>PM_CTLVALIDATEMARGINS</b>	Validates the printer margins.

#### Text-Specific Command Messages

<b>PM_CTLGETSTRING</b>	Retrieves a specified string from a control.
<b>PM_CTLSEARCH</b>	Search the current file for a text string.

#### Mouse-Specific Command Messages

<b>PM_CTLGETLMBACTION</b>	Returns the left mouse button control window behavior.
<b>PM_CTLSETLMBACTION</b>	Sets the left mouse button control window behavior.

#### Device Context Command Messages

<b>PM_CTLRENDERONTODC</b>	Renders the specified extents onto the given device context.
---------------------------	--

#### Formatting Device Command Messages

<b>PM_CTLSETDEVICE</b>	Set formatting device for document files.
<b>PM_CTLGETDEVICE</b>	Gets current formatting device for document files.

#### Command Messages dropped from Version 1.1 of VCET Controls

<b>PM_CTLGETBOOKMARK</b>	Returns the bookmark position within the current file.
<b>PM_CTLGETDC</b>	Returns a handle to the device context set by PM_CTLSETDC.
<b>PM_CTLSETBOOKMARK</b>	Positions the bookmark at a given file position.
<b>PM_CTLSETDC</b>	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**

<b>Purpose</b>	Prepares a control for destruction.
<b>Parameters</b>	<i>wParam:</i> not used <i>lParam:</i> not used
<b>Returns:</b>	error code
<b>Compatibility:</b>	all control types
<b>Special Notes:</b>	This message must be sent before destroying the control window with the Windows <b>DestroyWindow()</b> 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.

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

<b>Parameters</b>	<i>wParam:</i> <b>Zero</b> Get all enabled capabilities of the control. <b>Non-Zero</b> Get all possible capabilities of the control.
	<i>lParam:</i> ( <b>wParam:Zero</b> )      ( <b>LPDWORD</b> ) mask of all currently enabled capabilities of the control ( <b>wParam:Non-Zero</b> )      ( <b>LPDWORD</b> ) 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**

<b>Purpose</b>	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.
<b>Description</b>	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 <i>PAN_CtlDimensions</i> structure pointed to by <i>lParam</i> .
<b>Parameters</b>	<i>wParam</i> : not used  <i>lParam</i> : ( <b>PAN_CtlDimensions *</b> ) dimensions
<b>Returns:</b>	error code
<b>Compatibility:</b>	all control types
<b>Special Notes:</b>	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);
}
```

<b>PM_CTLGETFILEFMTS</b>
--------------------------

<b>Purpose</b>	Returns file formats supported by a control.
<b>Description</b>	Return a description of each file format supported by the control in the <b>PAN_CtlFileFmtList</b> structure pointed to by <i>lParam</i> .
<b>Parameters</b>	<div> <i>wParam</i>: not used </div> <div> <i>lParam</i>: (<b>PAN_CtlFileFmtList *</b>) structure to be filled with the formats information. </div>
<b>Returns:</b>	error code
<b>Compatibility:</b>	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**

<b>Purpose</b>	Returns the type of control associated with a given file.		
<b>Description</b>	The control types are listed below.		
<b>Parameters</b>	<i>wParam:</i>	0	The file name encoded in MultiByte using the local system codepage.
		1	The file name encoded in Unicode
	<i>lParam:</i>	( <i>wParam</i> = 0)	(LPCSTR) filename
		( <i>wParam</i> = 1)	(wchar_t *) filename
<b>Returns</b>	<p>A constant of type <b>PAN_FileType</b> is returned in the low word, and an index into the list provided by <b>PM_CTLGETFILEFMTS</b> is returned in the high word. If the format of the given file is not supported or an error occurred, <b>PAN_UnknownFile</b> is returned.</p> <pre>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;</pre>		
<b>Compatibility:</b>	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) LOWORD(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**

<b>Purpose</b>	Returns information about a control.
<b>Description</b>	Returns type and version information about the control in the <b>PAN_CtlInfo</b> structure pointed to by <i>lParam</i> .
<b>Parameters</b>	<i>wParam</i> : not used  <i>lParam</i> : ( <b>PAN_CtlInfo *</b> ) control information
<b>Returns:</b>	error code
<b>Compatibility:</b>	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);
}
```

<b>PM_CTLGETMODE</b>
----------------------

**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.

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

*lParam:* (**wParam = PAN\_CTLCAOPTIONS**)

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:

<b>PAN_CTLSTATUSIDLE</b>	control is idle
<b>PAN_CTLSTATUSPROCESSING</b>	control is processing the file
<b>PAN_CTLSTATUSREFRESHING</b>	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

<b>Purpose</b>	Set the capabilities of a control.
----------------	------------------------------------

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

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

<b>Parameters</b>	<i>wParam:</i>	<b>Zero</b>	enable capabilities specified by the mask (disable those not specified)
		<b>Non-Zero</b>	enable all capabilities supported by the control
	<i>lParam:</i>	<b>(wParam:Zero)</b>	<b>(DWORD)</b> mask of capabilities
		<b>(wParam:Non-Zero)</b>	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**

<b>Purpose</b>	Renders the given file in the control window.	
<b>Description</b>	Render the given file in the control window. If <i>wParam</i> is not -1, it must be an index into the list of file formats returned by <b>PM_CTLGETFILEFMTS</b> . Otherwise, the first compatible format will be used.	
<b>Parameters</b>	<i>wParam</i>	file format index or -1
	<i>lParam</i> :	(LPCSTR) filename
<b>Returns:</b>	error code	
<b>Compatibility:</b>	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**

<b>Purpose</b>	Renders the given file in the control window.	
<b>Description</b>	This is an enhanced version of the message <b>PM_CTLSETFILE</b> that sets a file in the control window supporting <b>UNICODE</b> and application callback procedures.	
<b>Parameters</b>	<i>wParam</i>	The file format index from the list of file formats returned by <b>PM_CTLGETFILEFMTS</b> , otherwise -1
	<i>lParam</i> :	( <b>PAN_CTLSetFile *</b> ) filename and callback
<b>Returns:</b>	error code	
<b>Compatibility:</b>	all control types	
<b>Special Notes</b>	This message has been designed to support <b>UNICODE</b> file names. However, not all formats support <b>UNICODE</b> . For those, which do not, the <b>UNICODE</b> string will be converted to <b>MULTIBYTE</b> using the local system code page, then used to load the file. Despite this limitation, <b>UNICODE</b> resource names will be sent back to the application for user resource location. This allows the application to locate <b>UNICODE</b> 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());
        }
    }
}
```

---

<sup>(1)</sup> 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);
}
```

---

<sup>(1)</sup> 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 initially 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.

---

<sup>(1)</sup> 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::Locations**.

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:

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

*lParam:* (**wParam = PAN\_CTLCAOPTIONS**)  
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\_PAFSEXPOR\_T\_IDENTIFYIMAGE\_HANDLE
- GET\_PAFSEXPOR\_T\_QUERYIMAGE\_HANDLE
- GET\_PAFSEXPOR\_T\_IDENTIFY\_HANDLE
- GET\_PAFSEXPOR\_T\_QUERYFORMAT\_HANDLE
- GET\_PAFSEXPOR\_T\_QUERYSUBFORMAT\_HANDLE
- GET\_PAFSEXPOR\_T\_CONVERTFILE\_HANDLE
- GET\_PAFSEXPOR\_T\_CONVERT\_HANDLE
- GET\_PAFSEXPOR\_T\_OPTIMIZEPALETTE\_HANDLE
- GET\_PAFSEXPOR\_T\_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\_PAFSEXPOR\_T\_IDENTIFYIMAGE\_HANDLE

Returns a pointer to the following function:

```
BOOL PAFSEXPOR_T_IdentifyImageDLL(LPWORD lpNumDlls);
```

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

#### GET\_PAFSEXPOR\_T\_QUERYIMAGE\_HANDLE

Returns a pointer to the following function:

```
BOOL PAFSEXPOR_T_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\_PAFSEXPOR\_T\_IDENTIFY\_HANDLE

Returns a pointer to the following function:

```
BOOL PAFSEXPOR_T_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\_PAFSEXPOR\_QUERYFORMAT\_HANDLE

Returns a pointer to the following function:

```
BOOL PAFSEXPOR_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\_PAFSEXPOR\_QUERYSUBFORMAT\_HANDLE

Returns a pointer to the following function:

```
BOOL PAFSEXPOR_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\_PAFSEXPOR\_CONVERTFILE\_HANDLE

Returns a pointer to the following function:

```
BOOL PAFSEXPOR_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\_PAFSEXPOR\_CONVERT\_HANDLE

Returns a pointer to the following function:

```
BOOL PAFSEXPOR_Convert(LPCVTPANX cvcbs, DWORD dwFormat, DWORD dwSubFormat, LPCSTR szInFileName, LPCSTR szOutFileName);
```

This function calls **PAFSEXPOR\_ConvertFile**, but will first render the file if needed.

#### GET\_PAFSEXPOR\_OPTIMIZEPALETTE\_HANDLE

Returns a pointer to the following function:

```
BOOL PAFSEXPOR_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\_PAFSEXPOR\_REDUCECOLORS\_HANDLE

Returns a pointer to the following function:

```
BOOL PAFSEXPOR_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

<b>Parameters</b>	<i>wParam</i>	not used
	<i>lParam</i>	( <b>int *</b> ) flip mode, one of: <b>PAN_CTLFLIPNONE</b> <b>PAN_CTLFLIPX</b> <b>PAN_CTLFLIPY</b> <b>PAN_CTLFLIPXY</b>

**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**

<i>wParam</i>	not used
<i>lParam</i>	<b>(int *)</b> 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**

<i>wParam:</i>	<b>(WORD)</b> zoom flag, one of: <b>PAN_CTLZOOMX</b> <b>PAN_CTLZOOMY</b> <b>PAN_CTLZOOMBOTH</b>
<i>lParam:</i>	<b>(LPDWORD)</b> 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**

<i>wParam:</i>	<b>(WORD)</b> zoom flag, one of: <b>PAN_CTLZOOMX</b> <b>PAN_CTLZOOMY</b> <b>PAN_CTLZOOMBOTH</b>
<i>lParam:</i>	<b>(DWORD)</b> 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

<b>Purpose</b>	Converts a caret based position to a view position.
<b>Description</b>	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 <b>NULL</b> if the translation could not be done (non-existent caret position).
<b>Parameters</b>	<p><i>wParam:</i> not used</p> <p><i>lParam:</i> (struct PAN_CtlCaretPos *) lpCaretPos;</p>
<b>Returns:</b>	(struct PAN_CtlPos *) lpViewPos
<b>Compatibility:</b>	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**

<b>Purpose</b>	Clears all selections.	
<b>Description</b>	Clear all selections in the current file.	
<b>Parameters</b>	<i>wParam:</i>	not used
	<i>lParam:</i>	not used
<b>Returns:</b>	error code	
<b>Compatibility:</b>	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;
}
```

<b>PM_CTLCLIENTTOWORLD</b>
----------------------------

<b>Purpose</b>	Performs a client to world coordinates conversion.
<b>Description</b>	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).
<b>Parameters</b>	<i>wParam</i> : not used  <i>lParam</i> : (PAN_CtlPos *) client coordinates
<b>Returns:</b>	error code
<b>Compatibility:</b>	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**

<b>Purpose</b>	Returns the current position of the caret within a text stream.
<b>Description</b>	This message allows an application to obtain the current position of the caret in the control.
<b>Parameters</b>	<i>wParam:</i> not used  <i>lParam:</i> <b>(PAN_CtlCaretPos *)</b> caret position
<b>Returns:</b>	error code
<b>Compatibility:</b>	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**

<b>Purpose</b>	Sets the caret to the specified position
<b>Description</b>	The position is specified as an offset within a flow on a specified page.
<b>Parameters</b>	<i>wParam:</i> not used. <i>lParam:</i> (PAN_CtlCaretPos *) lpCaretPos;
<b>Returns:</b>	error code
<b>Compatibility:</b>	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**

<b>Purpose</b>	Converts a view based position to a caret position.
<b>Description</b>	This message allows the caller to convert a view based coordinate into a caret based position.
<b>Parameters</b>	<i>wParam:</i> not used <i>lParam:</i> (struct PAN_CtlPos *) lpViewPos
<b>Returns:</b>	(struct PAN_CtlCaretPos *) lpCaretPos;
<b>Compatibility:</b>	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**

<b>Purpose</b>	Performs a world to client coordinates conversion.
<b>Description</b>	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.
<b>Parameters</b>	<i>wParam</i> : not used  <i>lParam</i> : ( <b>PAN_CtlPos *</b> ) world coordinates
<b>Returns:</b>	error code
<b>Compatibility:</b>	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**

<i>wParam:</i>	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**

<b>Purpose</b>	Copy the current selections to the clipboard.
<b>Description</b>	Copy the current selections to the clipboard using the clipboard formats specified in the given mask of <b>PAN_CtlClpbrd</b> constants.
<b>Parameters</b>	<i>wParam</i> : not used <i>lParam</i> : <b>(DWORD)</b> format(s)
<b>Returns:</b>	error code
<b>Compatibility:</b>	all control types

**Example:**

```
void CopyToClipboard(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**

<b>Purpose</b>	Returns clipboard formats supported by a control.		
<b>Description</b>	Return a description of each clipboard format supported by the control in the <b>PAN_CtlClpbrdFmtList</b> structure pointed to by <i>lParam</i> .		
<b>Parameters</b>	<i>wParam</i> :	not used	
	<i>lParam</i> : clipboard formats	<b>(PAN_CtlClpbrdFmtList *)</b>	supported
<b>Returns:</b>	error code		
<b>Compatibility:</b>	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**

<b>Purpose</b>	Returns the current number of selections		
<b>Description</b>	Returns the current number of selections in the <b>WORD</b> pointed to by <i>lParam</i> . 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.		
<b>Parameters</b>	<i>wParam</i> :	not used	
	<i>lParam</i> :	<b>(LPWORD)</b> number of selections	
<b>Returns:</b>	error code		
<b>Compatibility:</b>	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**

<b>Purpose</b>	Returns a list of the current selections.	
<b>Description</b>	Return a list of the current selections for the current file in the <b>PAN_CtlSelList</b> structure pointed to by <i>lParam</i> .	
<b>Parameters</b>	<i>wParam</i> :	not used
	<i>lParam</i> :	( <b>PAN_CtlSelList *</b> ) pointer to selections list
<b>Returns:</b>	error code	
<b>Compatibility:</b>	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**

<b>Purpose</b>	Sets or removes a selection based on view coordinates.
<b>Description</b>	If the selection flag is <b>TRUE</b> , set the given selection in the current file, otherwise, remove the given selection if it exists.
<b>Parameters</b>	<i>wParam</i> <b>(BOOL)</b> selection flag  <i>lParam</i> : <b>(const PAN_CtlRange *)</b> selection
<b>Returns:</b>	error code
<b>Compatibility:</b>	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

<b>Purpose</b>	Returns the foreground or background color.		
<b>Description</b>	If <i>wParam</i> is zero, return the current foreground color. If <i>wParam</i> is non-zero, return the current background color.		
<b>Parameters</b>	<i>wParam</i> :	<b>0</b>	Foreground Request
		<b>≠0</b>	Background Request
	<i>lParam</i> :	<b>(LPCOLORREF)</b> color	
<b>Returns:</b>	error code		
<b>Compatibility:</b>	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**

<b>Purpose</b>	Returns palette information.
<b>Description</b>	Return the color depth of the current file. If <i>lParam</i> points to a <b>LOGPALETTE</b> structure, the size of which corresponds to the number of colors, it is filled with the file's palette, if any.
<b>Parameters</b>	<i>wParam</i> :           not used  <i>lParam</i> : <b>(LOGPALETTE *)</b> logical palette
<b>Returns:</b>	<b>(WORD)</b> color depth on success or -1 on failure
<b>Compatibility:</b>	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**

<b>Purpose</b>	The calling application must send this message to the control whenever it receives a <b>WM_PALETTECHANGED</b> message.		
<b>Description</b>	The window handle specified in <i>wParam</i> is the same window handle as that specified in <i>wParam</i> for the <b>WM_PALETTECHANGED</b> message. Handled identically as a <b>WM_PALETTECHANGED</b> message.		
<b>Parameters</b>	<i>wParam</i> : the palette.	<b>(HWND)</b> Handle of the window which	changed
	<i>lParam</i> :	not used	
<b>Returns:</b>	error code		
<b>Compatibility:</b>	all control types		

**PM\_CTLQUERYNEWPALETTE**

<b>Purpose</b>	The calling application must send this message to the control whenever it receives a <b>WM_QUERYNEWPALETTE</b> message. Handled identically to <b>WM_QUERYNEWPALETTE</b> message. Useful for debugging purposes.		
<b>Parameters</b>	<i>wParam:</i>	<b>(HWND)</b> Handle of window for which to realize the new palette.	
	<i>lParam:</i>	not used	
<b>Returns:</b>	error code		
<b>Compatibility:</b>	all control types		

**PM\_CTLSETFGBGCOLOR**

<b>Purpose</b>	Sets the foreground or background color.
<b>Description</b>	Sets the foreground/background color. Can be done with or without updating the control window.
<b>Parameters</b>	<i>wParam:</i> 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.  <i>lParam:</i> <b>(const COLORREF *)</b> color
<b>Returns:</b>	error code
<b>Compatibility:</b>	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**

<b>Purpose</b>	Sets the color palette.
<b>Description</b>	Set the color palette of the current file.
<b>Parameters</b>	<i>wParam:</i> not used <i>lParam:</i> ( <b>LOGPALETTE *</b> ) logical palette
<b>Returns:</b>	error code
<b>Compatibility:</b>	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**

**MultiByte:**

*wParam:* 0  
*lParam:* (PAN\_VIEW \*) current view

**Unicode:**

*wParam:* -1  
*lParam:* (CPanView \*) current view

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

**Compatibility:** vector.

**Example:**

```
void ShowMultiByteViewName(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) &curView);
    if (ViewStatus) {
        Output("Current view name = %s\n", curView.name);
    } else {
        Output("No named view currently defined\n");
    }
}

void ShowUnicodeViewName(HWND ctrlHandle)
/*    Display the Unicode name of the view currently being viewed. */
{
    BOOL ViewStatus;
    CPanView curView;

    if (ctrlHandle == NULL) return;

    ViewStatus = SendMessage(ctrlHandle, PM_CTLGETVIEW,
                             (WPARAM)-1, (LPARAM) &curView);
    if (ViewStatus) {
        Output(L"Current view name = %s\n",
              curView.GetName().c_str());
    } else {
        Output(L"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**

<i>wParam:</i>	not used
<i>lParam:</i>	( <b>PAN_CtlRange *</b> ) 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)
        Output("VIEWEXTENTS: min = (%f, %f, %f)\n", rg.min.x, rg.min.y,          &rg);
        Output("VIEWEXTENTS: max = (%f, %f, %f)\n", rg.max.x, rg.max.y,      rg.min.z);
        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**

**MultiByte:**

*wParam:*            **(int)** number of views to return  
*lParam:*            **(PAN\_VIEW \*)** view buffer

**Unicode:**

*wParam:*            **(int)** -1  
*lParam:*            **(std::vector<CPanView> \*)** view buffer

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

**Compatibility:**    vector.

**Example:**

```
void ShowAvailViewsMultiByte(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\n", indx, ViewBuf[indx].name);
        }
    }
}
```

```
void ShowAvailViewsUnicode(HWND ctrlHandle)
{
    int numAvail;
    std::vector<CPanView> ViewVector;

    numAvail = SendMessage(ctrlHandle, PM_CTLGETVIEWNAMES, -1,
                           (LPARAM) &ViewVector);
    if (numAvail <= 0) {
        Output("No named views available.\r");
    } else {
```

---

<sup>(1)</sup> Look at Overview section about usage of STL and Boost libraries

```
        int indx;
        for (indx = 0; indx < numAvail; indx++) {
            Output(L"View %d = %s\n", indx,
                ViewVector[indx].GetName().c_str());
        }
    }
```

**PM\_CTLSETVIEW**

<b>Purpose</b>	Sets the view to use in the control.
<b>Description</b>	<p>If a vector decoder supports different views, it can determine which view to render through the <b>PANX_GetView</b> callback. This message allows the application to choose the view that will be decoded.</p> <p>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.</p>
<b>Parameters</b>	<p>use default view (the one defined with PAN_VIEW.id=-1)</p> <p><i>wParam:</i>           <b>(WORD)</b> -1  <i>lParam:</i>           unused value</p> <p>MultiByte user view provided in lParam</p> <p><i>wParam:</i>           <b>(WORD)</b> -2  <i>lParam:</i>           <b>(PAN_VIEW *)</b> view info</p> <p>Unicode user view provided in lParam</p> <p><i>wParam:</i>           <b>(WORD)</b> -3  <i>lParam:</i>           <b>(CPanView *)</b> view info</p> <p>Select a predefined view</p> <p><i>wParam:</i>           <b>(WORD)</b> index of the view defined in the control.  <i>lParam:</i>           unused value</p>
<b>Returns:</b>	error code
<b>Compatibility:</b>	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

<b>Purpose</b>	Sets the view extents of the current file in view coordinates.	
<b>Description</b>	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.	
<b>Parameters</b>	<i>wParam:</i>	not used
	<i>lParam:</i>	(const PAN_CtlRange *) extents
<b>Returns:</b>	error code	
<b>Compatibility:</b>	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

<b>Purpose</b>	Gets the “active block” from the control.	
<b>Description</b>	Get the current “active block” for the current page and set it in the <b>long</b> pointed to by <i>lParam</i> .	
<b>Parameters</b>	<i>wParam:</i>	Not used
	<i>lParam:</i>	<b>(long *)</b> current block. Set to zero if the page contains no blocks or if the whole page is active.
<b>Returns:</b>	error code	
<b>Compatibility:</b>	vector	
<b>Example:</b>		

```
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**

<b>Purpose</b>	Returns the buffer of block names set by the decoder.
<b>Description</b>	Get the names of the number of blocks specified in <i>wParam</i> and set them in the location specified by <i>lParam</i> .
<b>Parameters</b>	<p><b>MultiByte:</b></p> <p><i>wParam</i>:        <b>(WORD)</b> number of block names to retrieve.</p> <p><i>lParam</i>:        <b>(PAN_BLOCK *)</b> block structure</p> <p><b>Unicode:</b></p> <p><i>wParam</i>:        <b>(WORD)</b> must be -1</p> <p><i>lParam</i>:        <b>(std:: <sup>(1)</sup>vector&lt;CPanBlock&gt; *)</b> blocks</p>
<b>Returns:</b>	<p>Number of blocks returned</p> <p>Number of total blocks if <i>lParam</i> is NULL</p> <p>0 if no blocks exist</p> <p>-1 if error</p>
<b>Compatibility:</b>	vector
<b>Restrictions:</b>	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;
}

```

---

<sup>(1)</sup> Look at Overview section about usage of STL and Boost libraries

---

}



**PM\_CTLSETBLOCK**

<b>Purpose</b>	Sets the current “active block” of the page.
<b>Description</b>	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 <b>PANX_GetBlock</b> callback.
<b>Parameters</b>	<i>wParam:</i> <b>(WORD)</b> block number, 0 for the whole page  <i>lParam:</i> not used
<b>Returns:</b>	error code
<b>Compatibility:</b>	vector
<b>Special Notes:</b>	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

<b>Purpose</b>	Used with CAD decoders to obtain identifiers to objects within a specified region of the viewed drawing.
<b>Description</b>	<p>When displaying a vector file that supports entities, this message provides a means of obtaining all handles within a specified region.</p> <p>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.</p>
<b>Parameters</b>	<p><i>wParam</i>: reserved, must be 0</p> <p><i>lParam</i>: (PAN_CtlGetEntityInfo *) lpGetEntity</p>
<b>Returns:</b>	error code
<b>Compatibility:</b>	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**

<b>Purpose</b>	Used with CAD decoders to set the drawing color of a given entity e.g. to highlight a selected entity.
<b>Description</b>	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.
<b>Parameters</b>	<p><i>wParam:</i> Negative of the number of elements pointed to by <i>lParam</i>. 0 or -1 indicate 1 element in the list.</p> <p><i>lParam:</i> (<b>PAN_CtlShowEntity *</b>) pShowEntity NULL indicates that all current selections should be removed.</p>
<b>Returns:</b>	error code
<b>Compatibility:</b>	vector-2D
<b>Restrictions:</b>	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

<b>Purpose</b>	Retrieves the current contrast/anti-aliasing setting of an image.
<b>Description</b>	This message only applies to monochrome raster images. All other image classes return <b>PAN_CTLERRNOTCOMPATIBLE</b> .
<b>Parameters</b>	<p><i>wParam</i>: <b>(WORD)</b> input, one of:  <b>PAN_IMAGE_CONTRAST</b>,  <b>PAN_IMAGE_ANTIALIAS</b>,  <b>PAN_IMAGE_INVERT</b> or  <b>PAN_IMAGE_BRIGHTNESS</b>.</p> <p><i>lParam</i>: <b>(wParam = PAN_IMAGE_CONTRAST)</b>  <b>(int*)</b> output receiving a contrast value between -100 to 100.  <b>(wParam = PAN_IMAGE_ANTIALIAS)</b>  <b>(BOOL*)</b> output receiving <b>TRUE</b> or <b>FALSE</b> indicating the current anti-aliasing mode.  <b>(wParam = PAN_IMAGE_INVERT)</b>  <b>(BOOL*)</b> output receiving <b>TRUE</b> or <b>FALSE</b> indicating the current inversion mode.  <b>(wParam = PAN_IMAGE_BRIGHTNESS)</b>  <b>(int*)</b> output receiving a brightness value between -100 to 100.</p>
<b>Returns</b>	error code
<b>Compatibility:</b>	raster and vector-2D controls
<b>Restrictions</b>	<b>PAN_IMAGE_INVERT</b> is only used for monochrome raster images and <b>PAN_IMAGE_BRIGHTNESS</b> 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**

<b>Purpose</b>	Adjusts the contrast/anti-aliasing setting on an image.
<b>Description</b>	This message only applies to monochrome raster images. All other image classes return <b>PAN_CTLERRNOTCOMPATIBLE</b> .
<b>Parameters</b>	<p><i>wParam</i>:       <b>(WORD)</b> input, one of:                    <b>PAN_IMAGE_CONTRAST</b>,                    <b>PAN_IMAGE_ANTIALIAS</b>,                    <b>PAN_IMAGE_INVERT</b> or                    <b>PAN_IMAGE_BRIGHTNESS</b>.</p> <p><i>lParam</i>: <b>(wParam = PAN_IMAGE_CONTRAST)</b>            <b>(int)</b>     input, contrast setting -100 (fully saturated) to 100 (least saturated), default:0 (unchanged).            <b>(wParam = PAN_IMAGE_ANTIALIAS)</b>            <b>(BOOL)</b> input, anti-aliasing, <b>TRUE</b>                    (on) or <b>FALSE</b> (off), default: off.            <b>(wParam = PAN_IMAGE_INVERT)</b>            <b>(BOOL)</b> input, color inversion, <b>TRUE</b>                    (on) or <b>FALSE</b> (off), default: off.            <b>(wParam = PAN_IMAGE_BRIGHTNESS)</b>            <b>(int)</b>     input, brightness setting -100 (darkest (black)) to 100 (lightest (white)), default:0 (unchanged).</p>
<b>Returns</b>	error code
<b>Compatibility</b>	raster and vector-2D controls
<b>Restrictions</b>	<b>PAN_IMAGE_INVERT</b> is only used for monochrome raster images and <b>PAN_IMAGE_BRIGHTNESS</b> 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

<b>Purpose</b>	Returns the buffer of layer states set by the decoder.
<b>Description</b>	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.
<b>Parameters</b>	<p><b>MultiByte:</b></p> <p><i>wParam:</i> <b>(LOWORD)</b> number of layer structures to retrieve  <b>(HIWORD)</b> viewport to obtain the layer information from  <b>(PAN_LAYER *)</b> layers</p> <p><i>lParam:</i> <b>(PAN_LAYER *)</b> layers</p> <p><b>Unicode:</b></p> <p><i>wParam:</i> <b>(LOWORD)</b> must be -1  <b>(HIWORD)</b> viewport to obtain the layer information from  <b>(std:: <sup>(1)</sup>vector&lt;CPanLayer&gt; *)</b> layers</p> <p><i>lParam:</i> <b>(std:: <sup>(1)</sup>vector&lt;CPanLayer&gt; *)</b> layers</p>
<b>Returns:</b>	Number of layers copied to <i>lParam</i> , current layer count if <i>lParam</i> is NULL, 0 if no layers, -1 on failure.
<b>Compatibility:</b>	vector
<b>Restrictions:</b>	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;
    }
}
```

<sup>(1)</sup> Look at Overview section about usage of STL and Boost libraries

```
    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**

<b>Purpose</b>	Set the parameters of the layer state buffer.
<b>Description</b>	Sets the internal buffer of layers in the control. Any following <b>PANX_GetLayer</b> call references this buffer.
<b>Parameters</b>	<p><b>MultiByte:</b></p> <p style="margin-left: 40px;"><i>wParam:</i>       <b>(LOWORD)</b> number of layers to set  <b>(HIWORD)</b> viewport to set the layers on.  If less than zero, set these layers on all viewports.</p> <p style="margin-left: 40px;"><i>lParam:</i>       <b>(PAN_LAYER *)</b> layer information.  If NULL, the default layers are restored for the specified viewport.</p> <p><b>Unicode:</b></p> <p style="margin-left: 40px;"><i>wParam:</i>       <b>(LOWORD)</b> must be -1  <b>(HIWORD)</b> viewport to set the layers on.  If less than zero, set these layers on all viewports.</p> <p style="margin-left: 40px;"><i>lParam:</i>       <b>(std::<sup>(1)</sup>vector&lt;CPanLayer&gt; *)</b> layer information.  If NULL, the default layers are restored for the specified viewport.</p>
<b>Returns:</b>	error code
<b>Compatibility:</b>	vector
<b>Restrictions:</b>	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.
<b>Special Notes:</b>	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.

```

---

<sup>(1)</sup> Look at Overview section about usage of STL and Boost libraries

```
    SendMessage(ctrlHandle, PM_CTLSETLAYERSTATE, numLayers,
        (LPARAM) layers);

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

## XREF-Related Command Messages

### PM\_CTLGETXREFSTATE

<b>Purpose</b>	Returns the buffer of XRef states set by the decoder.
<b>Description</b>	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.
<b>Parameters</b>	<p><i>wParam</i>:           &lt; 0       (MultiByte) Negative of number of XRef structures to retrieve.</p> <p>                      1        (Unicode) Flag indicating the usage of the Unicode interface.</p> <p><i>lParam</i>:           <b>(wParam: &lt; 0)</b> (PAN_XREF *) lpXRefs                        <b>(wParam: 1)</b>   (std::vector&lt;CPanXRef&gt; *) pvXRefs</p>
<b>Returns:</b>	Number of XRefs or CPanXRefs copied to <i>lParam</i> , XRef count if <i>lParam</i> == NULL, 0 if no XRefs, -1 on failure.
<b>Compatibility:</b>	vector-2D
<b>Restrictions</b>	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"));
    }
}
```

<sup>(1)</sup> Look at Overview section about usage of STL and Boost libraries

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

    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"));
    }
}
```

---

<sup>(1)</sup> Look at Overview section about usage of STL and Boost libraries

**PM\_CTLSETXREFSTATE**

<b>Purpose</b>	Set the parameters of the XRef state buffer.		
<b>Description</b>	Sets the internal buffer of XRefs in the control. Any following <b>PANX_GetXRefs</b> call references this buffer.		
<b>Parameters</b>	<i>wParam:</i>	<b>(WORD) &lt; 0</b> <b>(WORD) 1</b>	(MultiByte) Negative of number of XRef structures. (Unicode) Indicating Unicode version.
	<i>lParam:</i>	<b>(wParam: &lt; 0)</b> <b>(wParam: 1)</b>	<b>(PAN_XREF *)</b> lpXRefs <b>(std::<sup>(1)</sup>vector&lt;CPanXRef&gt; *)</b> pvXRefs
<b>Returns:</b>	error code		
<b>Compatibility:</b>	vector-2D		
<b>Restrictions</b>	If <i>wParam</i> is larger than the number of existing XRefs, then only the number of existing XRefs is set.		
<b>Special Notes</b>	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)
```

<sup>(1)</sup> 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));
}
```

---

<sup>(1)</sup> Look at Overview section about usage of STL and Boost libraries

## Bookmark-Related Command Messages

### PM\_CTLGETBOOKMARKS

<b>Purpose</b>	Returns the buffer of PAN_BOOKMARK struct set by the decoder.
<b>Description</b>	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.
<b>Parameters</b>	<p><i>wParam</i>:           <b>(WORD) &lt; 0</b> Negative of the number of Bookmark structures to retrieve.</p> <p>                     <b>(WORD) 1</b>   Flag indicating the usage of the Unicode interface.</p> <p><i>lParam</i>:           <b>(wParam: &lt; 0)</b> <b>(PAN_BOOKMARK *)</b> lpBMarks</p> <p>                     <b>(wParam: 1)</b>   <b>(std::<sup>(1)</sup>vector&lt;CPanBookmark&gt; *)</b> pvBookmarks</p>
<b>Returns:</b>	Number of Bookmarks copied to <i>lParam</i> , Bookmark count if <i>lParam</i> == NULL, 0 if no Bookmarks, -1 on failure.
<b>Compatibility:</b>	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(ctrlHandle, 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
```

<sup>(1)</sup> 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);
}
```

---

<sup>(1)</sup> Look at Overview section about usage of STL and Boost libraries



## Resource-Related Command Messages

**PM\_CTLGETRESOURCEINFOSTATE**

<b>Purpose</b>	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.
<b>Description</b>	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.
<b>Parameters</b>	<p><i>wParam</i>: <math>\geq 0</math>                      number of resource info structures to retrieve.  <math>-1</math>                                      the resource iterator is requested.</p> <p><i>lParam</i>: <b>(wParam <math>\geq 0</math>)</b> (PAN_RESOURCEINFO *) lpResourceInfo  <b>(wParam = -1)</b>  <b>(boost::<sup>(1)</sup>shared_ptr&lt;IGetResourceInfo&gt; *)</b> resource iterator</p>
<b>Returns:</b>	Number of entries copied to <i>lParam</i> , resource count if <i>lParam</i> == NULL, 0 if no resources, -1 on failure.
<b>Compatibility:</b>	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;
    }
}

```

<sup>(1)</sup> Look at Overview section about usage of STL and Boost libraries

## Offset-Related Command Messages

### PM\_CTLGETOFFSET

<b>Purpose</b>	Get the origin offset.
<b>Description</b>	Returns the offset of the origin for the current file in world coordinates.
<b>Parameters</b>	<i>wParam:</i> not used <i>lParam:</i> ( <b>PAN_CtlPos *</b> ) origin offset
<b>Returns:</b>	error code
<b>Compatibility:</b>	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**

<b>Purpose</b>	Set the origin offset.
<b>Description</b>	Sets the offset of the origin for the current file in world coordinates.
<b>Parameters</b>	<i>wParam</i> : not used <i>lParam</i> : (PAN_CtlPos *) origin offset
<b>Returns:</b>	error code
<b>Compatibility:</b>	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

<b>Purpose</b>	Returns the base font of the current file.
<b>Parameters</b>	<i>wParam:</i> not used  <i>lParam:</i> <b>(LPLOGFONT)</b> base font
<b>Returns:</b>	error code
<b>Compatibility:</b>	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**

<b>Purpose</b>	Sets the base font of the current file.
<b>Parameters</b>	<i>wParam</i> : not used <i>lParam</i> : <b>(const LOGFONT *)</b> base font
<b>Returns:</b>	error code
<b>Compatibility:</b>	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 = ctrlHandle;
    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**

<b>Purpose</b>	Return the width of the specified field/column.
<b>Description</b>	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.
<b>Parameters</b>	<p><i>wParam</i>            <b>(WORD)</b> unit type to use.</p> <p><i>lParam</i>            <b>(DWORD)</b> field/column</p>
<b>Returns:</b>	<b>(LRESULT)</b> width on success or -1 on failure
<b>Compatibility:</b>	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**

<b>Purpose</b>	Returns the height of the given record/row in specified units.
<b>Description</b>	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.
<b>Parameters</b>	<i>wParam:</i> <b>(WORD)</b> unit type to use  <i>lParam:</i> <b>(DWORD)</b> record/row
<b>Returns:</b>	<b>(LONG)</b> height on success or -1 on failure
<b>Compatibility:</b>	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

<b>Purpose</b>	Returns the number of pages in the current file.
<b>Description</b>	Return the number of pages in the current file in the <b>int</b> pointed to by <i>lParam</i> .
<b>Parameters</b>	<p><i>wParam</i>: not used</p> <p><i>lParam</i>: <b>(LPINT)</b> 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).</p> <p>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 <b>PM_CTLGETNUMPAGES</b> message will return a positive number.</p>
<b>Returns:</b>	error code
<b>Compatibility:</b>	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**

<b>Purpose</b>	Returns the current page number within the current file	
<b>Description</b>	Return the current page number in the current file in the <b>WORD</b> pointed to by <i>lParam</i> .	
<b>Parameters</b>	<i>wParam</i>	not used
	<i>lParam</i> :	<b>(LPWORD)</b> current page number
<b>Returns:</b>	error code	
<b>Compatibility:</b>	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);
    }
}
```

<b>PM_CTLGETPAGESIZE</b>
--------------------------

<b>Purpose</b>	Returns the size of the specified page in the current file.
<b>Description</b>	<p>Return the maximum extents for the current page of the current file. The <b>PAN_CtlRange</b> structure returns the extents as a pair of points that bound the page.</p> <p>For vector, raster, archive, database and spreadsheet controls, the returned value always represents the maximum extents of the current page.</p>
<b>Parameters</b>	<p><i>wParam:</i>           <b>(WORD)</b> page number for document control. Not used by other controls.</p> <p><i>lParam:</i>           <b>(PAN_CtlRange *)</b> page size</p>
<b>Returns:</b>	error code
<b>Compatibility:</b>	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**

<b>Purpose</b>	Sets the current page number within the current file.
<b>Description</b>	Set the current page number for the current file based on the value specified by the <b>WORD</b> passed as the <i>wParam</i> .
<b>Parameters</b>	<i>wParam</i> : <b>(WORD)</b> current page number  <i>lParam</i> :           not used
<b>Returns:</b>	error code
<b>Compatibility:</b>	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

<b>Purpose</b>	Prints the specified extents.
<b>Description</b>	Print the specified extents using the options specified in the <b>PAN_CtlPrintOptions</b> structure pointed to by <i>lParam</i> .
<b>Parameters</b>	<p><i>wParam</i>: Page number specifying the page to print, <b>(WORD)-1</b> to print all pages</p> <p><i>lParam</i>: <b>(const PAN_CtlPrintOptions *)</b> printing options</p>
<b>Returns:</b>	error code
<b>Compatibility:</b>	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**

<b>Purpose</b>	Returns the print preview of the specified extents.
<b>Description</b>	Return the printing preview of the specified extents using the options specified in the <b>PAN_CtlPrintOptions</b> structure pointed to by <i>lParam</i> . All member variables and structures (except <b>printPreview</b> ) of the given <b>PAN_CtlPrintOptions</b> structure must be properly initialized.
<b>Parameters</b>	<i>wParam</i> :           not used  <i>lParam</i> : <b>(PAN_CtlPrintOptions *)</b> preview information
<b>Returns:</b>	error code
<b>Compatibility:</b>	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;
}

```

<b>PM_CTLVALIDATEMARGINS</b>
------------------------------

<b>Purpose</b>	Validates the printer margins.
<b>Description</b>	This message is used to allow the control to ensure that the printer margins in the given <b>PAN_CtlPrintOptions</b> 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.
<b>Parameters</b>	<i>wParam</i> : not used  <i>lParam</i> : ( <b>PAN_CtlPrintOptions *</b> ) print options
<b>Returns:</b>	error code
<b>Compatibility:</b>	all control types
<b>Restrictions:</b>	<p>This message requires that the <b>hDC</b> member of the <b>printDlg</b> structure contained in the <b>PAN_CtlPrintOptions</b> structure be a valid printer device context handle.</p> <p>The margins specified in the print options will not be adjusted if the <b>mode</b> member of the <b>PAN_CtlPrintOptions</b> structure has the <b>PAN_CTLMODEIGNOREMINMARGINS</b> flag set.</p>

**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

<b>Purpose</b>	Returns the number of retrievable strings from a control, or a specified string itself.
<b>Description</b>	PM_CTLGETSTRING allows an application to retrieve strings that were decoded from the displayed file.
<b>Parameters</b>	<p><i>wParam</i>:        -1: To return the total number of strings.                         -2: Use the Unicode interface.                         0..numstrings-1: To return the specified string.</p> <p><i>lParam</i>:        if wParam == -1 then (LPDWORD) nStrings.                         if wParam == -2 then (boost::<sup>(1)</sup>shared_ptr&lt;IGetString&gt; *) pspGetString                         if wParam &gt; 0 then (LPSTR) lpBuffer.</p>
<b>Returns:</b>	error code
<b>Compatibility:</b>	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);
    }
}

```

<sup>(1)</sup> 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 occurrences 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**

<i>wParam</i> :	<b>0</b>	MultiByte text search
	<b>≠ 0</b>	Unicode text search
<i>lParam</i> :	<b>(wParam: 0)</b>	<b>(PAN_CtlSearchInfo *)</b> MultiByte search information.
	<b>(wParam: ≠ 0)</b>	<b>(CPanTextSearch *)</b> 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 occurrence.
    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, &vContours)) {  
        ...  
    }  
}
```

---

<sup>(1)</sup> Look at Overview section about usage of STL and Boost libraries

## Mouse Command Messages

### PM\_CTLGETLMBACTION

<b>Purpose</b>	Returns the left/right mouse button control window behavior.		
<b>Description</b>	Return, in the <b>WORD</b> pointed to by <i>lParam</i> , 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.		
	<b>PAN_CTLLMBNONE:</b>	does nothing	
	<b>PAN_CTLLMBSELECT:</b>	selects a portion of the current file	
	<b>PAN_CTLLMBZOOM:</b>	zooms in on a portion of the current file	
<b>Parameters</b>	<i>wParam</i>	0 => LMB action 1 => RMB action	
	<i>lParam:</i>	<b>(LPARAM)</b> LMB action	
<b>Returns:</b>	error code		
<b>Compatibility:</b>	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:

<b>PAN_CTLLMBNONE</b>	does nothing
<b>PAN_CTLLMBSELECT</b>	selects a portion of the current file
<b>PAN_CTLLMBZOOM</b>	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**

<b>Purpose</b>	Renders the specified extents onto the given device context.
<b>Description</b>	This message may be used to implement printing or for combining the contents of a control with graphical elements from other sources.
<b>Parameters</b>	<p><i>wParam</i>: Not used</p> <p><i>lParam</i>: (<b>PAN_CtlRenderOptions *</b>) options</p>
<b>Returns:</b>	error code
<b>Compatibility:</b>	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

<b>Purpose</b>	Sets the formatting device for document files.
<b>Description</b>	Sets the device used by the document control for text formatting. Fonts not supported by the device are adapted. <i>wParam</i> specifies the type of device: screen, printer with description given in <i>lParam</i> , or the standard device specified in WIN.INI.
<b>Parameters</b>	<div><div><i>wParam</i>:</div><div>Device type which can be one of the following: PAN_CTLDEVICESTANDARD, PAN_CTLDEVICEPRINTER, PAN_CTLDEVICESCREEN.</div></div> <div><div><i>lParam</i>:</div><div>if <i>wParam</i> == PAN_CTLDEVICEPRINTER, <i>lParam</i> must point to a string containing the device description specified in the same format used in WIN.INI: "Name,Driver,Port". Otherwise, <i>lParam</i> must be set to NULL.</div></div>
<b>Returns:</b>	error code
<b>Compatibility:</b>	Document control.

**PM\_CTLGETDEVICE**

<b>Purpose</b>	Gets the formatting device for document files.
<b>Description</b>	Returns the device used by the document control for text formatting.
<b>Parameters</b>	<div><div><i>wParam:</i></div><div>Size of buffer passed in lParam to hold device description if current device type is PAN_CTLDEVICEPRINTER.</div></div> <div><div><i>lParam:</i></div><div>Buffer to hold device description if current device type is PAN_CTLDEVICEPRINTER. If NULL description is not returned</div></div>
<b>Returns:</b>	Type of current device: PAN_CTLDEVICESTANDARD, PAN_CTLDEVICEPRINTER, or PAN_CTLDEVICESTANDARD.
<b>Compatibility:</b>	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
<b>PNM_CTLDESTROY</b>	A control sends this message prior to its destruction.
<b>PNM_CTLDROPFILE</b>	Sent when a file has been dropped on the control window.
<b>PNM_CTLHELPSTRING</b>	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.
<b>PNM_CTLSIZE</b>	Sent when the control window has changed size.
<b>PNM_CTLSTATUS</b>	Sent when a controls' status changes.

### Image-Manipulation Related

<b>PNM_CTLPAINT</b>	Sent when the control window has been repainted.
<b>PNM_CTLREGEN</b>	Sent when the control has been regenerated.
<b>PNM_CTLSETFOCUS</b>	Sent when a control obtains the focus.
<b>PNM_CTLHSCROLL</b>	Sent when the control window has been horizontally scrolled
<b>PNM_CTLVSCROLL</b>	Sent when the control window has been vertically scrolled.

### Printing Notification Messages

<b>PNM_CTLPRINT</b>	A control sends this message when it has completed a print job initiated with PM_CTLPRINT.
<b>PNM_CTLPRINTINGPAGE</b>	A control sends this message when printing a given page during a print job.
<b>PNM_CTLPRINTPROCESSINGPAGE</b>	A control sends this message when processing (reading) a given page during a print job.

### Clipboard-Related Message

<b>PNM_CTLSETSEL</b>	A control sends this message whenever the selection has changed.
<b>PNM_CTLRENDERSEL</b>	Sent while selection is being copied to the clipboard to allow the application to append its own data.

## Archive/Database/Spreadsheet specific

<b>PNM_CTLARCFIELD</b>	An archive control sends this message when an entry has been double-clicked in the control window.
<b>*PNM_CTLCOLWIDTH</b>	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.
<b>*PNM_CTLROWHEIGHT</b>	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

<b>PNM_CTLOBJECT</b>	Sent when an OLE object marker has been double clicked.
----------------------	---

## Link Notification Message

<b>PNM_CTLLINK</b>	Sent when the mouse is moved on top of a hot link or when a hot link is double clicked
--------------------	--

## Views Notification Message

<b>PNM_CTLSETVIEWEXTENTS</b>	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**

<b>Purpose</b>	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.	
<b>Parameters</b>	<i>wParam:</i>	not used
	<i>lParam:</i>	(LPCSTR) help string

**PNM\_CTLSIZE**

**Purpose** A control sends this message when the control window has changed size.

**Parameters**

<i>wParam:</i>	not used
<i>lParam:</i>	<b>(const RECT *)</b> new size

**PNM\_CTLSTATUS**

**Purpose** A control sends this message whenever its status changes.

**Parameters**

<i>wParam:</i>	not used
<i>lParam:</i>	( <b>DWORD</b> ) status (see <b>PM_CTLGETSTATUS</b> )

## **Image Notification Messages**

### **PNM\_CTLPAINT**

**Purpose** A control sends this message whenever the control window has been repainted.

**Parameters**

<i>wParam:</i>	not used
<i>lParam:</i>	( <b>const RECT *</b> ) 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**

<i>wParam:</i>	not used
<i>lParam:</i>	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**

<i>wParam:</i>	not used
<i>lParam:</i>	( <b>const PAN_CtlSel *</b> ) new selection

**PNM\_CTLRENDERSEL**

<b>Purpose</b>	A control sends this message while the selection is being copied to the clipboard to allow the application to append its own data.
<b>Parameters</b>	<i>wParam:</i> not used  <i>lParam:</i> ( <b>PAN_CtlRenderOptions *</b> ) options
<b>Compatibility</b>	Raster and Vector controls.



**Archive-, Database- and Spreadsheet-Specific Notification Messages****PNM\_CTLARCFILE**

<b>Purpose</b>	An archive control sends this message when an entry has been double-clicked in the control window.		
<b>Parameters</b>	<i>wParam:</i>	not used	
	<i>lParam:</i>	<b>(LPCSTR)</b> name of temporary file in which of extracted file are placed followed by name of extracted file at offset <b>PAN_MAX_PATH</b>	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**

<i>wParam:</i>	not used
<i>lParam:</i>	(PAN_CtlObject *) object descriptor.

## 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 descriptor.

**Views-Related Notification Message****PNM\_CTLSETVIEWEXTENTS**

**Purpose** A control sends this message when the view extents have changed.

**Parameters**

<i>wParam</i>	not used
<i>lParam</i>	(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**=*paths*

*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**=*paths*

*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

**DGNCOLOR\_TBL=*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

## —A—

anti-aliasing, 56, 126, 128

## —B—

Background, 55, 62, 110, 114  
Base font, 56, 140, 141  
Bookmark, 57

## —C—

Capabilities, 15, 54, 60, 72  
Child window, 22, 58  
Client area coordinates, 55, 98, 102  
Clipboard, 13, 15, 39, 55, 60, 72, 104, 105  
Color, 36, 55, 62, 110, 111, 114, 115  
Command Message Descriptions, 58  
Command Message Summary, 54, 57  
Command Messages, 54  
contrast, 56, 126, 128  
Control ID, 22  
Control window rectangle, 22  
Convert, 8, 15, 54, 87  
Copy, 13, 15, 55, 60, 72, 104  
**ctlConsumeProc**, 22  
**ctlNotifyProc**, 22

## —D—

DestroyWindow, 20, 84  
Device context, 15, 27, 57, 151, 158, 176  
Duplicate, 54, 58

## —E—

Error codes, 25

## —F—

File, 10, 15, 26, 27, 28, 29, 36, 37, 39, 41, 42, 54, 55, 56, 57, 63, 64, 65, 70, 74, 75, 97, 98, 101, 102, 107, 108, 109, 111, 115, 117, 119, 120, 126, 138, 139, 140, 141, 145, 146, 147, 148, 154, 156, 157, 161, 164, 165, 177  
Flip, 89, 90, 91  
Foreground, 55, 62, 110, 114  
Formats, 15, 39, 54, 55, 64, 74, 104, 105  
Function summary, 19

## —H—

Height, 40, 57, 143

## —I—

Index, 188  
Information, 9, 15, 20, 22, 42, 54, 55, 63, 66, 111, 150, 154, 173, 174  
initialization files, 183  
Introduction, 8, 9

## —K—

Keyboard, 15, 22, 60, 72

## —M—

Margins, 40, 41, 57, 151  
Metafile, 15  
Mode, 26, 54, 67, 79  
monochrome, 126, 128  
Mouse, 15, 22, 57, 60, 72, 156, 157

## —N—

Notification Messages, 161

## —O—

Offset, 41, 56, 138, 139, 173, 174, 177  
Overview, 15

## —P—

Page, 36, 40, 41, 57, 60, 72, 142, 143, 145, 146, 147, 148, 161, 173, 174  
Paint, 54, 93  
Palette, 27, 55, 111, 112, 113, 115  
**PAN\_CreateControl()**, 20, 21, 22, 24  
**PAN\_CTLMODEINTERRUPTIBLE**, 81  
**PAN\_FreeControls()**, 20  
**PAN\_IMAGE\_ANTIALIAS**, 126, 128  
**PAN\_IMAGE\_CONTRAST**, 126, 128  
**PAN\_LoadControls()**, 20, 21, 183  
Path Settings, 184  
**PM\_CTLCARETTOWORLD**, 96  
**PM\_CTLCLEARSELS**, 97  
**PM\_CTLCLIENTTOWORLD**, 98  
**PM\_CTLCLONECONTROL**, 58  
**PM\_CTLCONVERT**, 87  
**PM\_CTLCOPY**, 104  
**PM\_CTLDESTROY**, 20, 59, 83, 84  
**PM\_CTLFLIP**, 89, 90, 91  
**PM\_CTLGETBASEFONT**, 140  
**PM\_CTLGETBLOCK**, 121  
**PM\_CTLGETBLOCKNAMES**, 122  
**PM\_CTLGETBOOKMARKS**, 135  
**PM\_CTLGETCAPS**, 60

PM\_CTLGETCARETPOS, 99  
 PM\_CTLGETCLPBRDFMTS, 105  
 PM\_CTLGETCOLWIDTH, 142  
 PM\_CTLGETDEVICE, 160  
 PM\_CTLGETDIMS, 62  
 PM\_CTLGETENTITY, 124  
 PM\_CTLGETFGBGCOLOR, 110  
 PM\_CTLGETFILE, 63  
 PM\_CTLGETFILEFMTS, 64  
 PM\_CTLGETFILETYPE, 65  
**PM\_CTLGETIMAGEEX**, 56, 126  
 PM\_CTLGETINFO, 66  
 PM\_CTLGETLAYERSTATE, 129  
 PM\_CTLGETLMBACTION, 156  
 PM\_CTLGETMODE, 67  
 PM\_CTLGETNUMPAGES, 145  
 PM\_CTLGETNUMSELS, 106  
 PM\_CTLGETOFFSET, 138  
 PM\_CTLGETOPTION, 69  
 PM\_CTLGETPAGE, 146  
 PM\_CTLGETPAGESIZE, 147  
 PM\_CTLGETPALETTE, 111  
 PM\_CTLGETRESOURCEINFSTATE, 137  
 PM\_CTLGETROWHEIGHT, 143  
 PM\_CTLGETSELS, 107  
 PM\_CTLGETSTATUS, 70  
 PM\_CTLGETSTRING, 152  
 PM\_CTLGETVIEW, 116  
 PM\_CTLGETVIEWEXTENTS, 117  
 PM\_CTLGETVIEWNAMES, 118  
 PM\_CTLGETXREFSTATE, 131  
 PM\_CTLGETZOOM, 92  
 PM\_CTLHSCROLL, 93  
 PM\_CTLPAINT, 93  
 PM\_CTLPALETTECHANGED, 112  
 PM\_CTLPRINT, 149  
 PM\_CTLPRINTPREVIEW, 150  
 PM\_CTLQUERYNEWPALETTE, 113  
 PM\_CTLREGEN, 71  
 PM\_CTLRENDERONTODC, 158  
 PM\_CTLROTATE, 94  
 PM\_CTLSEARCH, 154  
 PM\_CTLSETBASEFONT, 141  
 PM\_CTLSETBLOCK, 123  
 PM\_CTLSETCAPS, 72  
 PM\_CTLSETCARETPOS. *See*  
 PM\_CTLSETDEVICE, 159  
 PM\_CTLSETFGBGCOLOR, 114  
 PM\_CTLSETFILE, 74, 75  
**PM\_CTLSETIMAGEEX**, 56, 128  
 PM\_CTLSETLAYERSTATE, 130  
 PM\_CTLSETLMBACTION, 157  
 PM\_CTLSETMODE, 79  
 PM\_CTLSETOFFSET, 139  
 PM\_CTLSETOPTION, 85  
 PM\_CTLSETPAGE, 148  
 PM\_CTLSETPALETTE, 115  
 PM\_CTLSETSEL, 108  
 PM\_CTLSETSELCARET, 109  
 PM\_CTLSETVIEW, 119  
 PM\_CTLSETVIEWEXTENTS, 120  
 PM\_CTLSETXREFSTATE, 133

PM\_CTLSETZOOM, 95  
 PM\_CTLSHOWENTITY, 125  
 PM\_CTLSIZE, 93  
 PM\_CTLSORT, 144  
 PM\_CTLVALIDATEMARGINS, 151  
 PM\_CTLVSCROLL, 93  
 PM\_CTLWORLDTOCARET, 101  
 PM\_CTLWORLDTOCLIENT, 102  
 PM\_CTLXFRMRECT, 103  
 PNM\_CTLARCFILE, 177  
 PNM\_CTLCOLWIDTH, 178  
 PNM\_CTLDESTROY, 163  
 PNM\_CTLDROPFILE, 164  
 PNM\_CTLHELPSTRING, 165  
 PNM\_CTLHSCROLL, 171  
 PNM\_CTLLINK, 181  
 PNM\_CTLOBJECT, 180  
 PNM\_CTLPAINT, 168  
 PNM\_CTLPRINT, 172  
 PNM\_CTLPRINTINGPAGE, 173  
 PNM\_CTLPRINTPROCESSINGPAGE, 174  
 PNM\_CTLREGEN, 169  
 PNM\_CTLRENDERSEL, 176  
 PNM\_CTLROWHEIGHT, 179  
 PNM\_CTLSETFOCUS, 170  
 PNM\_CTLSETSEL, 175  
 PNM\_CTLSETVIEWEXTENTS, 182  
 PNM\_CTLSIZE, 166  
 PNM\_CTLSTATUS, 167  
 PNM\_CTLVSCROLL, 171  
 Print, 15, 57, 149, 150, 151, 161, 172, 173, 174

## —R—

Raster Viewing Options, 184  
 Render, 15, 54, 57, 74, 75, 158  
 Rotate, 54, 94

## —S—

Scroll, 15, 22, 60, 72, 93, 171  
 Search, 10, 37, 57, 60, 72, 154  
 Selections, 40, 43, 44, 55, 57, 96, 97, 101, 104, 106, 107,  
 149, 150, 175  
**ShowWindow**, 20  
 Stand-alone window, 22, 58  
 Status, 54, 70, 161, 167  
 struct PAN\_CtlClpbrdFmt, 39  
 struct PAN\_CtlClpbrdFmtList, 39  
 struct PAN\_CtlFileFmt, 39  
 struct PAN\_CtlFileFmtList, 39  
 struct PAN\_CtlFileInfo, 36  
 struct PAN\_CtlInfo, 35  
 struct PAN\_CtlObject, 44, 45  
 struct PAN\_CtlPos, 35  
 struct PAN\_CtlPrintOptions, 41  
 struct PAN\_CtlPrintPreview, 40  
 struct PAN\_CtlRange, 35  
 struct PAN\_CtlRenderOptions, 43  
 struct PAN\_CtlSearchInfo, 37  
 struct PAN\_CtlSel, 40

struct PAN\_CtlSelList, 40  
Structure Descriptions, 33  
Structure Summary, 31

—T—

Text search, 10

—V—

Validate, 151  
Vector Viewing Options, 183  
View extents, 55, 117, 120, 162, 182

—W—

Width, 57, 120, 142  
World coordinates, 15, 55, 56, 98, 102, 117, 120, 138, 139

—Z—

Zoom, 11, 12, 54, 60, 72, 92, 95, 120

# 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](mailto: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](mailto:autovuesales_ww@oracle.com)

## Customer Support:

Web Site: <http://www.oracle.com/support/index.html>

