Markup Control API Manual

Markup Enabling Technology

Contents

INTRODUCTION **5**

API FOR USER DEFINED ENTITY DLLS **6**

OVERVIEW **6**
ENTRY FUNCTIONS: **6**
GENERAL INFORMATION: 6
CONSTRUCTORS/DESTRUCTORS: 6
I/O FUNCTIONS: 6
EVENT FUNCTIONS: 7
RENDER FUNCTIONS: 7
GEOMETRY/CONTENTS FUNCTIONS: 7

MARKUP CONTROL API **9**

OVERVIEW **9**
DLL LOADING/UNLOADING: 9
CONTROL CREATION: 9
COMMAND MESSAGES: 13
MARKUP OBJECT MESSAGES: 13
PALETTE MESSAGES: 15
LAYER MESSAGES: 15
PAGE INFORMATION MESSAGES: 16
VIEW MESSAGES: 16
COORDINATE MESSAGES: 17
ENTITY MANIPULATION MESSAGES: 18
PROPERTY MESSAGES: 20
ACTION MESSAGES: 22
SELECTION MESSAGES: 24
CLIPBOARD MESSAGES: 25
I/O MESSAGES: 25
UDE MESSAGES: 26
RENDER MESSAGES: 26
NOTIFICATION MESSAGES: **27**
COMMON DEFINITIONS: **29**
ENTITY TYPES: 29
ACTION CODES: 30
PROPERTY CODES: 30
TEXT ALIGNMENT CODES: 31
PENSTYLE CODES: 31
FILL CODES: 31

ENTITY STATUS CODES: 31
ENTITY GET/ENUM CODES: 32
MOUSE NOTIFICATION CODES: 32
FLIP CODES: 34
HYPERLINK TYPES: 34
DATA STRUCTURES: 34

Introduction

The goals of implementing the markup control are:

- Implementing all the capabilities embedded in the markup control of AutoVue. These capabilities should be available to the licenser of the markup.
- Providing the markup capability in DLL form.
- Adding support for user defined entities (UDEs).
- Keeping the API highly compatible with current API to facilitate its integration into a future release of AutoVue Pro.

The markup implementation in AutoVue is available as a code base, and will be used to fulfill these goals. Note that the final product is intended to be independent of both VCET and AutoVue. It will also be independent of the toolkit that customers will be using.

API for User Defined Entity DLLs

Overview

A new feature of the markup control will be its provision for supporting new markup entity types that may be implemented by application developers. These new entities will be implemented in User Defined Entity DLLs (UDEs).

The UDE DLL provides the user interface for entity creation, I/O functionality (read/write contents from/to a memory buffer), support for entity copying (cloning), and drawing routines. Additional functionality includes returning Bounding Box info, performing hit tests, and editing geometry and contents as requested by the markup control. The markup control, on the other hand, provides the memory space for storing UDE entities, and manages the selection.

Users will construct new entities by interacting with the UDE DLL. During entity creation the mouse messages are forwarded to the UDE through the mouse event function. The return value of this function is used to inform the markup control when an entity is completed. The markup control then adds the new UDE entity to its internal entity list.

Entry Functions:

LibMain()

Standard DLL initialization function. Provides implementers with an opportunity to handle any one-time initialization.

WEP()

Standard DLL termination function. Allows implementer to free any resources that have not yet been released.

General information:

void PCALLBACK WhoIAm(LPUSEINFO Info);

Enables the UDE to describe itself by filling in the appropriate fields in Info, except for entID

Constructors/Destructors:

void PCALLBACK InitEntity(LPMRK_ENTITYSPEC Entity);
void PCALLBACK ReleaseEntity(LPMRK_ENTITYSPEC Entity);

Perform memory management of the UDE internal data structure, using MrkAlloc/MrkFree, as well as any necessary initializations.

I/O functions:

long PCALLBACK ReadEntity(LPMRK_ENTITYSPEC EntitySpec);

Converts entity data read from disk into an internal memory representation of the entity. The internal format is written into a buffer created using MrkAlloc(). The buffer size and pointer are returned inside MRK_UdeInfo part of pEntitySpec. The function always returns the number of bytes read from the input buffer.

**long PCALLBACK WriteEntity(LPMRK_ENTITYSPEC EntitySpec,
BOOL fSizeOnly);**

If fSizeOnly flag is not set, the function converts the internal memory representation of the entity into an output format suitable for writing to a file. The output format is loaded into a memory buffer created using MrkAlloc. The buffer size and pointer are returned in MRK_UdeInfo part of pEntitySpec. The function always returns the byte count of the output format.

Event functions:

**int PCALLBACK MouseProc(int MouseMsg, WPARAM wParam,
LPARAM lParam, LPMRK_ENTITYSPEC pEntitySpec);**

Used to forward mouse messages to the UDE DLL during entity creation. Function arguments are as follows:

MouseMsg:	Mouse messages which can have one of the following values: MRK_LBUTTONDOWN, MRK_LBUTTONUP, MRK_LBUTTONDBLCLK, MRK_RBUTTONDOWN, MRK_RBUTTONUP, MRK_MOUSEMOVE, MRK_LEFTMOUSEDOWN.
wParam:	Same as wParam of corresponding Windows mouse message.
lParam:	Pointer to PAN_Point giving the mouse position in world coords.
pEntitySpec:	Pointer to entity spec structure containing the UDE entity info.

The markup control interprets the return value as follows:

-1	Entity is completed.
0	Abort entity creation.
1	Continue.

Render functions:

**void PCALLBACK Draw(LPMRK_RenderOptions pRenderOpts,
LPMRK_ENTITYSPEC pEntitySpec);**

This function will instantiate an instance of a UDE on the user's display. It needs to be hardened to handle the following needs:

- Display the UDE entity even when partly defined (optional),
- Display the UDE entity in XOR mode when specified by pRenderOpts (e.g. when UDE entity is selected).

Geometry/Contents functions:

BOOL PCALLBACK BoundingBox (LPMRK_ENTITYSPEC pEntitySpec);

The entity bounding box is returned as a PAN_CtlPos[2] in the data pointer of MRK_UdeInfo part of pEntitySpec. The pointer must be allocated using MrkAlloc.

**BOOL PCALLBACK SelectionTest(PAN_CtlRange*pselRange,
LPMRK_ENTITYSPEC pEntitySpec);**

Returns TRUE if the given range covers the entity, and the entity should be selected as a result.

**BOOL PCALLBACK GetControlPoints(int FAR *pNumPts,
LPPANPOINT FAR *pPts, LPMRK_ENTITYSPEC pEntitySpec);**

Retrieves the control points used to place the edit handles. The point buffer must be allocated using MrkAlloc() (markup control will take care of freeing the buffer). If it returns FALSE, the positioning of the edit handles is determined by the entity's bounding box.

```
BOOL   PCALLBACK   Translate(PAN_Point *pShift,
                             LPMRK_ENTITYSPEC pEntitySpec);
```

Shifts the coordinates of the UDE entity by the given vector.

```
BOOL   PCALLBACK   DoEdit(PAN_CtlRange *pRange1,
                          PAN_CtlRange *pRange2,
                          LPMRK_ENTITYSPEC pEntitySpec);
```

Behavior differs depending on pRange1 and pRange 2:

- pRange1 == NULL and pRange2 == NULL:	Edit contents. (e.g. Text contents). Returning FALSE when entity is not yet finished is interpreted as “ABORT” and the entity is destroyed. If entity is already finished, return TRUE if entity was changed and FALSE if it was not.
- pRange1 != NULL and pRange2 == NULL:	Control point at pRange1->min got moved to pRange1->max. Return TRUE is entity was changed and FALSE if it was not.
- pRange1 != NULL and pRange2 != NULL:	Warp the entity based on the warping factor indicated by pRange1 and pRange2. Return TRUE if entity was changed and FALSE if it was not.

```
void   PCALLBACK   DoCopy(LPMRK_ENTITYSPEC pEntitySpec);
```

Copies the internal data structure of the UDE entity. Pointer to the copied data is returned in the data pointer member of MRK_UdeInfo part of pEntitySpec.

Markup Control API

Overview

The markup control interface is implemented through a message based API. The application sends command messages to the markup control. The markup control sends notification messages to the application to signal significant events. In addition, the markup control provides a set of exported functions, which can be called by the application and the UDE DLLs, to handle common functionality (e.g. memory allocation, entity drawing, etc...).

DLL Loading/Unloading:

int PCALLBACK LoadMarkupControl();

An application using the markup control DLL should call LoadMarkupControl() once at the start of the execution of the program. Returns TRUE on success and FALSE on failure.

int PCALLBACK FreeMarkupControl();

Should be called by the application before it terminates. Returns TRUE on success and FALSE on failure.

Control Creation:

Calling the exported CreateMarkupCtl function creates a markup control. The syntax of the function is as follows:

HWND CreateMarkupCtl(HWND hWndParent, int x, int y, int cx, int cy);

hWndParent is a handle to the markup control parent (this is the window to which notification messages are sent). The rest of the arguments specify the control's initial position and size. The newly created control is hidden i.e. Application is responsible for making it visible. Also, the new control does not contain any markup objects. An application must send at least one MRK_NEW message to the newly created control in order to create a markup object.
If successful, the function returns a handle to the newly created control; otherwise, it returns NULL.

Additional Exported Functions:

void huge * PCALLBACK MrkAlloc(long Size);
void PCALLBACK MrkFree(void huge *Ptr);

Memory management functions. Memory allocated with MrkAlloc() can only be freed by calling MrkFree().

void PCALLBACK MrkMemCopy(void huge *p*dest*, void huge **psrc*,
long *count*)
copies *count* bytes of p*src* to p*dest.*

int PCALLBACK MrkRotate(PAN_Point FAR *pPoint, PAN_Point FAR
***pCenter, Real Angle);**

Rotates pPoint around pCenter by Angle (radians).

int PCALLBACK MrkSegmentVisibleInBox(PAN_Point FAR *pPoint1,
PAN_Point FAR *pPoint2, PAN_Point FAR *pMin,
PAN_Point FAR *pMax);

Does the line segument given by pPoint1 and pPoint2 intersect the box given by pMin and pMax?

**int PCALLBACK MrkBoxVisibleInBox(PAN_Point FAR *pMin1,
PAN_Point FAR *pMax1, PAN_Point FAR *pMin2,
PAN_Point FAR *pMax2);**

Does the box given by pMin1 and pMax1 intersect the box given by pMin2 and pMax2?

**int PCALLBACK MrkArcVisibleInBox(PAN_Point FAR *pCenter,
PAN_Point FAR *pRad1, PAN_Point FAR *pRad2,
PAN_Point FAR *pMin, PAN_Point FAR *pMax,
Real StAng, Real EndAng);**

Does the given arc intersect the given box?

**int PCALLBACK MrkSegIntersectBox(PAN_Point FAR *pPoint1,
PAN_Point FAR *pPoint2, PAN_CtlRange FAR *pBox,
PAN_Point FAR *pPoint3, PAN_Point FAR *pPoint4);**

Computes intersection points between line and box. Result is copied to pPoint3 and pPoint4. Returns the number of intersection points.

**Real PCALLBACK MrkDistance(PAN_Point FAR *pP1, PAN_Point FAR
*pP2);**

Returns the 2D distance between two points.

**Real PCALLBACK MrkDistance3D(PAN_Point FAR *pP1, PAN_Point FAR
*pP2);**

Returns the 3D distance between two points.

**int PCALLBACK MrkDoEdit(PAN_Point FAR *pPoint,
PAN_CtlRange FAR *pRange1, PAN_CtlRange FAR *pRange2);**

Warping of pPoint.

**int PCALLBACK MrkDoEditRadius(PAN_Point FAR *pCenter, PAN_Point FAR *pMajorAxis,
PAN_CtlRange FAR *pRange1, PAN_CtlRange FAR *pRange2);**

Warping of an ellipse when the bounding box changes (pRange1 and pRange2). The function returns the new values for the center and the major axis.

**void PCALLBACK MrkArcExtents(MRKENTHANDLE handle,
PAN_CtlRange FAR *pRange, PAN_Point FAR *pCenter,
PAN_Point FAR *pRad1, PAN_Point FAR *pRad2,
Real stang, Real endang);**

Returns the arc Bounding Box inside pRange

**void PCALLBACK MrkDrawArc(MRKENTHANDLE handle, HDC hdc,
PAN_Point FAR *pCenter, PAN_Point FAR *pRad1,
PAN_Point FAR *pRad2, Real stang, Real endang,
DOWRD dwMode, LPMRK_DrawInfo pDrawInfo);**

Draws an arc using the drawing information set in pDrawInfo and the drawing mode specified by dwMode (currently we only support MRK_RENDERMODEXOR).

**void PCALLBACK MrkLineExtents(MRKENTHANDLE handle,
PAN_CtlRange FAR *pRange, int ArrowHead,
int ArrowTail);**

Returns the line Bounding Box inside pRange.

**void PCALLBACK MrkDrawLine(MRKENTHANDLE handle, HDC hdc,
PAN_Point FAR *pPoint1, PAN_Point FAR *pPoint2,
DOWRD dwMode, LPMRK_DrawInfo pDrawInfo);**

Draws a line from pPoint1 to pPoint2 using the drawing information set in pDrawInfo and the drawing mode specified by dwMode (currently we only support MRK_RENDERMODEXOR).

**void PCALLBACK MrkPolyExtents(MRKENTHANDLE handle,
PAN_CtlRange FAR *pRange, int numPts,
PAN_Point FAR *pPts, int ArrowHead,
int ArrowTail);**

Returns the polyline bounding box inside pRange.

**void PCALLBACK MrkDrawPoly(MRKENTHANDLE handle, HDC hdc,
int numPts, PAN_Point FAR *pPts,
DOWRD dwMode, LPMRK_DrawInfo pDrawInfo);**

Draws a polyline using the drawing information set in pDrawInfo and the drawing mode specified by dwMode (currently we only support MRK_RENDERMODEXOR).

**void PCALLBACK MrkTextExtents(MRKENTHANDLE handle, LPCSTR
szText, int Align, PAN_CtlRange FAR *pRange);**

Returns the text Bounding Box inside pRange

**void PCALLBACK MrkDrawText(MRKENTHANDLE handle, HDC hdc,
PAN_Point FAR *pPoints, LPCSTR szText, int Align,
DOWRD dwMode, LPMRK_DrawInfo pDrawInfo);**

Draws a text string using the drawing information set in pDrawInfo and the drawing mode specified by dwMode (Supports MRK_RENDERMODEXOR, and MRK_RENDERMODEPIXELFONT which specifies that the entity font size is given in pixels and not in world coordinates).

**void PCALLBACK MrkDrawTextBox(LPMRK_EntitySpec pEntity, PAN_CtlRange *pRange,
MRK_RenderOptions *pRenderSpec);**

Draws a rectangle around the text string.

**void PCALLBACK MrkDrawRect(MRKENTHANDLE handle, HDC hdc,
PAN_Point FAR *pPoint1, PAN_Point FAR *pPoint2,
DOWRD dwMode, LPMRK_DrawInfo pDrawInfo);**

Draws a rectangle using the drawing information set in pDrawInfo and the drawing mode specified by dwMode (currently we only support MRK_RENDERMODEXOR).

**BOOL PCALLBACK MrkDoFontDialog(HWND hOwner, LPLOGFONT
pFont);**

Shows font dialog and lets the user select font to set.

void PCALLBACK MrkSetFont(MRKENTHANDLE Handle, LPLOGFONT

pFont);

Sets the font of the entity specified by the handle.

void PCALLBACK MrkGetFont(MRKENTHANDLE Handle, LPLOGFONT pFont);

Gets the font of the entity specified by the handle.

HWND PCALLBACK MrkGetBaseWindow(HWND hwndCtl);

Returns the handle of the window being marked-up.

BOOL PCALLBACK MrkGetBaseExtents(HWND hwndCtl, PAN_CtlRange *pExtents);

Gets the 3D base file extents.

void PCALLBACK MrkTranslateFromFont(LPLOGFONT Font, char FAR *font);
void PCALLBACK MrkTranslateToFont(char FAR *font, LPLOGFONT Font);

Convert between Windows LOGFONT struct and a platform independent font struct (important because sizeof(LOGFONT) varies between WIN16 and WIN32). The buffer must be large enough to hold MRK_CST_FONTSIZE characters.

void PCALLBACK MrkAdjustPoints(HWND hwndCtl, PAN_Point FAR *pPoints, int nPoints);
Transforms the markup points pPoints using the new scale and offset of the base file.

BOOL PCALLBACK MrkSnap(MRKENTHANDLE Handle, PAN_Point *pPoint, DWORD dwSnapTo);

Snapping support;
pPoint:(IN) Client coord point. (OUT) World coord point if successful.
dwSnapTo: (IN) Combination of MRK_SNAPTO_XXX.
The function returns TRUE if it was able to snap. FALSE otherwise.

BOOL PCALLBACK MrkSnapSupport(MRKENTHANDLE Handle, DWORD dwSnapSupport);

Informs the application which snap type the entity will support.

BOOL PCALLBACK MrkWorld3DtoWorld2D(HWND hwndCtl, PAN_Point *pPoint);

Converts a 3D point to a 2D point coordinates.

BOOL PCALLBACK MrkWorld2DtoWorld3D(HWND hwndCtl, PAN_Point *pPoint);

Converts a 2D point to a 3D point coordinates.

void PCALLBACK MrkDrawGrip(MRK_RenderOptions *pRenderSpec, LPMRK_EntitySpec pEntitySpec, PAN_Point *pGripCenter);

Draws a filled rectangle centered at the grip point.

void PCALLBACK MrkDrawExtensionLine (MRK_RenderOptions *pRenderSpec, LPMRK_EntitySpec pEntitySpec, PAN_Point *pStartPt, PAN_Point *pEndPt);
This function simply draws a line from pStartPt to pEndPt and extends it by a small amount at the end. It is used to draw the dimension extension lines.

void PCALLBACK MrkGetTextBoxControlPoints (HWND hwndCtl, MRKENTHANDLE Handle, LPCSTR szText, PAN_Point FAR *pTextPt, int NumPts, PAN_Point *pPoints);

Computes the control points of a text box. This function supports four points (NumPts should be equal to 4.)

void PCALLBACK MrkFillInDrawInfoStruct (MRK_RenderOptions *pRenderSpec, int FillType, int StartArrow, int EndArrow MRK_DrawInfo *pDrawInfo);

Utility function to fill up the DrawInfo structure.

PAN_CtlRange PCALLBACK MrkGet2DSnapBox(LPMRK_EntitySpec pEntity, PAN_Point *pBoxCenter, int Size);

Computes a rectangle of the specified size (in pixels) around the given point.

BOOL PCALLBACK Mrk3DProjectPoint(HWND hwndCtl, PAN_Point *pPoint, PAN_Point *pInPlaneNormal, PAN_Point *pInPlanePt);

Projects a 3D point onto the screen.

BOOL PCALLBACK MrkIsReadOnly(MRKENTHANDLE Handle);

Is the given entity read only ?.

LRESULT PCALLBACK MrkGetUserName(MRKENTHANDLE Handle, int nBufSize, char FAR *pBuf);

Gets the user name from the application.

void PCALLBACK MrkDrawCustomIcon(MRKENTHANDLE Handle, HDC hdc, PAN_Point Pt, POINT ptIconSize, LPCSTR szIconName, DWORD dwMode);

Draws a custom icon with the specified parameters.

BOOL PCALLBACK MrkGetEntityBoundingBox(MRKENTHANDLE Handle, BOOL fChildren, PAN_CtlRange* pBBox);
Return the bounding box of the given entity.

Command Messages:

Note:

(A):	Applies to active markup object
(C):	Applies to markup control

Unless specified otherwise, all indexing are zero based.

Markup Object messages:

MRK_NEW			(A)
	WPARAM:	int (nIndex)	N/A
	LPARAM:	Unused	N/A
	Returns:	BOOL (success)	

Creates a new markup object which will have the index specified by (wParam-1). If wParam <= 0, the object is assigned an index equal to the current number of markup objects. The new markup object is initially hidden and has the following default properties:

- One page
- One layer : (name : "0", color RED)
- Default color: BYLAYERRGB
- Default palette

MRK_DELETE (A)

WPARAM:	int (nIndex)	in
LPARAM:	Unused	N/A
Returns:	BOOL (success)	

Deletes the markup object with the given index. All layers, palette entries, and entities associated with the markup object are destroyed.

MRK_QUIT (C)

WPARAM:	Unused	N/A
LPARAM:	Unused	N/A
Returns:	BOOL (success)	

Destroys the entire contents of the markup control. This message should be send before calling DestroyWindow() to destroy the control window. Note that changes to the markup objects will not be saved automatically. It is the responsibility of the client application to check for changes, using MRK_ISMODIFIED and save modified markup objects using MRK_SAVE.

MRK_SETACTIVE (C)

WPARAM:	int (index)	in
LPARAM:	Unused	N/A
Returns:	BOOL (success)	

Sets the index of the active markup object.

MRK_GETACTIVE (C)

WPARAM:	Unused	N/A
LPARAM:	Unused	N/A
Returns:	int nActiveMarkupObject	

Returns the index of the currently active markup object. If the markup control does not contain any markup objects, -1 is returned.

MRK_GETNUMMARKUPS (C)

WPARAM:	Unused	N/A
LPARAM:	Unused	N/A
Returns:	int	

Returns the number of markup objects contained within the markup control (zero if none).

MRK_REDDELALL (A)

WPARAM:	Unused	N/A
LPARAM:	Unused	N/A
Returns:	BOOL (success)	

Destroys all entities belonging to the current markup object. The markup object itself is not destroyed and all its attributes are left intact (e.g. layers, palette…)

Palette messages:

MRK_SETPALETTE (C)

WPARAM:	int (number of entries in the buffer)	N/A
LPARAM:	const LPLOGPALETTE (palette buffer)	in
Returns:	BOOL (success)	

Sets the markup control palette to the values pointed to by LPARAM. Old palette is destroyed.

MRK_GETPALETTE (C)

WPARAM:	int (PalCount)	N/A
LPARAM:	LPLOGPALETTE (buffer to hold entries)	out
Returns:	int (# of entries in palette/ # of entries copied)	

If wParam is 0, the number of palette entries is returned. Otherwise, the requested number of palette entries is copied to the buffer.

MRK_SETFGBGCOLOR (C)

WPARAM:	BOOL fBgColor	N/A
LPARAM:	COLORREF	in
Returns:	BOOL (success)	

Sets the markup controls background color if fBgColor is 1. Setting the foreground color is not implemented.

MRK_GETFGBGCOLOR (C)

WPARAM:	BOOL fBgColor	N/A
LPARAM:	COLORREF FAR *	out
Returns:	BOOL (success)	

Gets the markup control background color if fBgColor is 1. Getting the foreground color is not implemented.

Layer messages:

MRK_SETLAYERS (A)

WPARAM:	int (number of layers)	in
LPARAM:	LPPAN_LAYER	in
Returns:	BOOL (success)	

Sets layer information for the active markup object. Old layer info is destroyed.

MRK_GETLAYERS (A)

WPARAM:	int (number of layers)	in
LPARAM:	LPPAN_LAYER	out
Returns:	int (layer count/# of layers copied)	

If wParam is 0, the layer count is returned. Otherwise, the requested number of layers is copied to the buffer.

MRK_DELETELAYER (A)

WPARAM:	int (layer index to be deleted)	in
LPARAM:	unused	
Returns:	BOOL (success)	

Delete the layer and all entities that belong to this layer.

Page Information messages:

MRK_SETPAGE (C)

WPARAM:	int	in
LPARAM:	Unused	N/A
Returns:	BOOL (success)	

Sets the current page index to the specified value. The Page number is 1-indexed. If the page number set is greater than the number of existing pages, a new page is automatically created. Note: After changing pages, it is important to call MRK_SETBASEEXTENTS and MRK_SETVIEWEXTENTS if the dimensions of the new page have changed.

MRK_GETPAGE (C)

WPARAM:	Unused	N/A
LPARAM:	LPINT lpPage	out
Returns:	BOOL (success)	

Retrieves the current page index. The Page number is 1-indexed. A new markup control defaults to page 1.

MRK_GETNUMPAGES (C)

WPARAM:	Unused	N/A
LPARAM:	Unused	N/A
Returns:	int	

Retrieves the maximum number of pages used by entities in the markup control. If the markup control has more than one markup object, then the maximum page count over all markup objects is returned.

View messages:

MRK_SETBASEEXTENTS (C)

WPARAM:	Unused	N/A
LPARAM:	PAN_CtlRange FAR *	in
Returns:	BOOL (success)	

Sets the limits of the markup's world coordinate extents. This is normally set to the limits of the base drawing.

MRK_GETBASEEXTENTS (C)

WPARAM:	Unused	N/A
LPARAM:	PAN_CtlRange FAR *	out
Returns:	BOOL (success)	

Retrieves the limits of the markups' world coordinate extents, i.e. the range set with the most recent MRK_SETBASEEXTENTS.

MRK_SETVIEWEXTENTS (C)

WPARAM:	Unused	N/A
LPARAM:	PAN_CtlRange FAR * (extents)	in
Returns:	BOOL (success)	

Sets the view extents of the markup control in world coordinates. This is normally set to the current view extents of the base drawing.

MRK_GETVIEWEXTENTS (C)

WPARAM	Unused	N/A

LPARAM:	PAN_CtlRange FAR * (extents)	out
Returns:	BOOL (success)	

Retrieves the current view extents of the markup control in world coordinates. i.e. the range set with the most recent MRK_SETVIEWEXTENTS.

MRK_SETROTATE (C)

WPARAM	TRUE/FALSE	in
LPARAM:	Real FAR * (rotation angle in radians)	in
Returns:	BOOL (success)	

If wParam == TRUE, the base file rotation angle is set (this value is stored in the markup file when the markup is saved, but is otherwise unused by the markup control)

If wParam == FALSE, the markup control rotation angle is set. This angle is used to rotate the contents of the markup control.

MRK_GETROTATE (C)

WPARAM	TRUE/FALSE	in
LPARAM:	Real FAR * (rotation angle in radians)	out
Returns:	BOOL (success)	

If wParam == TRUE, the base file rotation angle is retrieved. Otherwise, it returns the markup control rotation angle.

MRK_SETFLIP (C)

WPARAM	Unused	N/A
LPARAM:	FLIP_NULL/FLIP_X/FLIP_Y/FLIP_XY	in
Returns:	BOOL (success)	

Sets markup control-flip value.

MRK_GETFLIP (C)

WPARAM	Unused	N/A
LPARAM:	int FAR *	out
Returns:	BOOL (success)	

Returns current markup control flip value. lParam points to an int variable that will hold the flip value.

MRK_GETPAGESIZE (C)

WPARAM	Unused	N/A
LPARAM:	PAN_CtlRange FAR * (extents)	out
Returns:	BOOL (success)	

Retrieves the true extents (limits) of the markup control contents. Range is computed based on all visible entities in all markup objects..

Coordinate messages:

MRK_CLIENTTOWORLD (C)

WPARAM:	Unused	N/A
LPARAM:	PAN_CtlPos FAR *	in/out
Returns:	BOOL (success)	

Converts a position in client coordinates to world coordinates. Returns FALSE if the application has not yet set the view extents.

MRK_WORLDTOCLIENT (C)

WPARAM:	Unused	N/A
LPARAM:	PAN_CtlPos FAR *	in/out
Returns:	BOOL (success)	

Converts a position in world coordinates to client coordinates. Returns FALSE if the application has not yet set the view extents.

MRK_GETMODELVIEWTRANSFORM

WPARAM:	Unused	
LPARAM:	LPHmatrix	out
Return	BOOL (success).	

MRK_SETMODELVIEWTRANSFORM

WPARAM:	Unused	
LPARAM:	LPHmatrix	in
Return	BOOL (success).	

MRK_LEFTMOUSEDOWN

WPARAM:	Unused	
LPARAM:	PAN_CtlPos* World coordinate Point	in
Returns:	BOOL (success)	

Allows application to simulate the left mouse down action.

Entity Manipulation Messages:

MRK_ADDENTITY (C)

WPARAM:	Unused	N/A
LPARAM:	LPMRK_ENTITYSPEC	in
Returns:	ENTHANDLE (new handle)	

Enables the application to add markup entities without using the built-in user interface. The message supports the addition of both top level and child entities: To add a top level entity, ParentHandle member of MRK_EntitySpec must be set to NULL. To add a child entity, set the ParentHandle value to the handle of the desired parent entity. Note: Currently, we only support text, links, and notes as child entities.

Returns the handle of the newly added entity on success or NULL on failure.

MRK_EDITENTITY (C)

WPARAM:	Unused	N/A
LPARAM:	ENTHANDLE	in
Returns:	BOOL (success)	

When this message is received, the markup control sends a MRKN_EDIT notification message to the application allowing it to edit the entity. If the application does not edit the entity, the markup control performs the built in editing action for the particular entity type. Returns FALSE if it fails to locate the entity with the specified handle.

MRK_GETENTITIES: **(C)**

WPARAM:	TypeOfGet (int)	In
LPARAM:	LPMRK_GetEntities	In
Returns:	BOOL (success)	

Returns the count and possibly the handles of the entities with the given specifications. The group of entities parsed is set by wParam, which can be one of the following.

- GET_ALLENTITIES	All markup entities.
- GET_SELECTED	Only selected entities
- GET_ENTCHILDREN	Children of the entity whose handle is specified inside LPMRK_GetEntities.

If TypeOfGet value in MRK_GetEntities is not set to ENTTYPE_NULL. the search is restricted to entities of that type.

If the handle buffer inside MRK_GetEntities is NULL, the entity count is returned. Otherwise, the buffer is filled with the handles of the entities found and the min of the buffer size and the entity count is returned.

MRK_ENUMENTITIES: (C)

WPARAM:	EnumType	In
LPARAM:	LPMRK_EnumEntities	In
Returns:	BOOL (success)	

The markup control calls the EnumProc, specified inside MRK_EnumEntities, once for each entity with the given specifications. The group of entities parsed is set by wParam, which can be one of the following.

- ENUM_ALLENTITIES	All markup entities.
- ENUM_SELECTED	Only selected entities
- ENUM_ENTCHILDREN	Children of the entity whose handle is specified inside MRK_EnumEntities.

If EnumType value in MRK_EnumEntities is not set to ENTTYPE_NULL, the enumeration is restricted to entities of that type.

The EnumProc return value determines the action to apply to the entity:

- ENUM_NULL	Do nothing.
- ENUM_EDIT	Edit the entity.
- ENUM_DEL	Delete the entity.
- ENUM_SEL	Select the entity.
- ENUM_UNSEL	Unselect entity if selected.
- ENUM_QUIT	Stop enumeration.

MRK_LOCKENTITY (C)

WPARAM:	Unused	N/A
LPARAM:	LPMRK_EntitySpec	In/Out
Returns:	BOOL (success)	

Locks the entity whose handle is given in MRK_EntitySpec: The entity info is returned inside MRK_EntityInfo. The message fails if the handle is invalid or the entity is already locked.

MRK_UNLOCKENTITY (C)

WPARAM:	BOOL fApplyChanges	in
LPARAM:	LPMRK_EntitySpec	In
Returns:	BOOL (success)	

Unlocks an entity previously locked using MRK_LOCKENTITY. If fApplyChanges is TRUE, the entity contents are modified using the information in MRK_EntitySpec. The message fails if the handle is invalid or if the entity has not been locked.

MRK_COPYLAYER (C)

WPARAM:	BOOL bNotify	in

LPARAM:	LPCOPYLAYERSTRUCT	in
Returns:	BOOL (success)	

Copies all entities on the source layer of the source markup object to the destination layer of the destination markup object. Source/Destination layers and markup objects are passed inside COPYLAYERSTRUCT. If bNotify is TRUE, notification message MRKN_COPYLAYER will be sent whenever a single entity was being copied.

MRK_GETENTITYEXTRADATA (C)

WPARAM:	EnumType currently only support MRK_ENTEXTRADATA_USERDATA in	
LPARAM:	LPMRK_EntitySpec	in/out
Returns:	BOOL (success)	

Get extra data attached to the entity whose handle is given in MRK_EntitySpec. The extra data type is specified in WPARAM, currently only user data is supported. The entity extra data is returned inside LPMRK_EntitySpec.

MRK_GROUP (C)

WPARAM:	int (number of entities in handle buffer)	in
LPARAM:	LPMRKENTHANDLE(handle buffer)	in
Returns:	BOOL (success)	

Group the specified entities and create a group entity. The entities that are to be grouped must have the same page index and belong to the same redline.

MRK_UNGROUP (C)

WPARAM:	int (number of group entities in handle buffer) in	
LPARAM:	LPMRKENTHANDLE (group entity handle buffer) in	
Returns:	BOOL (success)	

Ungroup the specified group entities. The original entities in each group entity are restored and the group entities are destroyed.

Property messages:

MRK_SET_EPROP (C)

WPARAM:	WPARAM (property)	in
LPARAM:	LPARAM (parameter)	in
Returns:	BOOL (success)	

Changes the current settings of the markup control to which effects the drawing attributes of any entity that are subsequently created. Refer to the table below.

MRK_GET_EPROP (C)

WPARAM:	WPARAM (property)	in
LPARAM:	LPARAM (parameter)	out
Returns:	BOOL (success)	

Retrieves the current settings of the markup. Refer to the following table.

PROPERTY (wParam)	PARAMETER (lParam)
ME_COLOR	Explicit RGB COLOR, or BYLAYERRGB to set BYLAYER color, or WIPEOUTRGB to set the erase color.

ME_FILLCOLOR	Explicit RGB COLOR, or BYLAYERRGB to set BYLAYER color, or WIPEOUTRGB to set the erase color, or LINECOLORRGB to make the fill color follow the line color.
ME_FILLTYPE	LPINT: MRK_FILLNONE, MRK_FILLSOLID, MRK_FILLTRANSPARENT.
ME_FONT	LPLOGFONT
ME_LAYER	LPINT: Layer index
ME_PENSTYLE	LPINT: MRK_PENSTYLE_SOLID / MRK_PENSTYLE_DASH / MRK_PENSTYLE_DOT / MRK_PENSTYLE_DASHDOT / MRK_PENSTYLE_DASHDOTDOT / MRK_PENSTYLE_HOLLOW MRK_PENSTYLE_ARC MRK_PENSTYLE_TRIANGLE
ME_PENWIDTH	LPDOUBLE Pen Width. If the width given is negative, then it is assumed to specify a Pixel width that remains constant, independent of the viewextents. A positive width indicates a value in world coordinates.
ME_LINEARROWSTART	LPINT (TRUE or FALSE)
ME_LINEARROWEND	LPINT (TRUE or FALSE)
ME_SNAPTO	LPINT: MRK_SNAPTO_VERTEX/ MRK_SNAPTO_EDGE/ MRK_SNAPTO_MIDEDGE/ MRK_SNAPTO_ARCCENTER/ MRK_SNAPTO_FACE or MRK_SNAPTO_ALL
ME_PENSTYLE_INFO	LPMRK_LineStyleInfo that describes the extra line style info
ME_USERDATASET	LPMRK_UserDataSetInfo that describes the user data set

MRK_SET_MOPROP (A)

WPARAM:	WPARAM (property)	in
LPARAM:	LPARAM (parameter)	in
Returns:	BOOL (success)	

Sets the properties of the current markup object. Refer to the table below.

MRK_GET_MOPROP (A)

WPARAM:	WPARAM (property)	in
LPARAM:	LPARAM (parameter)	out
Returns:	BOOL (success)	

Retrieves properties of the current markup object. Refer to the following table.

PROPERTY (wParam)	**PARAMETER (lParam)**
MO_INS	LPMRK_BaseInfo: Base file information
MO_INFO	LPSTR: 5 Title strings + 5 Description strings all separated by line break char '\n'.
MO_CURSOR	LPINT: HCURSOR value.

MO_VISIBLE	LPINT: Visibility flag (TRUE or FALSE)
MO_READONLY	LPINT: ReadOnly flag (TRUE or FALSE)
MO_ISMODIFIED	LPINT: Modified flag (TRUE or FALSE)
MO_SCALECALIBRATION	LPDOUBLE: X and Y scaling factors used for measurement.
MO_TRUECOLOR	BOOL(SET)/LPINT(GET): If set to TRUE, it disables color inversion when the entity color matches the background color. Default is FALSE.
MO_TRUEBACKGROUND	BOOL(SET)/LPINT(GET): If set to FALSE, it indicates that the control's background color does not match the drawing's background color. In this case, the highlight fill is drawn using a dithered pattern. Default is TRUE.
MO_BASEWINDOW	HWND(SET)/(HWND*)(GET): Handle of window being marked-up.
MO_USEUSERFONT	Set TRUE if the application want markup control to use the font specified in ini file instead of the font read from red line file
MO_VIEWMODESEL	Set TRUE if the view mode drawing entity selection mode is active. In this case, the view mode selection take precedence when there is mouse message conflict between drawing entity selection and markup entity selection

Action Messages:

MRK_SETACTION		(C)
WPARAM:	int MRK_ACTION_XXX	in
LPARAM:	Unused	N/A
Returns:	BOOL (success)	

Sets current action state of the markup control. Note that this is the only message that is allowed to alter the markup control action state..

MRK_GETACTION		(C)
WPARAM:	Unused	N/A
LPARAM:	Unused	N/A
Returns:	ACTIONCODE	

Gets the current action state of the markup control.

Action codes:
MRKP_ACTION_NONE: Non-Edit mode. All mouse messages are forwarded to window underneath.
MRKP_ACTION_ADD: Entity Addition mode. Only mouse messages that are not related to the creation of the new entity are forwarded.
MRKP_ACTION_SEL: Pure selection mode. No mouse messages are forwarded
MRKP_ACTION_HYBRID: Hybrid selection mode. Only mouse messages that do not affect selection are forwarded.

Note: While the Control Key is pressed the markup control simulates a MRK_ACTION_NONE mode i.e. all mouse messages are forwarded. The current mode is resumed as soon as the key is released.

MRK_SETENTITY (A)

WPARAM:	int (entity ID)	In
LPARAM:	LPCSTR: UDE name (NULL if not UDE)	In
Returns:	BOOL (success)	

Sets the current entity type. This is the type of entity that will be created if the application sets the MRKP_ACTION_ADD state and an entity is defined through the user interface. To set a UDE Type: Set wParam to ENTTYPE_UDE_BASE plus the zero based index of the UDE in the UDE list, and lParam to NULL. Or, set wParam to ENTTYPE_UDE_BASE and lParam to a string pointer containing the UDE name.

MRK_GETENTITY (A)

WPARAM:	Unused	N/A
LPARAM:	LPCSTR* to hold UDE name pointer (if any)	out
Returns:	int (entity ID)	

Retrieves the current entity type. If the entity is built-in, the corresponding type is returned (ENTTYPE_XXX). For UDEs, the return value is equal to ENTTYPE_UDE_BASE plus the zero based index of the UDE in the UDE list. If lParam is not NULL, the UDE name is also returned.

MRK_UNDO (C)

WPARAM:	BOOL fQueryOnly	in
LPARAM:	Unused	N/A
Returns:	BOOL (success)	

Undoes the previous operation if possible. If fQueryOnly is set, nothing is undone, but returns whether an Undo is possible.

MRK_REDO (C)

WPARAM:	BOOL fQueryOnly	in
LPARAM:	Unused	N/A
Returns:	BOOL (success)	

Does the next operation if possible. If fQueryOnly is set, nothing is redone, but returns whether a redo is possible.

Selection messages:

MRK_SETSEL (A)

WPARAM:	int (entity type : ENTTYPE_XXX)	in
LPARAM:	PAN_CtlRange FAR * (selRange)	in
Returns:	int (num sel)	

Selects all entities that intersect the specified range. Returns the number of entities that are selected. If WPARAM != ENTTYPE_NULL, selection is restricted to entities of the type specified.

MRK_CLEARSEL (A)

WPARAM:	Unused	N/A
LPARAM:	Unused	N/A
Returns:	BOOL (success)	

Clears existing selection.

MRK_ADDSEL (A)

WPARAM:	int (number of entities in handle buffer)	in

LPARAM:	LPMRKENTHANDLE(handle buffer)	in
Returns:	BOOL (success)	

Selects the specified entities

MRK_REMOVESEL (A)

WPARAM:	int (number of entities in handle buffer)	in
LPARAM:	LPMRKENTHANDLE(handle buffer)	in
Returns:	BOOL (success)	

Unselects the specified entities

Clipboard messages:

MRK_COPYCLPBRD (A)

WPARAM:	Unused	N/A
LPARAM:	Unused	N/A
Returns:	BOOL (success)	

Copies currently selected entities to the clipboard. The markup's own clipboard format is used .

MRK_PASTECLPBRD (C)

WPARAM:	Unused	N/A
LPARAM:	Unused	N/A
Returns:	BOOL (success)	

Pastes the contents of the clipboard to the currently active markup. The data is pasted in the following order of priority: Markup's private clipboard data is pasted as markup entities. RTF data is pasted as a note entity. Text data is pasted as a text entity. Graphic data (wmf, Bitmap & DIB) is pasted as a symbol entity.

I/O messages:

MRK_ISMRKFILE

WPARAM:	Unused	N/A
LPARAM:	LPCSTR (file name)	in
Returns:	BOOL (fValid)	

Tests if the given file is a valid markup file.

MRK_READ (A)

WPARAM:	Unused	N/A
LPARAM:	LPCSTR (file name)	in
Returns:	BOOL (success)	

Loads the contents of the given markup file into the currently active markup object. Existing contents of the markup object are cleared. The markup object is marked as unmodified.

MRK_SAVE (A)

WPARAM:	Unused	N/A
LPARAM:	LPCSTR (file name)	in
Returns:	BOOL (success)	

Saves the contents of the currently active markup object to the specified file. The markup object is marked as unmodified.

UDE messages:

MRK_GETUDEINFO

WPARAM:	int nUDE	in
LPARAM:	LPUDEINFO	out
Returns:	int numUDEs	

Returns information about the specified UDE. If LPARAM is zero, the total number of UDEs available is returned.

Render messages:

MRK_RENDERONTODC (C)

WPARAM:	Unused	N/A
LPARAM:	LPMRK_RenderOptions (options)	in
Returns:	BOOL (success)	

Renders entities in the source Rect. onto the device Rect. of the output DC using the parameters specified in MRK_RenderOptions. The current viewextents remain unchanged.

MRK_NOTEBATCHPRINT (C)

WPARAM:	Unused	N/A
LPARAM:	LPMRK_NotePrint (options)	In
Returns:	BOOL (success)	

Batch prints the note entities whose handles are passed inside MRK_NotePrint. Additional print options can also specified in MRK_NotePrint.

MRK_NOTEPREPAREPRINT (C)

WPARAM:	Unused	N/A
LPARAM:	LPMRK_NotePrint (options)	In
Returns:	int (page count)	

Called to get the print preview of one or multiple note entities with handles given in MRK_NotePrint (also contains the formatting information). Returns the number of pages. MRK_RENDERPAGE can then be called to render each page.

MRK_NOTERENDERPAGE (C)

WPARAM:	Unused	N/A
LPARAM:	LPMRK_NoteRender (options)	In
Returns:	BOOL (success)	

Sent after MRK_NOTEPREPAREPAGE to render a specific page onto a DC. The rendering information is passed in MRK_NoteRender.

MRK_NOTEENDPRINT (C)

WPARAM:	Unused	N/A
LPARAM:	Unused	N/A
Returns:	BOOL (success)	

Must be sent when done rendering note pages (Cleans up what was done in MRK_NOTEPREPAREPRINT).

Notification Messages:

Note that some of these notifications may require feedback from the client side through an appropriate return value (generally TRUE or FALSE) and data inserted in the structure being pointed to by lParam:

MRKN_CANCEL

WPARAM:	Unused	N/A
LPARAM:	Unused	N/A
Returns	Unused	

The cancel notification is sent to the parent window of the markup control whenever a built-in user interface operation is aborted. This usually occurs when the right mouse button is pressed.

MRKN_EDIT

WPARAM:	Unused	N/A
LPARAM:	LPMRK_ENTITYSPEC	in/out
Returns	-1 => Canceled, 0 => Not handled, 1 => Done	

The edit notification is sent to an application when the markup control requires additional information about an entity, or the application has caused an entity to be selected for modification.
If the application implements a method for modifying the entity, it should return a non-zero value (1 if entity was changed or -1 if editing was canceled). Otherwise, the markup control will use built-in facilities to modify the entity.

MRKN_FIRELINK

WPARAM:	Unused	in
LPARAM:	LPMRK_ENTITYSPEC	in
Returns	Unused	

Notification that an entity, that can be activated, has been double clicked.

MRKN_ENTITYADDED

WPARAM:	int (entity type ENTTYPE_XXX)	in
LPARAM:	Unused	N/A
Returns	Unused	

Sent to the markup controls' parent when an entity has been added through the user interface.

MRKN_SELCHANGED

WPARAM:	Unused	N/A
LPARAM:	Unused	N/A
Returns	Unused	

Sent when the set of selected entities has changed.

MRKN_MODIFIED

WPARAM:	Unused	N/A
LPARAM:	Unused	N/A
Returns	Unused	

Sent when markup contents are modified (modifications to markup entities only).

MRKN_CURSOR

WPARAM:	State of entity intersecting the cursor.	in
LPARAM:	MRKENTHANDLE	in/out

Returns	HCURSOR

Request for built-in cursor overloading. The client application can provide its own cursor. lParam specifies the handle of the entity that intersects the current cursor position (0 if none). if lParam is not zero, then wParam provides more information about the entity:

- MRK_ENTNOTSELECTED	Entity is not selected.
- MRK_ENTSELECTED	Entity is selected.
- MRK_ENTDRAGGING	Entity is being dragged with the mouse.

Returns handle of cursor to set or 0 to use the built in cursor.

MRKN_GETPROPERTY

WPARAM:	int (Buffer size)	in
LPARAM:	LPSTR (Buffer)	in/out
Returns	BOOL (success).	

Used by the markup control to query miscellaneous information from the application. The markup control passes the string describing the property in LPARAM. The application returns the value of the requested property in the same buffer. Currently, the markup control only requests the "UserName" property.

MRKN_CLIENTTOWORLD

WPARAM:	POINT_3D (3D conversion);	
	POINT_2D (2D conversion)	in
LPARAM:	PAN_CtlPos FAR*	in/out
Returns	BOOL (success).	

The notification message will pass the 2D client position to the application. The application will perform the conversion and return the corresponding world coordinates.

MRKN_SNAP

WPARAM:	Snap type; It could be any combination of the following:	
	MRK_SNAPTO_VERTEX	
	MRK_SNAPTO_EDGE	
	MRK_SNAPTO_MIDEDGE	
	MRK_SNAPTO_ARCCENTER	
	MRK_SNAPTO_FACE or	
	MRK_SNAPTO_ALL	
LPARAM:	PAN_CtlPos FAR*	in/out
Returns	BOOL (TRUE if able to snap, FALSE otherwise).	

The notification message will pass the 2D client position to the application and return the 3D coordinate position if it succeeded (able to snap).

MRKN_SNAPCHANGED

WPARAM:	Unused
LPARAM:	Old snap type. It could be any Combination of the following: (in/out)
	MRK_SNAPTO_VERTEX
	MRK_SNAPTO_EDGE
	MRK_SNAPTO_MIDEDGE
	MRK_SNAPTO_ARCCENTER

	MRK_SNAPTO_FACE or
	MRK_SNAPTO_ALL
Returns	BOOL (success).

The notification message will pass the old snap type and return the new snap type.

MRKN_SNAPSUPPORT

WPARAM:	Snap type; It could be any combination of the following:
	MRK_SNAPTO_VERTEX
	MRK_SNAPTO_EDGE
	MRK_SNAPTO_MIDEDGE
	MRK_SNAPTO_ARCCENTER
	MRK_SNAPTO_FACE or
	MRK_SNAPTO_ALL
LPARAM:	Unused
Returns	BOOL (success).

The message will notify the application of the snap types the markup supports.

MRKN_COPYLAYER

WPARAM:	Unused	
LPARAM:	MRKN_CopyLayer*	in
Returns	Unused	

The message will notify the application a single markup entity had been copied in the process of MRK_COPYLAYER.

Common Definitions:

The markup control defined constants that are included in file mrkupctl.h.

Entity Types:

```
// Definitions of entity types.
enum {
        ENTTYPE_NULL,
        ENTTYPE_LINE,               // Line Entity
        ENTTYPE_TEXT,               // Text Entity
        ENTTYPE_SOLID,              // Solid Entity
        ENTTYPE_CIRCLE,             // Circle Entity
        ENTTYPE_ARC,                // Arc Entity
        ENTTYPE_RECT,               // Rectangle  Entity
        ENTTYPE_FSTYLE,             // FreeStyle Entity
        ENTTYPE_NOTE,               // Note Entity
        ENTTYPE_LINK,               // Link Entity
        ENTTYPE_POLY,               // Polyline Entity
        ENTTYPE_VERTEXDIM,          // 3D Vertex Entity
        ENTTYPE_LENEARDIM,          // 3D Linear Dimension
                                    // Entity
        ENTTYPE_ARCDIM,             // 3D Arc Dimension
                                    // Entity
        ENTTYPE_ANGULARDIM,         // 3D Angular Dimension
```

```
                                        // Entity

        ENTTYPE_UDEBASE                 // User defined entity
};
```

```
// 3D Leader Sub Types.
enum {
        ENTITY_UDE_3DLEADER_TEXT,       // 3D text
        ENTITY_UDE_3DLEADER_NOTE,       // 3D Note
        ENTITY_UDE_3DLEADER_VERTEX,     // Vertex Dim.
};
```

Action Codes:

```
// Used by MRK_SETACTION and MRK_GETACTION messages.

enum {
        MRKP_ACTION_NONE,               // Transparent
        MRKP_ACTION_ADD,                // Add entity
        MRKP_ACTION_DEL,                // Delete entity
        MRKP_ACTION_COPY,               // Copy Entity
        MRKP_ACTION_EDIT,               // Edit Entity
        MRKP_ACTION_MOVE,               // Move Entity
        MRKP_ACTION_HYBRID              // Hybrid selection.
};
```

Property Codes:

```
// Basic attributes.  Used by MRK_SET_EPROP and MRK_GET_EPROP

enum {
        ME_NULL,
        ME_FONT,
        ME_FILLTYPE,
        ME_COLOR,
        ME_FILLCOLOR,
        ME_LAYER,
        ME_PENSTYLE,
        ME_PENWIDTH,
        ME_LINEARROWEND,
        ME_LINEARROWSTART,
        ME_SNAPTO,
        ME_PENSTYLE_INFO,
        ME_USERDATASET,
        ME_LAST
};

// Additional properties.  Used by MRK_SET_MOPROP and
// MRK_GET_MOPROP.

enum {
        MO_NULL = 0,
        MO_INS,
        MO_INFO,
        MO_CURSOR,
        MO_VISIBLE,
        MO_READONLY,
        MO_ISMODIFIED,
        MO_SCALECALIBRATION,
        MO_TRUECOLOR,
        MO_BASEWINDOW,
        MO_TRUEBACKGROUND,
        MO_USEUSERFONT,
        MO_VIEWMODESEL,
        MO_LAST
};
```

Text Alignment Codes:

```
// Alignment values.  Used by MRK_TextInfo, MrkDrawText() and
// MrkTextExtent()
enum {
          MRK_ALIGN_TOPLEFT,
          MRK_ALIGN_TOPCENTER,
          MRK_ALIGN_TOPRIGHT,
          MRK_ALIGN_CENTERLEFT,
          MRK_ALIGN_CENTERCENTER,
          MRK_ALIGN_CENTERRIGHT,
          MRK_ALIGN_BOTTOMLEFT,
          MRK_ALIGN_BOTTOMCENTER,
          MRK_ALIGN_BOTTOMRIGHT,
};
```

PenStyle Codes:

```
// Penstyle.
enum {
          MRK_PENSTYLE_SOLID,
          MRK_PENSTYLE_DASH,
          MRK_PENSTYLE_DOT,
          MRK_PENSTYLE_DASHDOT,
          MRK_PENSTYLE_DASHDOTDOT,
          MRK_PENSTYLE_HOLLOW
          MRK_PENSTYLE_ARC
          MRK_PENSTYLE_TRIANGLE
};
```

Fill Codes:

```
// Fill types.
enum {
          MRK_FILLNONE,                // Not filled.
          MRK_FILLSOLID,               // Solid filling.
          MRK_FILLTRANSPARENT,         // Highlight filling.
};
```

Entity Status Codes:

```
// Used by MRKN_SETCURSOR notification to indicate the status of
// the entity that intersects the cursor.
enum {
          MRK_ENTNOTSELECTED,          // Entity is not selected.
          MRK_ENTSELECTED,              // Entity is selected.
          MRK_ENTDRAGGING               // Entity is being dragged
                                        // with the mouse.
};
```

Entity GET/ENUM Codes:

```
// Which entities to get:  wParam for MRK_GETENTITIES.
enum {
          GET_ALLENTITIES,       // All entities in the control.
          GET_SELECTED,          // The selected entities.
          GET_ENTCHILDREN        // The children of the
          // entity with handle value
                                 //specified in MRK_GetEntities
};

// Which entities to enumerate:  wParam for MRK_ENUMENTITIES.
enum {
          ENUM_ALLENTITIES,      // All entities in the control.
          ENUM_SELECTED,         // The selected entities.
          ENUM_ENTCHILDREN       // The children of the entity with
          // handle value specified in
                                 // MRK_EnumEntities
};

// Return Values for the Entity Enumeration Procedure:
// Returned by the Application's Enumeration Procedure,
// invoked with the MRK_ENUMENTITIES Message.
enum {
          ENUM_NULL,             // Do nothing.
          ENUM_EDIT,             // Edit entity.
          ENUM_DEL,              // Delete entity.
          ENUM_SEL,              // Select entity (only if wParam ==
                                 // ENUM_ALLENTITIES ).
          ENUM_UNSEL,            // Unselect entity (for wParam !=
                                 // ENUM_ENTCHILDREN).
          ENUM_QUIT              // Stop enumeration.
};

// The type of extra data
enum {
          MRK_ENTEXDATA_USERDATA = 0,
};
```

Mouse Notification Codes:

```
//  Sent to MouseProc() entry function of a UDE DLL to notify the
// UDE of the current mouse action.
enum {
          MRK_LBUTTONDOWN,
          MRK_LBUTTONUP,
          MRK_LBUTTONDBLCLK,
          MRK_RBUTTONDOWN,
          MRK_RBUTTONUP,
		MRK_MOUSEMOVE
	};
```

Flip Codes:

```
// Flip values:  Used by MRK_SETFLIP and MRK_GETFLIP.
enum {
        FLIP_NULL,              // No flipping.
        FLIP_X,                 // Flip Horizontal axis
        FLIP_Y,                 // Flip vertical axis
        FLIP_XY                 // Flip both
};
```

Hyperlink Types:

```
// Different Link Types.  Specific to the Link Entity
enum {
        MLINK_TO_DDE,
        MLINK_TO_DLL,
        MLINK_TO_APP,
        MLINK_TO_FILE,
        MLINK_TO_SCRIPT
};
```

Data Structures:

The markup control data structures are included in file mrkupctl.h.

MRK_EntitySpec:

Structure that provides information about all markup entities. Used by a number of messages and as an argument to all UDE DLL entry functions:

```
typedef struct {
        MRK_CommonInfo  Com;  // Info common to all. entities
        union{
                MRK_LineInfo    Line;       // Line entity data
                MRK_FStyleInfo Fstyle;     // FreeStyle entity data
                MRK_PolyInfo    Poly;       // Ploy entity data
                MRK_SolidInfo   Solid;      // Solid entity data
                MRK_RectangleInfo Rect; // Rect entity data
                MRK_TextInfo    Text;      // Text entity data
                MRK_ArcInfo     Arc;       // Arc entity data
                MRK_CircleInfo  Circle;    // Circle entity data
                MRK_NoteInfo    Note;      // Note entity data
                MRK_LinkInfo    Link;      // Link entity data
                MRK_UdeInfo     Ude;       // UDE entity data
        } Ent;
} MRK_EntitySpec, FAR *LPMRK_EntitySpec;


// Common entity info.
typedef struct {
        int                     Type;               // Entity type.
        MRKENTHANDLE            Handle;             // Entity handle
        MRKENTHANDLE            ParentHandle;       // Parent entity
```

```
                                                        // handle
        HWND                    hWndCtl;                // Markup control
                                                        // handle
         Int            numChildren;                    // Number of child
                                                        // entities.
        DWORD           dwFlags;                        // Combination of
                                                        // the following flags:
                                        //MRK_ENTITY_FINISHED //MRK_ENTITY_FIREABLE
                                        //MRK_ENTITY_FILLABLE
                                        //MRK_ENTITY_HIDDEN
                                        //MRK_ENTITY_EDITABLE
                                        //MRK_ENTITY_UNICODE
                                        //MRK_ENTITY_ANISOTROPIC
        int             nRedlineIndex;  // Markup object index.
        int             nPageIndex;     // Page index.
        int             nLayerIndex;    // Layer index.
        COLORREF        LineColor;      // Line color.
        COLORREF        FillColor;      // Fill color.
        int             FillType;       // Fill type.
        int             PenStyle;       // Pen style.
        Real            PenWidth;       // Pen width.
        int             StartArrow;     // Start arrow type.
        int             EndArrow;       // End arrow type.
        Real            Rotation;       // Rotation angle in radians
        Char            szAuthor[_MAX_PATH]; //Author
        long            LastModifiedTime;               // the latest time when
                                                        // the entity is modified
        LOGFONT         Font;           // Entity font.
        MRK_LineStyleInfo LineStyleInfo;// Extra info for line style
        MRK_UserDataInfo UserDataInfo; // Entity user data
} MRK_CommonInfo, FAR *LPMRK_CommonInfo;

// Line info
typedef struct  {
        PAN_Point               Pt1;    // Start point.
        PAN_Point               Pt2;    // End point.
} MRK_LineInfo, FAR *LPRMK_LineInfo;

// FreeStyleinfo.
typedef struct  {
        LONG            numPts;         // Number of points.
        PAN_Point       huge    *pPoints;  // Buffer of points.
} MRK_FStyleInfo, FAR *LPRMK_FStyleInfo;

// Polyline info.
typedef struct  {
        LONG            numPts;         // Number of points.
        PAN_Point       huge    *pPoints;  // Buffer of points.
} MRK_PolyInfo, FAR *LPRMK_PolyInfo;

// Text info.
typedef struct  {
        PAN_Point       InsPoint;       // Text insertion point.
        int             Align;          // Text alignment.
        Real            Rotang;         // Text rotation angle.
        Real            Oblique;        // Text obliquing angle.
        LPCSTR          szText;         // Text string.
} MRK_TextInfo, FAR *LPRMK_TextInfo;

// Solid info.
typedef struct  {
        LONG            numPts;         // Number of points.
        PAN_Point       FAR     *pPoints;  // Buffer of points.
} MRK_SolidInfo, FAR *LPRMK_SolidInfo;

// Rectangle info
typedef struct  {
        PAN_Point       Pt1;            // First corner.
        PAN_Point       Pt2;            // Second corner.
} MRK_RectangleInfo, FAR *LPRMK_RectangleInfo;

// Arc info
typedef struct  {
        PAN_Point       Pt1;            // First corner of bbox
        PAN_Point       Pt2;            // Second corner of bbox.
        Real            StartAng;       // Start angle in radians.
        Real            EndAng;// End angle in radians.
} MRK_ArcInfo, FAR *LPRMK_ArcInfo;

// Circle info.
typedef struct  {
        PAN_Point       Pt1;            // First corner of bbox.
        PAN_Point       Pt2;            // Second corner of bbox.
} MRK_CircleInfo, FAR *LPRMK_CircleInfo;

// Note info.
typedef struct  {
        PAN_Point       InsPt;          // Insertion point.
        LPCSTR          szName;         // Note name.
        LPCSTR          szAuthor;       // Note author.
        LPCSTR          szKeyWords;     // Note Key words.
        LPCSTR          szIcon;         // Icon name.
        LONG            NoteLength;     // Length of note content.
        HPSTR           szNote;         // Note content.
} MRK_NoteInfo, FAR *LPRMK_NoteInfo;

// Link info
typedef struct  {
        PAN_Point       InsPt;          // Insertion point.
        int             LinkType;       // Link type.
        BOOL            fStartApp;      // Link flag.
        BOOL            fHideIcon;      // Do not display the icon.
        LPCSTR          szName;         // Link name.
        LPCSTR          szIcon;         // Icon name.
        LPCSTR          szDef1;         // Link definition 1.
        LPCSTR          szDef2;         // Link definition 2.
        LPCSTR          szDef3;         // Link definition 3.
        LPCSTR          szDesc;         // Link description.
} MRK_LinkInfo, FAR *LPRMK_LinkInfo;
```

```
// UDE info.
typedef struct  {
        LPCSTR          szName;         // UDE name.
        LONG            DataSize;       // UDE data size.
        void            huge    *pData;    // Pointer to UDE data.
} MRK_UdeInfo, FAR *LPRMK_UdeInfo;

// Extra info for line style
typedef struct {
        struct    {
                int nArcXRadius; // X radius of arc in pixels
                Real Ratio;       // X radius and Y radius ratio used to
                                  // define the arc
        } ArcStyleInfo;
        struct    {
                int nBottomEdge; // Bottom edge length of
                                  // triangle in pixels
                int nHeight;      // Height of the triangle in
                                  // pixels
        } TriangleStyleInfo;

} MRK_LineStyleInfo, FAR * LPMRK_LineStyleInfo;

// User Data Info
typedef struct {
        char      szAppID[_MAX_PATH];           // Application ID
        char      szDataSetType[_MAX_PATH];     // Dataset type
        long      nDataSize;                    // Data size of raw data
        void*     pData;                        // pointer to raw data
} MRK_UserDataSetInfo, FAR *LPMRK_UserDataSetInfo;

typedef struct {
        long                      nNumDataSets;   // number of datasets
        MRK_UserDataSetInfo*      pDataSets;      // pointer to datasets
} MRK_UserDataInfo, FAR *LPMRK_UserDataInfo;
```

MRK_RenderOptions:

Structure containing rendering information. Used by MRK_RENDERONTODC message and as an argument to the Draw() function inside a UDE DLL:

```
typedef struct {
        HDC             hdc;      // DC to use for rendering
        DWORD           mode;     // One or more render modes.
                                  // See below for explanation.
        PAN_CtlRange    source;   // Portion to be rendered.
        RECT            devRect;  // Output device rectangle.
        LPMRK_DrawInfo  lpDrawInfo; // Special drawing attributes.
} MRK_RenderOptions, FAR *LPMRK_RenderOptions;
```

If not NULL, lpDrawInfo contains special drawing attributes that apply to all entities e.g. specific line color or fill type. If lpDrawInfo is NULL, each entity is rendered using its own drawing attributes.

The following render modes are currently supported:

- MRK_RENDERMODEXOR
  Render in XOR mode.

- MRK_RENDEROMODEEDIT
  Used to inform the UDE DLL that the entity about to be drawn is being dragged/distorted (some UDEs may elect to draw just an outline in this case).
- MRK_RENDEROMODEMONOCHROME
  Render in monochrome mode (all entities are drawn in black)
- MRK_RENDEROMODEPRESERVECLIP
  Prevents the markup control from resetting the DC's clip region prior to rendering its contents.
- MRK_RENDEROMODENOPALETTE

  Prevents the markup control from selecting its palette onto the DC prior to rendering its contents.
- MRK_RENDERMODEPIXELFONT

  Specifies that the entity font size is given in pixels and not in world coordinates.

MRK_NotePrint:

Contains information for printing a single or multiple notes. Use by MRK_BATCHPRINTNOTES and MRK_NOTEPREPAREPRINT.

```
typedef struct {
        int         numNotes;       // Number of notes to print.
        LPMRKENTHANDLE    pHandles;  // Handles of            //notes to print.
        DWORD           dwFlags;          // Print flags:
                                          //MRK_NOPAGEBREAK
                                          // set => no page break
                // between notes.
        struct {
                WORD            units;    // in: units one of
                                          // PAN_CTLUNIT*
                double          top;      // in: Top margin
                double          left;     // in : Left margin
                double          bottom;   // in : bottom margin
                double          right     // in : right margin
        } margins;
        LOGFONT                 font;     // in: title font.
        PRINTDLG        FAR     *printDlg;        // int: common
                                                  //dialog options
        int                     numPages;         // out: pagecount.
        LONGRECT                pageRect;         // out: page rect //
                                                  //in twips.
        RECT                    deviceRect;       // out:
                                          //corresponding device
                        // rect in pixels
} MRK_NotePrint, FAR *LPMRK_NotePrint;
```

MRK_NoteRender:

Contains information for rendering a single note page. Used by MRK_NOTERENDERPAGE.

```
typedef struct {
        int             pageIndex;        // in: pageIndex
        HDC             hdc;              // in: Render DC. LONGRECT      pageRect;         // in:
page rect in twips.
        RECT            deviceRect;       // int: corresponding device
                // rect in pixels
} MRK_NoteRender, FAR *LPMRK_NoteRender;
```

MRK_DrawInfo:

Contains general drawing information. Used as an argument to MRK_RENDERONTODC message and the built in entity draw functions MrkDrawXXX():

```
typedef struct  {
        COLORREF      LineColor;        // Line color: -1 to use
                                        //entity line color
        COLORREF      FillColor;        // Fill color: -1 to use
                                        // entity fill color
        int           FillType;         // Fill type: -1 to use entity
                                        // fill type
        int           PenStyle;         // Pen style: -1 to use
                                        // entity pen style
        int           PenWidth;         // Pen width in pixels: -1
                // to use entity
                                        // penwidth
        int           StartArrow;       // Start arrow: -1 to use
                                        // entity start arrow
        int           EndArrow;         // End arrow: -1 to use
                                        // entity end arrow
}  MRK_DrawInfo, FAR *LPMRK_DrawInfo;
```

MRK_BaseInfo:

Contains information about the base file. Used by MRK_SETBASEINFO and MRK_GETBASEINFO messages.

```
typedef struct {
        int           Type;             // PAN_FileType.
        PAN_Point     Offset;           // Base offset.
        PAN_Point     Scale;            // Base scale.
        PAN_Point     Dpi;              // Base DPI.
        WORD          Units;            // Base units (PAN_UNITXXX)
        Real          Rotation;         // Base rotation.
        int           nView;            // Base view index.
} MRK_BaseInfo, FAR *LPMRK_BaseInfo;
```

MRK_GetEntities:

Contains information about which entities to get and a buffer to hold the handles of entities found. Used by MRK_GETENTITIES.

```
typedef struct {
        MRKENTHANDLE         Handle;  // Handle of entity whose
                //children
                                        // we want to get.
                                        // Used only if wParam is
                                        //set to
                                        // GET_ENTCHILDREN.
        int     Type;           // Type of entity or
                                //ENTTYPE_NULL to get all.
        in      nMaxSize;       // Maximum number handles the
                                // given buffer can hold.
        LPMRKENTHANDLE  pHandles; // Buffer to hold entity
                                        // handles.
} MRK_GetEntities, FAR * LPMRK_GetEntities;
```

MRK_EnumEntities:

Contains information about which entities to enumerate and the enumeration callback procedure. Used by MRK_ENUMENTITIES.

```
typedef struct {
        MRKENTHANDLE          Handle; // Handle of entity whose
                                      //children
                                      // we want to enumerate.
                                      // Used only if wParam is
                                      //set to
                                      // ENUM_ENTCHILDREN.
        int              Type;        // Type of entity or
                                      // ENTTYPE_NULL to
                                      // enumerate all.
        FARPROC          pCallbackFn; // Enumeration
                                      // Callback procedure.
        LPVOID           lpData;      // Application specific data to
                                      // be passed as arg. To the
                                      // callback procedure.
} MRK_EnumEntities, FAR * LPMRK_EnumEntities;
```

The Callback procedure is defined as follows:

```
int     EnumProc(LPMRK_EntitySpec pEntitySpec , LPVOID lpData);
```

MRK_CopyLayer:

Information used by MRK_COPYLAYER.

```
typedef struct {
        int     nSrcMO          // Source markup object.;
        int     nSrcLayer;      // Source layer
        int     nDestMO;        // Destination markup object
        int     nDestLayer;     // Destination layer
} MRK_CopyLayer, FAR *LPMRK_CopyLayer;
```

MRKN_CopyLayer:

Information used by MRKN_COPYLAYER.

```
typedef struct {
        MRK_CopyLayer copyStruct;       // MRK_COPYLAYER info
        MRK_EntitySpec entitySpec;      // Info of entity being copied
} MRKN_CopyLayer;
```

UDEINFO:

Contains information about a given UDE DLL. Used by MRK_GETUDEINFO.

```
typedef struct {
        UINT            entID;
        char            entName[_MAX_PATH];
        char            entFileName[_MAX_PATH];
        char            entMenuDesc[_MAX_PATH];
        char            entDescription[_MAX_PATH];
        char            entShortDescription[_MAX_PATH];
        UINT            BitmapResID;
        HCURSOR         hCursor;
        HINSTANCE       hinstDLL;
        DWORD       dwHints;        // Bit flags. Any or combination
                                    // of the following:
                                    // UDE_SUPPORTS2D ,
                                    // UDE_SUPPORTS3D, and
                                    // UDE_SUPPORTSNOFILL.
        DWORD       dwSnapSupport;  // Bit falgs. Any or combination
                                    // of the following:
                                    // MRK_SNAPTO_NONE,
                                    // MRK_SNAPTO_VERTEX,
                                    // MRK_SNAPTO_EDGE,
                                    // MRK_SNAPTO_MIDEGDE,
                                    //MRK_SNAPTO_ARCCENTER,
                                    // MRK_SNAPTO_FACE,
                                    // MRK_SNAPTO_ALL,

        void (PASCAL *WhoIAm)(void* Info);

        void (PASCAL *InitEntity)(LPMRK_EntitySpec Entity);

        void (PASCAL *ReleaseEntity)(LPMRK_EntitySpec Entity);

        long (PASCAL *ReadEntity)(LPMRK_EntitySpec Entity);

        long (PASCAL *WriteEntity)(LPMRK_EntitySpec Entity,
                                BOOL fSizeOnly);

        int  (PASCAL *MouseProc)(int Msg, WPARAM wParam,
                    LPARAM lParam, LPMRK_EntitySpec pEntitySpec);

        BOOL (PASCAL *GetControlPoints)(LPINT pNumPts,
                    LPPANPOINT *pPts, LPMRK_EntitySpec pEntitySpec);

        BOOL (PASCAL *DoEdit)(PAN_CtlRange *R1,
                    PAN_CtlRange *R2,       LPMRK_EntitySpec pEntitySpec);

        BOOL (PASCAL *Translate)(PAN_Point *Vector,
                    LPMRK_EntitySpec pEntitySpec);

        BOOL (PASCAL *DoCopy)(LPMRK_EntitySpec
                    pEntitySpec);

        BOOL (PASCAL *Draw)(MRK_RenderOptions *lpOptions,
        LPMRK_EntitySpec pEntitySpec);

        BOOL (PASCAL *SelectionTest)(PAN_CtlRange *selrange,
        LPMRK_EntitySpec pEntitSpec);

        BOOL (PASCAL *BoundingBox)(LPMRK_EntitySpec
                    pEntitySpec);

} UDEINFO, FAR *LPUDEINFO;
```

Feedback

Oracle products are designed according to your needs. We appreciate your feedback, comments or suggestions. Contact us by e-mail or telephone.

General Inquiries

Telephone:	+1.514.905.8400
E-mail:	autovuesales_ww@oracle.com
Web Site:	http://www.oracle.com/autovue/index.html

Sales Inquiries

Telephone:	+1.514.905.8400
E-mail:	autovuesales_ww@oracle.com

Customer Support

Web Site:	http://www.oracle.com/autovue/index.html