Voverlay SDK User Manual

For

Writing Software to Overlay PC-Generated Text, Graphics & Video on External TV & Video Signal

Version 1.2.1.0

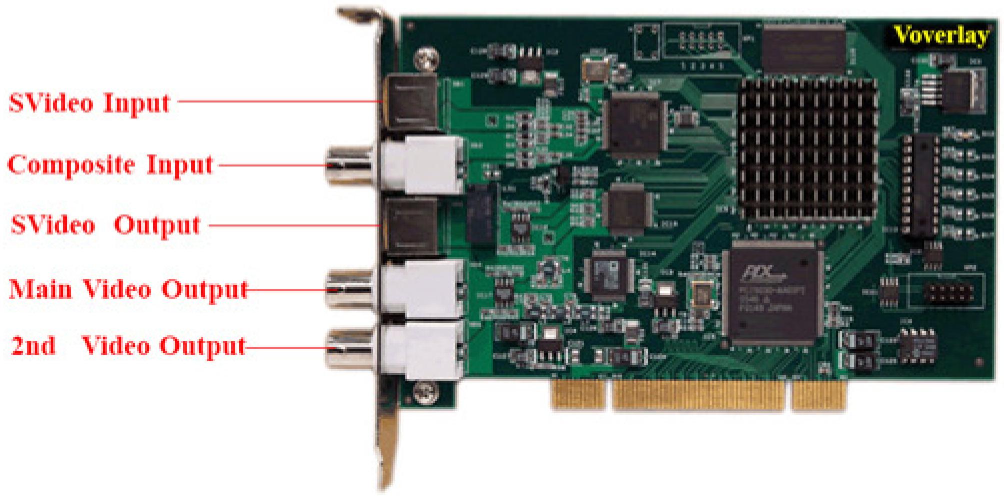


Table of Contents

1. Introduction ------------------ 2
2. The VOVERLAY Card Working Principle ------------------ 2
3. VOVERLAY.DLL & VOVERLAY SDK ------------------ 3
4. SDK Function List ------------------ 3
5. SDK Function Calling Sequence ------------------ 18
6. SDK Operation Requirement ------------------ 18
7. Sample Source Codes ------------------ 19

1. Introduction

VOVERLAY SDK (Software Development Kit) is for rapidly developing application software using **VOVERLAY** Text & Graphics Overlay PCI card. **VOVERLAY SDK** shields the software developers from coding complicated hardware interfacing by supplying highly integrated and extremely easy to use C-styled function calls. Using **VOVERLAY SDK**, even entry-level software developers can quickly write application software to control **VOVERLAY** cards on the PC to output PC-generated full-colour text, graphics and video onto external TV and VCR devices, using C++, VB, VC# or other programming languages.

The main functions of the **VOVERLAY** PCI card include realtime overlaying/superimposing superior-quality PC-generated, fully **alpha-blended** colour text, graphics and video onto external TV/VCR signal, and realtime-displaying these text, graphics & video on its multiple video output ports regardless the availability of any input video signal. **VOVERLAY** PCI card allows application software to directly read and write on-board 32-bit per pixel(8-bit each for RGB and Alpha channels) memory locations whose contents can be instantly mixed with the incoming video signal (when they exist), and output at the video output ports that can be connected to TV, VCR, DVD-recorder, camcorder, video capture card, etc. With such an easy-to-process memory-mapped graphical output mechanism, application software can utilise the full power of the MS Window's GDI (Graphics Device Interface) programming interface to generate un-limited graphical objects to display on external TV instantly, without any software conversion.

Multiple VOVERLAY cards (2~32) can be installed and programmed on the same PC to output different text/graphics/video contents onto multiple TVs and VCRs simultaneously.

2. The VOVERLAY Card Working Principle

Several important factors make it possible to deliver PC-generated text/graphics/video onto the video output ports of the VOVERLAY card:

2.1 Overlay Memory:

This is an on-board (the **VOVERLAY** card) memory area of 720 X 576 X 4 bytes that holds the PC generated graphical data (colour pixels) before they are displayed on the output video ports mixed with the input video signal. Every 4-byte of memory in this area represent an overlay colour pixel: the highest byte represents the "Alpha Channel"(See below), the second highest byte represents the Red colour, the third highest byte the Green colour, and the lowest byte represents the Blue colour. The 720 X 576 –Pixel is for displaying one frame of PAL-format video signal. When the input video signal is NTSC format, only 720X480X4 bytes of this overlay memory area are used, since NTSC video is digitized as 720X480-Pixel per frame.

2.2 Alpha Value:

As mentioned above, every pixel representing colour to be overlaid onto incoming video signal has an "Alpha" byte: this Alpha value controls how much visibility the overlaid graphical pixel has compared with the video signal pixel underneath it (at the same X/Y position) carried by the external video input: an Alpha value 255 means the PC-generated pixel is completely in front of the underneath external video pixel, while an Alpha value 0 means the PC-generated pixel is completely invisible (the external video signal pixel is fully exposed). Any Alpha value between 254 and 1 will see some degree of mixing PC-generated overlay pixel with external incoming video pixel: bigger Alpha value means more visibility for the PC-generated overlay pixel.

2.3 Video Input and Output Ports:

VOVERLAY card uses its Composite and SVideo video input ports to accept external video signal, mixing them in realtime with PC-generated text/graphics/video overlay data, then sending the mixed result to the video output ports(Main Composite and SVideo, Second Composite) immediately, thus displaying a Video + Overlay signal at any external TV etc devices connected at the output ports. Furthermore, each of the main and second

output ports can be independently configured as to output video only, overlay only, or video + overlay by a simple SDK function call.
To display one graphical pixel generated by PC onto a (X,Y) location on the video output frame of the **VOVERLAY** card, the application software needs to copy 4-byte colour data to the (X,Y) location of the Overlay Memory Area: the highest byte is the Alpha value, and next 3 bytes represent the RGB colour of the overlay pixel.

3. VOVERLAY.DLL & VOVERLAY SDK

The **VOVERLAY SDK** is based on a dynamic linking library **VOVERLAY.DLL,** which supports all the **VOVERLAY SDK** function calls the application software might use. **VOVERLAY.**DLL and some supporting library and files will need to be installed on the target PC running your application software controlling the **VOVERLAY** card.

4. VOVERLAY SDK Function List

-- char * **textol_GetSDKVer**(void);
Function: Return the version of the **VOVERLAY** SDK as a string constant, such as 1.2.1.0.

-- int **textol_GetCardNum** (void);
Function: Return the total number of installed **VOVERLAY** cards in the PC.
Note: **1.** This function depends on the properly installed device drivers of the **VOVERLAY** cards, not the physical cards themselves. So if one **VOVERLAY** card's driver is not installed properly then that card will not be counted as being present in the PC, even though that **VOVERLAY** card sits in a PCI slot.
2. The SDK supports maximum 32 **VOVERLAY** cards in one PC.
Return Value: the number of the **VOVERLAY** card installed in this PC, 0 means no card.

-- bool **textol_Initial**(UINT uCardNo, BOOL bClearScreen);
Function: Initialize one **Voverlay** card in the PC.
Parameters:
uCardNo: Card number, from 0 to 31
bClearScreen: True to clear all overlay output on all output ports during the init process
Return Value: True if successful.
Note: This function must be called first and return successfully before a **VOVERLAY** card can be used.
.

-- bool **textol_InitialEvent**(UINT uCardNo, LPCTSTR szEvent);
Function: Assign an event name to an initialized **Voverlay** card, Return non-zero for success.
Parameters:
uCardNo: Card number, from 0 to 31
szEvent: Event name, can be any string, usually the words "Global Event" + Card Number.
Return Value: True if successful
Note: At least one call to this function must follow the **textol_Initial()** call to the same card before any other calls can be used on that card.

-- bool **textol_Close**(UINT uCardNo);
Function: De-initialize a **Voverlay** card and release all resources allocated to it. Return true for success.
Parameters:
uCardNo: Card number, from 0 to 31
Note: Before exiting software this function must be called for every Voverlay card that has called **textol_Initial**, or memory leak will happen**.**
Return Value: True if successful

-- bool **textol_CloseIEvent**(UINT uCardNo, LPCTSTR szEvent);
Function: Close an event name that was created by calling **textol_InitialEvent**.
Parameters:
uCardNo: Card number, from 0 to 31
szEvent: Event name, must be the same as the contents used in **textol_InitialEvent.**
Return Value: True if successful

-- bool **textol_IsCardInited**(UINT uCardNo);
Function: Test if a card has been initiated properly.
Parameters:
uCardNo: Card number, from 0 to 31
Return Value: True if this card has been initialised

-- void **textol_RestoreDefault**(UINT uCardNo);
Function: Restore a card to its default hardware configuration.
Parameters:
uCardNo: Card number, from 0 to 31

-- bool **textol_LoadTextSingle**(UINT uCardNo, int textBkMode, int Alpha , int TRed, int TGreen, int TBlue, int AlphaBk, int BRed, int BGreen , int BBlue, char *fontName, int fontPoint , char *textString , int textX, int textY, bool transparent) ;
Function: Display one text string on output port
Parameters:
uCardNo: Card number, from 0 to 31
textBkMode: Text display background mode, must be either 1 for Transparent, or 2 for Opaque. This has the same meaning as the "*iBkMode*" parameter of the Windows GDI function "**SetBkMode".**
Alpha: The alpha value used to display the text on Voverlay card's output ports, valid values are from 0 (overlay text is fully hidden so if there is external video pixel behind the overlay pixel then the video is fully exposed) to 255(overlay text is fully in front of the video beneath it). Values between 1 and 254 represent different exposure degree of overlaid text in front of the underneath external video (if they exist): higher values means stronger overlay content.
TRed: The **Red** colour component for text, valid from 0 to 255
TGreen: The **Green** colour component for text, valid from 0 to 255
TBlue: The **Blue** colour component for text, valid from 0 to 255
AlphaBk: The alpha value used to display the background colour pixels (the spots where no

	stroke of the text is drawn) for the text on **Voverlay** card's output ports, valid values are from 0 (background is fully hidden) to 255 (background is fully in front of the video beneath it). Values between 1 and 254 represent different exposure degree of background colour in front of the external video behind(if they exist): higher values means stronger background colour pixels**.** The background Alpha as discussed here only has effects when the **textBkMode** parameter is **Opaque** and the **transparent** parameter is **false.**
BRed:	The **Red** colour component for the background pixels, valid from 0 to 255
BGreen:	The **Green** colour component for background pixels, valid from 0 to 255
BBlue:	The **Blue** colour component for background pixels, valid from 0 to 255
fontName:	The font name used to draw the text string, must be present in the current PC
fontPoint:	The font point used to draw the text string
textString:	The text content to be drawn, must be a null-terminated string.
textX:	The X position on the output video area to draw the text
textY:	The Y position on the output video area to draw the text
transparent:	Do not fill the empty spots in-between the text strokes with background colour (therefore exposing the video pixels underneath these empty spots)

Return Value: True if successful

Note: **1.** The "**output video area**" is also the video memory area on board the **Voverlay** card used to hold the pixels drawn by application software before they are displayed on the output video port: this area is of fixed size for different video signals currently being used at the input ports: 720X576-Pixels for PAL incoming video, and 720X480-Pixels for NTSC incoming video.
2. If a text string's end extends beyond the right-edge of the "output video area" (720-Pixel), the part of the text beyond the right edge will wrap around the screen to appear at the left-end of the video output area: on a TV screen they will appear at the start of the left end of the same line.

-- bool **textol_SetVideoBypass**(UINT uCardNo, ULONG videobypass);
Function: Set the main video output mode
Parameters:
uCardNo: Card number, from 0 to 31
videobypass: 0 – Video Only (No Overlay is displayed). If no video input then full black output.
1 – Video Overlay: Overlay contents is mixed with external video signal
2 – Overlay content only(No ext. video) without applying alpha (Frame Buffer)
3 – Overlay content only(No ext. video) also applying alpha (Graphics Alpha)
Return Value: True if successful
Note: The "**main video output**" are the SVideo and the first BNC/RCA(from top) video output ports

-- ULONG **textol_GetVideoBypass**(UINT uCardNo);
Function: Return the currently set main video output mode, as described in **textol_SetVideoBypass**
Parameters:
uCardNo: Card number, from 0 to 31
Return Value: The main video output mode value as described in **textol_SetVideoBypasss**

-- bool **textol_SetKeyOrPreviewOutput**(UINT uCardNo, ULONG OutputSelect);
Function: Set the second video output mode
Parameters:

uCardNo: Card number, from 0 to 31
OutputSelect: 0 : Display overlay content in black & white (key output)
1 : Display overlay content without applying alpha (preview output)
2: Display incoming video only without overlay content (video bypass)
3 : Display incoming video mixed with overlay contents
4 : Similar as 1 (CG buffer)
5 : Display overlay content but also applying alpha (CG buffer alpha)

Return Value: True if successful
Note: The "**second video output**" is the bottom BNC/RCA(from top) video output port on the card

-- ULONG **textol_GetKeyOrPreviewOutput**(UINT uCardNo);
Function: Return the current second video output mode as described in **textol_SetKeyOrPreviewOutput**
Parameters:
uCardNo: Card number, from 0 to 31
Return Value: The second video output mode value as described in **textol_SetKeyOrPreviewOutput**

-- bool **textol_ClearCurrentScreen**(UINT uCardNo = 0);
Function: Wipe out all overlay contents on all output ports
Parameters:
uCardNo: Card number, from 0 to 31
Return Value: True if successful
Note: After wiping out overlay contents, if there is incoming video signal on the video input ports, the video output ports will show all the incoming video signal. If there is no incoming video signal on the video input ports, the video output ports will show total blackness.

-- void **textol_ClearArea**(UINT uCardNo, int x, int y, int width, int height);
Function: Wipe out overlay contents in the area (x, y, width, high) on all output ports
Parameters:
uCardNo: Card number, from 0 to 31
x: the clear area's upper left corner x co-ordinate: must be between 0, and 719
y: the clear area's upper left corner y co-ordinate: valid between 0, and 575(PAL) or 479(NTSC)
width: the clear area's width in pixels. x + width must be < 720
height: the clear area's height in pixels. y + height must be < 576(PAL) or < 480 (NTSC)

Note: After wiping out overlay contents in the clear area, if there is incoming video signal on the video input ports, the cleared area on video output ports will show the incoming video signal. If there is no incoming video signal on the video input ports, the cleared area on video output ports will show total blackness.

-- **void textol_ClearTextArea**(UINT uCardNo, char *fontName, int fontPoint ,
char *textString, int textX , int textY) ;
Function: Wipe out overlay contents in the area as big as occupied by textString using fontName and fontPoint on all output ports, from upper left corner (textX, textY)
Parameters:
uCardNo: Card number, from 0 to 31
fontName: The font name used to calculate the text string's width and height

fontPoint: The font point used to calculate the text string's width and height
textString: The text content used to calculate the area
textX: the clear area's upper left corner x co-ordinate: must be between 0, and 719
textY: the clear area's upper left corner y co-ordinate: valid between 0, and 575 or 478

-- bool **textol_GetTextWidthHeight**(UINT uCardNo, char *fontName , int fontPoint ,
char *textString , int *textWidth , int *textHeight) ;

Function: Retrieve the width and height of the overlay contents in the area as big as occupied by textString using fontName and fontPoint

Parameters:

uCardNo: Card number, from 0 to 31
fontName: The font name used to calculate the text string's width and height
fontPoint: The font point used to calculate the text string's width and height
textString: The text content used to calculate the area
textWidth: hold the returned area's width, must point to an integer
textHeight: hold the returned area's height, must point to an integer

Return Value: True if successfully retrieved the width and height value

-- void **textol_GetOverlayContent**(UINT uCardNo, ULONG *mem , int x, int y, int width, int height);

Function: Retrieve the overlay contents in the area (x, y, width, height)

Parameters:

uCardNo: Card number, from 0 to 31
mem: the buffer to hold the returned overlay contents, must be at least width * height * 4 bytes
x: the area's upper left corner x co-ordinate: must be between 0, and 719
y: the area's upper left corner y co-ordinate: valid between 0, and 575(PAL) or 479(NTSC)
width: the area's width in pixels. x + width must be < 720
height: the area's height in pixels. y + height must be < 576(PAL) or < 480 (NTSC)

Note: **1**.**mem** must have enough space(width * height * 4 Bytes) to hold the returned content or this function will crash!
2. If x= 0, y = 0 and width =0 and height =0 then return the entire overlay content according to the current TV format (PAL 576 lines, NTSC 480 lines vertically). In this case the **mem** must be at least 720X576X4 bytes for PAL or 720X480X4 bytes for NTSC incoming video.

-- bool **textol_SetVideoMode**(UINT uCardNo, ULONG mode);

Function: Set the incoming video mode (PAL or NTSC) at the currently selected input video port

Parameters:

uCardNo: Card number, from 0 to 31
mode: 0 for NTSC, 1 for PAL

Return Value: true for successful

-- ULONG **textol_GetVideoMode**(UINT uCardNo);

Function: Retrieve the currently set incoming video mode (PAL or NTSC)

Parameters:

uCardNo: Card number, from 0 to 31

Return Value: 0 for NTSC, 1 for PAL
Note: This function returns the value set by **textol_GetVideoMode**, not the actual video mode.
i.e., this function does not auto-detect which video signal (PAL or NTSC) is at the input port.

-- bool **textol_SetCmpsOrSvideo**(UINT uCardNo , ULONG mode)；
Function: Set the input video port (SVideo or BNC/RCA/Composite socket)
Parameters:
uCardNo: Card number, from 0 to 31
mode: 0 for BNC/RCA/Composite socket, 1 for SVideo socket
Return Value: true for successful

-- ULONG **textol_GetCmpsOrSvideo**(UINT uCardNo)；
Function: Retrieve the currently selected input video port as set by **textol_SetCmpsOrSvideo**
Parameters:
uCardNo: Card number, from 0 to 31
Return Value: 0 for BNC/RCA/Composite socket, 1 for SVideo socket

-- bool **textol_SetTVORVTR**(UINT uCardNo , ULONG mode);
Function: Set the input device type at the currently selected input video port
Parameters:
uCardNo: Card number, from 0 to 31
mode: 0 for VTR, 1 for TV
Return Value: true for successful

-- ULONG **textol_GetTVORVTR**(UINT uCardNo);
Function: Get the input device type at the currently selected input video port
Parameters:
uCardNo: Card number, from 0 to 31
Return Value: 0 for VTR, 1 for TV

-- bool **textol_SetContrast** (UINT uCardNo , ULONG contrast);
Function: Set the video contrast at the input and output ports
Parameters:
uCardNo: Card number, from 0 to 31
contrast: 0 ~ 255, default is 128
Return Value: true for successful
Note: Default contrast is 128

-- ULONG **textol_GetContrast** (UINT uCardNo);
Function: Get video contrast at the input and output ports
Parameters:
uCardNo: Card number, from 0 to 31
Return Value: 0 ~ 255

-- bool **textol_SetBrightness** (UINT uCardNo , ULONG brightness);
Function: Set the video brightness at the input and output ports
Parameters:
  uCardNo: Card number, from 0 to 31
  brightness: 0 ~ 255, default is 128
Return Value: true for successful
Note: Default brightness is 128

-- ULONG **textol_Get Brightness** (UINT uCardNo);
Function: Get video brightness at the input and output ports
Parameters:
  uCardNo: Card number, from 0 to 31
Return Value: 0 ~ 255

-- bool **textol_SetHue** (UINT uCardNo , ULONG hue);
Function: Set the video brightness at the input and output ports
Parameters:
  uCardNo: Card number, from 0 to 31
  hue: 0 ~ 255, default is 128
Return Value: true for successful
Note: Default hue is 128

-- ULONG **textol_GetHue**(UINT uCardNo);
Function: Get video brightness at the input and output ports
Parameters:
  uCardNo: Card number, from 0 to 31
Return Value: 0 ~ 255

-- bool **textol_SetSaturation** (UINT uCardNo , ULONG saturation);
Function: Set the video saturation at the input and output ports
Parameters:
  uCardNo: Card number, from 0 to 31
  saturation: 0 ~ 255, default is 128
Return Value: true for successful
Note: Default saturation is 128

-- ULONG **textol_GetSaturation** (UINT uCardNo);
Function: Get video saturation at the input and output ports
Parameters:
  uCardNo: Card number, from 0 to 31
Return Value: 0 ~ 255

-- ULONG **textol_GetVideoInputStatus**(UINT uCardNo);
Function: Test if there is video signal at the currently selected video input socket (SVideo/BNC port)
Parameters:
    uCardNo: Card number, from 0 to 31
Return Value: Non-Zero if the socket has video signal, zero if no signal

-- bool **textol_SetVideoBlackLevel** (UINT uCardNo , ULONG blacklevel);
Function: Set the NTSC video black level
Parameters:
    uCardNo: Card number, from 0 to 31
    blacklevel: 0 for 0 IRE, 1 for 7.5 IRE
Return Value: true for successful
Note: Only valid when incoming video is NTSC

-- ULONG **textol_GetVideoBlackLevel** (UINT uCardNo);
Function: Get the current NTSC video black level
Parameters:
    uCardNo: Card number, from 0 to 31
Return Value: 0 for 0 IRE, 1 for 7.5 IRE, -1 for failure
Note: Only valid when incoming video is NTSC

-- bool **textol_SetVideoSample**(UINT uCardNo);
Function: Start retrieving video sample (the still image of the video frame at the input port) process
Parameters:
    uCardNo: Card number, from 0 to 31
Return Value: true for successful
Note: After successfully calling this function, poll **textol_GetIfVideoSampleDone** to test if the video sample is ready for process, then call **textol_GetVideoSampleData** to retrieve the actual sample data (the pixel values in the captured frame)

-- ULONG **textol_GetIfVideoSampleDone**(UINT uCardNo);
Function: Test if the sample data retrieving process started by **textol_SetVideoSample** has finished
Parameters:
    uCardNo: Card number, from 0 to 31
Return Value: Non 0 if the retrieving process is finished so **textol_GetVideoSampleData** can be called
Note: Only after this function returns non-zero then **textol_GetVideoSampleData** can be called

-- bool **textol_GetVideoSampleData**(UINT uCardNo, ULONG* value);
Function: Retrieve video sample data into buffer pointed to by “value”
Parameters:
    uCardNo: Card number, from 0 to 31
    value: Buffer to hold the sample data, must be >= 4X720X576(PAL)/4X720X480(NTSC) bytes
Return Value: true for successful
Note: before calling this function, **textol_GetIfVideoSampleDone** must return true.

-- bool **textol_SetSyncMode**(UINT uCardNo, ULONG mode);
Function: Set Sync mode (the clock used to generate timing)
Parameters:
    uCardNo: Card number, from 0 to 31
    mode: 0 – use internal sync (the clock on-board Voverlay card)
        1 – use external sync (the clock embedded in the input video signal)
Return Value: true for successful
Note: When there is no incoming video signal, use internal sync, otherwise use external sync.

-- ULONG **textol_GetSyncMode**(UINT uCardNo);
Function: Get Sync mode (the clock used to generate timing)
Parameters:
    uCardNo: Card number, from 0 to 31
Return Value: 0 –internal sync(clock on Voverlay card), 1 –external sync(clock in incoming video)

-- ULONG **textol_StartTimer**(UINT uCardNo , int timeBkMode ,

```
                    int Alpha ,     int TRed ,  int TGreen , int TBlue ,
                    int AlphaBk ,  int BRed ,  int BGreen, int BBlue ,
                    char *fontName, int fontPoint, int timeX, int timeY,
                    bool transparent ,
                    bool clearPrevTimeDisplay,
                    bool displayDate = true,
                    int displayMS_FN = 1,
                    unsigned int timerInterval);
```

Function: Start to display timer as overlay
Parameters:
    uCardNo: Card number, from 0 to 31

    timeBkMode: same as described for **textol_LoadTextSingle**
    Alpha , **TRed , TGreen , TBlue ,**
    AlphaBk , **BRed , BGreen, BBlue ,**
    ***fontName,** **fontPoint, timeX, timeY,**
    transparent : These parameters are exactly the same as described for **textol_LoadTextSingle**

    clearPrevTimeDisplay: if true, then the previously started time display will be wiped out when new timer defined by this call starts
    displayDate: True to display date with time
    displayMS_FN: 1 = display mille-seconds following the second's position,
        2 = display Frame Number following the second's position
        0 = display neither of them
    timerInterval: The interval to display time in mille-seconds, must be >= 10

Return Value: If succeeds, return the timer ID(non-zero) that can be passed to Windows SDK's KillTimer function**.**
If fails, return zero.

Note: 1. To create a transparent time display on live video (only time ticks without any background Colour displayed), use Black Background((BRed, BGreen, BBlue) = (0,0,0)), transparent = false, AlphaBk = -1, and timeBkMode = 2 (OPAQUE)
2. To create a time display with a half-transparent background on a live video (the background is some colour half-transparent on top of the video), use transparent = false, AlphaBk = the desired background colour (left-most/highest byte must be zero), and timeBkMode = 2 (OPAQUE)
3. Each Voverlay card can have at most one timer display at any time

-- void **textol_StopTimer**(UINT uCardNo, bool clearPrevTimeDisplay);
Function: Stop to display timer as started by **textol_StartTimer** on card uCardNo
Parameters:
uCardNo: Card number, from 0 to 31
clearPrevTimeDisplay: true to wipe out the current time display

-- bool **textol_IsTimerOn**(UINT uCardNo);
Function: If card uCardNo is displaying time
Parameters:
uCardNo: Card number, from 0 to 31
Return Value: True if this card is displaying time (has successfully called **textol_StartTimer)**

-- bool **textol_LoadTargaImageFile**(UINT uCardNo, unsigned char *imageFileName,
int Alpha, int putImageX, int putImageY);
Function: Display a Targa (.tga) graphics file's contents on all output ports
Parameters:
uCardNo: Card number, from 0 to 31
imageFileName: the Targa file's full path and name inc. disk drive letter
Alpha: Alpha blending value for displaying (0~255), bigger value means stronger overlay display
putImageX: the horizontal location in pixels of the starting position to display the graphics
putImageY: the vertical location in pixels of the starting position to display the graphics
Return Value: True if the loading and displaying succeed.

-- bool **textol_LoadImageFile**(UINT uCardNo, char *imageFileName,
int Alpha, int putImageX, int putImageY, int putImageWidth, int putImageHeight,
bool transparent , ULONG transparencyKey,
DWORD rop, ULONG TKErrorRange, bool clearOldOverlay) ;
Function: Display a BMP/JPG/GIF/PNG/TIF graphics file's contents on all output ports
Parameters:
uCardNo: Card number, from 0 to 31
imageFileName: the graphics file's full path and name inc. disk drive letter
Alpha: Alpha blending value for displaying (0~255), bigger value means stronger overlay display
putImageX: the horizontal location in pixels of the starting position to display the graphics
putImageY: the vertical location in pixels of the starting position to display the graphics
putImageWidth: width of the displayed image, 0 means using the graphic file's original width
putImageHeight: height of the displayed image, 0 means using the graphic file's original height
transparent: true to hide (make invisible) the pixels whose colours and **transparencyKey** value have minimum difference, which means:
abs(Rp - Rt) + abs(Gp - Gt) + abs(Bp - Bt) <= **TKErrorRange**;

where abs(X) is the absolute value of X,
Rp/Gp/Bp is the RGB value of the pixel,
Rt/Gt/Bt is the RGB value of the **transparencyKey**(the lower 24 bits).

transparencyKey: used to calculate pixel colour difference if "**transparent**" is true, see above

TKErrorRange: see "**transparent**" and "**transparencyKey**" above, the default is zero

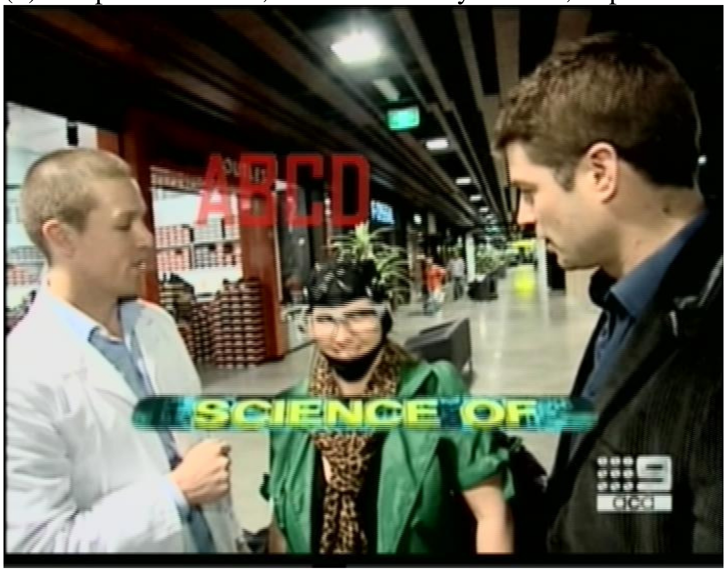
rop: same as the dwRop(Raster Operation Code) parameter of the BitBlt function in the MS Windows SDK: it defines how the graphics file's pixels are combined with the pixels previously being displayed on the same positions to achieve the final overlay result. The default value is **SRCCOPY**: copy the graphics pixel over to replace the original pixel.

clearOldOverlay: This value is only meaningful when parameter "**transparent**" is TRUE:

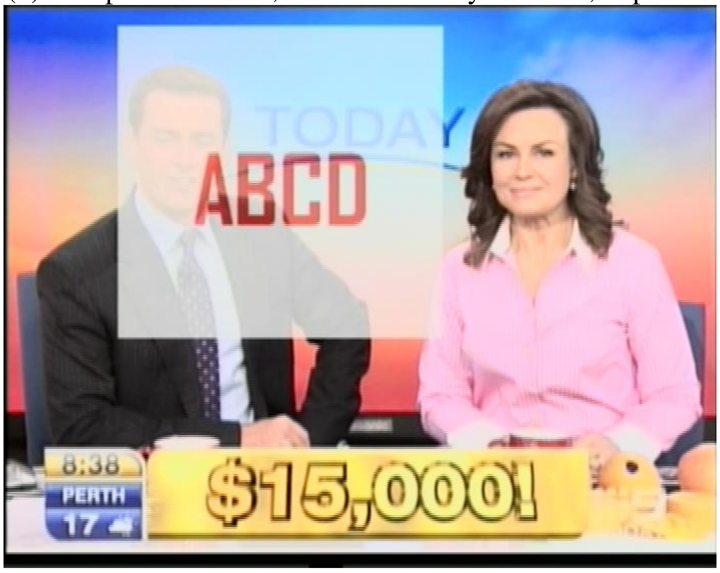
If **clearOldOverlay** is **TRUE**, then those pixels in the graphics file whose colour values and the "**transparencyKey**" colour have the minimum difference (as described in the "**transparent**" parameter) will become totally transparent, i.e., their alpha value will be set to zero.

If **clearOldOverlay** is **FALSE** (this is default), then those pixels in the graphics file whose colour values and the "**transparencyKey**" colour have minimum difference as the "**transparent**" parameter described above will combine (logical or) their old alpha value with the new "**Alpha**" parameter value passed by this function call, so that if "**Alpha**" is nonzero then some degree of overlay will appear on top of the video – this is useful for example to display an half-transparent background exposing some of the underneath video. Examples to use **clearOldOverlay** differently when the same red text "ABCD" in front of white background graphics file is used while **transparencyKey** is also white:

(1)transparent=TRUE, clearOldOverlay=TRUE, Alpha=128



(2) transparent=TRUE, clearOldOverlay=FALSE, Alpha=128



Return Value: True if the loading and displaying succeed.

Note: **1.** If a graphics file's content has a dimension (width by height) larger than the current overlay memory area (720X576-Pixel for PAL, 720X480-Pixel for NTSC), it is better to use third-party image processing software such as MS Paint, Adobe PhotoShop etc to create a shrunk image file before calling this function to display the graphics as overlay, because the built-in graphics shrinking mechanism often create a file with distorted colour.
2. The type of graphics file is determined by its file extension: .bmp for BITMAP, .gif for GIF, .JPG for JPEG, .png for PNG, and .tif for TIFF. Animated GIF is not supported.

3. The 4-byte long "**transparencyKey**" value represents RGB colour, with the highest byte un-used, the Red byte at the second highest position (23~16 bit), the Green byte the next highest position (15~8 bit), and the Blue byte at the lowest position (7~0 bit). Note this arrangement of RGB colour components is different from the COLORREF value used in Windows SDK, where the Red byte is at the lowest bit position.
4. The Raster Operation Code (rop) determines the combination of pixels from the graphics file and from the previously drawn overlay pixels on the same position, un-related with the input video's pixels on the same positions.
5. When **transparent** is true, if **TKErrorRange** is zero, pixels from the loaded file whose colours equal to **transparencyKey** value will not be loaded on top of the live video thus exposing the video pixel beneath them; if **TKErrorRange** is bigger than zero, then pixels from the loaded file whose colours are similar to **transparencyKey** value will not be loaded on top of the live video ---- this will be useful when trying to achieve a "Blue-screen" effect from an image file whose "Key-colour" area contains non-uniform colours, e.g. a blue background where some pixels are similar to blue but not having the exactly pure blue RGB value (0, 0, 255).
6. To achieve clear "blue screen" effect when overlaying a graphics file with some smooth background colour onto live video, so that the background area will become completely transparent, set both parameters "**transparent"** and "**clearOldOverlay**" to TRUE.

-- void **textol_SetAreaAlphaColour**(UINT uCardNo, int Alpha , int x, int y, int width, int height,
int Red , int Green, int Blue,
bool transparent, ULONG transparencyKey, ULONG TKErrorRange);

Function: Change the alpha and/or colour values of an overlay memory area
Parameters:
uCardNo: Card number, from 0 to 31
Alpha: New alpha blending value for the overlay area, valid 0~255
x: the X-co-ordinate of the upper-left corner of the overlay area
y: the Y-co-ordinate of the upper-left corner of the overlay area
width: the width in pixels of the overlay area
height: the height in pixels of the overlay area
Red: the Red colour byte of the new colour for all the pixels within the overlay area
Green: the Green colour byte of the new colour for all the pixels within the overlay area
Blue: the Blue colour byte of the new colour for all the pixels within the overlay area
transparent: If true, do not change the Colour of those pixels whose current overlay colours and the "**transparencyKey**" colour have minimum difference, which means:
abs(Rp - Rt) + abs(Gp - Gt) + abs(Bp - Bt) <= **TKErrorRange**;
where abs(X) is the absolute value of X,
Rp/Gp/Bp is the RGB value of the pixel,
Rt/Gt/Bt is the RGB value of the **transparencyKey**(the lower 24 bits).

If false, change all pixels' alpha and/or colour

transparencyKey: see "**transparent**" above, the colour used to calculate the minimum difference
TKErrorRange: see "**transparent**" above, default is zero.

Note: 1. To set the Alpha value without changing the colours of the overlay area, pass -1 to the Red, Green and Blue parameters
2. When **transparent** is true, if **TKErrorRange** is 0**,** pixels that have their colour values equal to **transparencyKey** will remain un-changed; if **TKErrorRange** is > 0**,** pixels that have their colour values similar to **transparencyKey** (including equal to) will remain un-changed.

-- void **textol_GetAreaAlphaColour**(UINT uCardNo, int x, int y, int width, int height, ULONG *buffer)

Function: Retrieve the alpha and colour values of an overlay memory area

Parameters:

uCardNo: Card number, from 0 to 31
x: the X-co-ordinate of the upper-left corner of the overlay area
y: the Y-co-ordinate of the upper-left corner of the overlay area
width: the width in pixels of the overlay area
height: the height in pixels of the overlay area
buffer: buffer to hold the retrieved data, must be at least width X height X 4 bytes

Note: Every 4-Byte retrieved in "buffer" represent the Alpha (the highest byte), Red (the next highest byte), the Green (the third highest byte) and the Blue (the lowest byte) value of one pixel

-- void **textol_MoveArea**(UINT uCardNo, int Alpha, int sx, int sy, int width , int height, int dx, int dy, bool transparent, bool noErase);

Function: Copy the overlay contents from source location to destination location with alpha blending

Parameters:

uCardNo: Card number, from 0 to 31
Alpha: alpha blending value used to display the source pixels at destination location. valid 0~255
x: the X-co-ordinate of the upper-left corner of the source overlay area
y: the Y-co-ordinate of the upper-left corner of the source overlay area
width: the width in pixels of the source overlay area
height: the height in pixels of the source overlay area
dx: the X-co-ordinate of the upper-left corner of the destination overlay area
dy: the Y-co-ordinate of the upper-left corner of the destination overlay area
transparent: If set true, and the source pixel is empty(no overlay content, i.e. black overlay colour), the destination pixel will also become empty (fully exposing the underneath input video content), regardless the **Alpha** value. If set false, then the **Alpha** value will be used to blend with the empty overlay(black colour) onto the destination location: this will show some degree of black colour on a non-black video background.
noErase: If true, the source area overlay contents remain unchanged, if false they are erased.

-- bool **textol_LoadBitmapPixels**(UINT uCardNo, int Alpha, bool transparent , ULONG transparencyKeyColour, char *pixels , int bytesPerPixel, bool top_down , int x, int y, int width, int height);

Function: Display a graphics bitmap's pixels at some overlay memory area

Parameters:

uCardNo: Card number, from 0 to 31
Alpha: If **bytesPerPixel** is 3, this is the alpha blending value used to display all the bitmap pixels at the overlay memory area. This value is ignored if **bytesPerPixel** is 4, since then each graphics pixel will use its own alpha byte as alpha blending value.
transparent: true to make the pixels with the **transparencyKey** parameter's value invisible
transparencyKey: if "**transparent**" is true, bitmap pixels with this colour will not be displayed
pixels: memory buffer holding the bitmap's pixels, see **Note 2** below for its format details
bytesPerPixel: either 3 or 4, number of bytes per graphics pixel in the "**pixels**" buffer
top_down: if the scan-lines in "pixels" are arranged as 1st-line at the lower memory address
x: the X-co-ordinate of the upper-left corner of the overlay area to display the graphics bitmap
y: the Y-co-ordinate of the upper-left corner of the overlay area to display the graphics bitmap
width: the width in pixels of the graphics bitmap pointed to by "**pixels**"
height: the height in lines of the graphics bitmap pointed to by "**pixels**"

Return Value: True if succeeded

Notes: **1.** The "scan-lines" of the bitmap pixels are counted from the upper-left corner as line 1, line 2… towards the bottom of the bitmap. If the "**top_down**" parameter is true, the pixels representing each scan-line will be arranged in buffer "**pixels**" from low memory addresses to high memory addresses as scan-line1, scan-line2, … scan-lineN, where "N" is the total number of pixel lines in the bitmap. If the "**top_down**" parameter is false, the lowest memory addresses in the "**pixels**" memory area will hold the scan-line**N,** then the next lowest addresses will hold scan-line(**N-1**), …, and the highest "**pixels**" memory addresses will hold scan-line1.

2. "**pixels**" contains 3 or 4 bytes (according to **bytesPerPixel**) per graphics pixel in the format of: Blue-Byte, Green-Byte, Red-Byte, and Alpha-Byte(if **bytesPerPixel** is 4), i.e., the Alpha/Red byte is at the highest memory address while the blue-byte is at the lowest memory address.

-- bool **textol_LoadImageFromWindow**(UINT uCardNo , HWND wnd , int Alpha,
int putImageX, int putImageY, int putImageWidth, int putImageHeight, DWORD rop ,
bool ClientAreaOnly,
bool transparent, ULONG transparencyKey,
ULONG repeatTimes, ULONG pauseMS, ULONG TKErrorRange,
boolclearOldOverlay) ;

Function: Display a Window's image at some overlay memory area

Parameters:

uCardNo: Card number, from 0 to 31
wnd: the handle of the window whose image is to be displayed
Alpha: alpha blending value used to display the window's image at the overlay memory area.
putImageX: the X-co-ordinate of the upper-left corner of the overlay area to display the window
putImageY: the Y-co-ordinate of the upper-left corner of the overlay area to display the window
putImageWidth: the width in pixels of the overlay area to display the window's image
putImageHeight: the height in pixels of the overlay area to display the window's image
rop: Raster Operation Code to combine the window's pixels with overlay pixels already there
ClientAreaOnly: true to only display the image of client area of the window
transparent: true to not displaying the window image pixels satisfying the following conditions:

abs(Rp - Rt) + abs(Gp - Gt) + abs(Bp - Bt) <= **TKErrorRange**;
where abs(X) is the absolute value of X,
Rp/Gp/Bp is the RGB value of the window image pixel,
Rt/Gt/Bt is the RGB value of the **transparencyKey** (the lower 24 bits).

transparencyKey: if "**transparent**" is true, used to calculate the colour difference as above
TKErrorRange: see "**transparent**" and "**transparencyKey"** above, default to zero
repeatTimes: How many times to display the window image before ending this function
pauseMS: In mille-seconds, the pause period in-between displaying the image repeatedly
clearOldOverlay: This value is only meaningful when parameter "**transparent**" is TRUE:

If **clearOldOverlay** is **TRUE**, then those pixels in the window whose colour values and the "**transparencyKey**" colour have the minimum difference (as described in the "**transparent**" parameter) will become totally transparent, i.e., their alpha value will be set to zero.
If **clearOldOverlay** is **FALSE** (this is default), then those pixels in the window whose colour values and the "**transparencyKey**" colour have minimum difference as the "**transparent**" parameter described above will combine (logical or) their old alpha value with the new "**Alpha**" parameter value passed by this function call, so that if "**Alpha**" is nonzero then some degree of overlay will appear on top of the video – this is useful for example to display an half-transparent background exposing some of the underneath video. See examples shown under the same **clearOldOverlay** parameter for function **textol_LoadImageFile.**

Return Value: true if succeeds.

Note: 1. This function is suitable to display a window that has static contents. To display a window that has constantly changing images such as an animation or video playback window, use function **textol_LoadImageFromWindowOnThread.**

-- HANDLE **textol_LoadImageFromWindowOnThread**(UINT uCardNo, HWND wnd , int Alpha, int putImageX, int putImageY, int putImageWidth, int putImageHeight, DWORD rop, bool ClientAreaOnly, bool transparent , ULONG transparencyKey , ULONG threadRunTime , ULONG threadPauseTime, int threadPriority, bool eraseOnExit, UINT exitMsg, HWND parentWnd, bool clearOldOverlay, ULONG TKErrorRange);

Function: Start a separate thread to continuously display a Window's image at overlay memory area

Parameters:

uCardNo: Card number, from 0 to 31

wnd: the handle of the window whose image is to be displayed

Alpha: alpha blending value used to display the window's image at the overlay memory area.

putImageX: the X-co-ordinate of the upper-left corner of the overlay area to display the window

putImageY: the Y-co-ordinate of the upper-left corner of the overlay area to display the window

putImageWidth: the width in pixels of the overlay area to display the window's image

putImageHeight: the height in pixels of the overlay area to display the window's image

rop: Raster Operation Code to combine the window's pixels with overlay pixels already there

ClientAreaOnly: true to only display the image of client area of the window

transparent: true to not displaying the window image pixels satisfying the following conditions:

abs(Rp - Rt) + abs(Gp - Gt) + abs(Bp - Bt) <= **TKErrorRange**;
where abs(X) is the absolute value of X,
Rp/Gp/Bp is the RGB value of the window image pixel,
Rt/Gt/Bt is the RGB value of the **transparencyKey** (the lower 24 bits).

transparencyKey: if "**transparent**" is true, used to calculate the colour difference as above

TKErrorRange: see "**transparent**" and "**transparencyKey"** above, default to zero

threadRunTime: In seconds, how long the thread will run.
If zero is supplied then the thread will run forever until being killed by calling **textol_StopImageFromWindowOnThread,** or application exits.

threadPauseTime: In mille-seconds, the thread pause time in-between updating the window's image. Longer pause time will use less CPU but might cause a rapidly changing window's image(such as video window) to appear less smooth.

threadPriority: thread running priority, normally "THREAD_PRIORITY_NORMAL" (0), but can also be other THREAD_PRIORITY_ values as defined in Windows SDK's winbase.h file.

eraseOnExit: True to erase the window's image when the thread exit.

exitMsg: Any WIN_USER or higher message sent to "parentWnd" when the thread ends

parentWnd: handle of the window to receive the **exitMsg** message when the thread ends

clearOldOverlay: This value is only meaningful when parameter "**transparent**" is TRUE:
If **clearOldOverlay** is **TRUE**, then those pixels in the window whose colour values and the "**transparencyKey**" colour have the minimum difference (as described in the "**transparent**" parameter)
will become totally transparent, i.e., their alpha value will be set to zero.
If **clearOldOverlay** is **FALSE** (this is default), then those pixels in the window whose colour values and the "**transparencyKey**" colour have minimum difference as the "**transparent**" parameter described above will combine (logical or) their old alpha value with the new "**Alpha**" parameter value passed by this function call, so that if "**Alpha**" is nonzero

then some degree of overlay will appear on top of the video – this is useful for example to display an half-transparent background exposing some of the underneath video. See examples shown under the same **clearOldOverlay** parameter for function **textol_LoadImageFile.**

Return Value: If succeeds, the thread handle (4-Byte long) that can be passed to function **textol_StopImageFromWindowOnThread** to stop the thread. Zero means failure.
Note: **1.** This function is useful to continuously display a content-changing window on overlay area, such as a video or animation playback window. To simply display a static window content, the function **textol_LoadImageFromWindow** can be used.

-- bool **textol_StopImageFromWindowOnThread**(UINT uCardNo, HANDLE hThread ,
bool eraseOnExit, ULONG waitTime = 0);
Function: Stop a thread started by calling **textol_LoadImageFromWindowOnThread**
Parameters:
uCardNo: Card number, from 0 to 31
hThread: the thread handle returned by calling **textol_LoadImageFromWindowOnThread**
eraseOnExit: true to erase the window's image when ending the thread
waitTime: In mille-seconds, time to wait before exiting this function(make sure the thread ends)
Return value: true if succeeds.

5. SDK Function Calling Sequences

(1). **textol_GetSDKVer**(void)
textol _GetCardNum (void):
textol_IsCardInited(UINT uCardNo);
these functions can be called anytime anywhere.

(2). **textol_Initial**(void):
textol_InitialEvent(UINT uCardNo, LPCTSTR szEvent);
must be called before all other functions on card number "uCardNo" can be used

(3). **textol_Close**(UINT uCardNo);
must be called before exiting software for every card that has been initialised

(4). **All other functions:** must be called between **textol_Initial / textol_InitialEvent** and **textol_Close.**

6. SDK Operation Requirement

6.1 To use **VOVERLAY** SDK functions to write application software, **VOVERLAY.DLL-related files** must be copied to the development PC's C:\Windows\System32 folder: these include all files under the "**lib**"and the "**inc**" folders on the **VOVERLAY** Setup CD.

To run the linked C++ program **Voverlay.exe** under the "**exe**" folder of the **VOVERLAY** Setup CD, **VOVERLAY** device driver must be installed from the folder "**driver**" on the Setup CD.

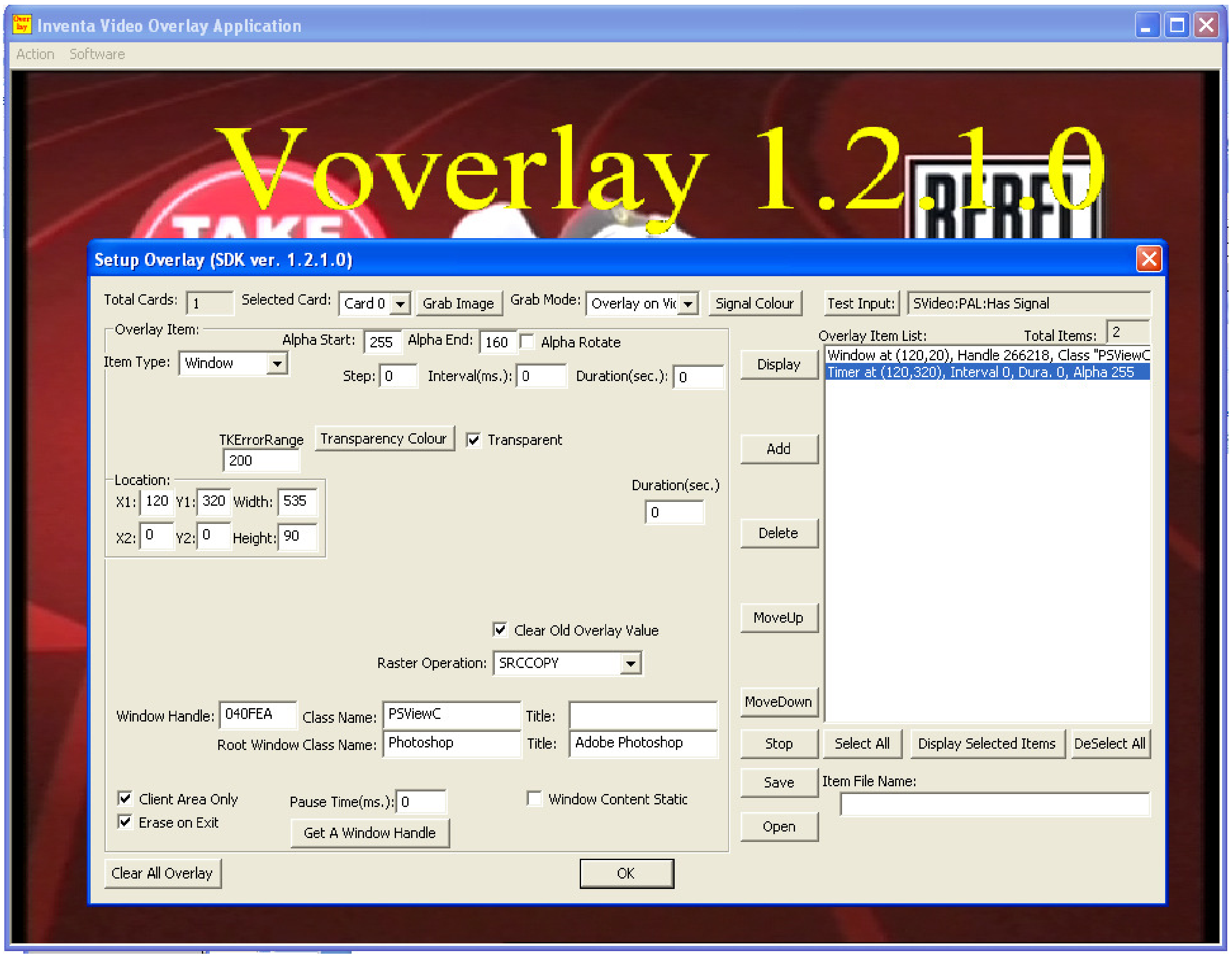
6.2 To prepare a target PC to run your application software written with **VOVERLAY** SDK, copy all the “*.dll” files from “**lib**” folder on the Setup CD to the target PC’s C:\Windows\System32 folder, install the **VOVERLAY** card’s device driver from the “**driver**” folder.

6.3 The function prototype declaration include file **VOVERLAY.H** needs to be used by C++ projects, while VisualBasic and VisualC# projects need to declare individual **VOVERLAY** functions one by one. See the relevant source codes under the “**src\C++**”, “**src\VB**” and **“src\CSharp”** folders on the Setup CD for actual source code and compiler project examples --- note the included VisualC# project created a Class Library called VoverlayAPI.Dll (under its sub-folder **ClassLibrary1**) and declared all Voverlay SDK functions inside this Class Library then called them as needed from its application software (CSharp.exe).

7. Sample Source Codes

The fully functioning MS Visual C++ Application software “**VOVERLAY.EXE”** version 1.2.1.0 with full source codes is included under the “**src\C++**” folder of the **VOVERLAY** Setup CD. A VisualBasic sample application with source code is included under the “**src\VB**” folder. A Visual C# sample application with source code is included under the “**src\CSharp**” folder. Note the C++ project was written with MS VisualStudio 2002 (VC++ 7), while the VB and Visual C# samples were written with MS VisualStudio 2008.

Screenshot for the **Voverlay.exe** application software:



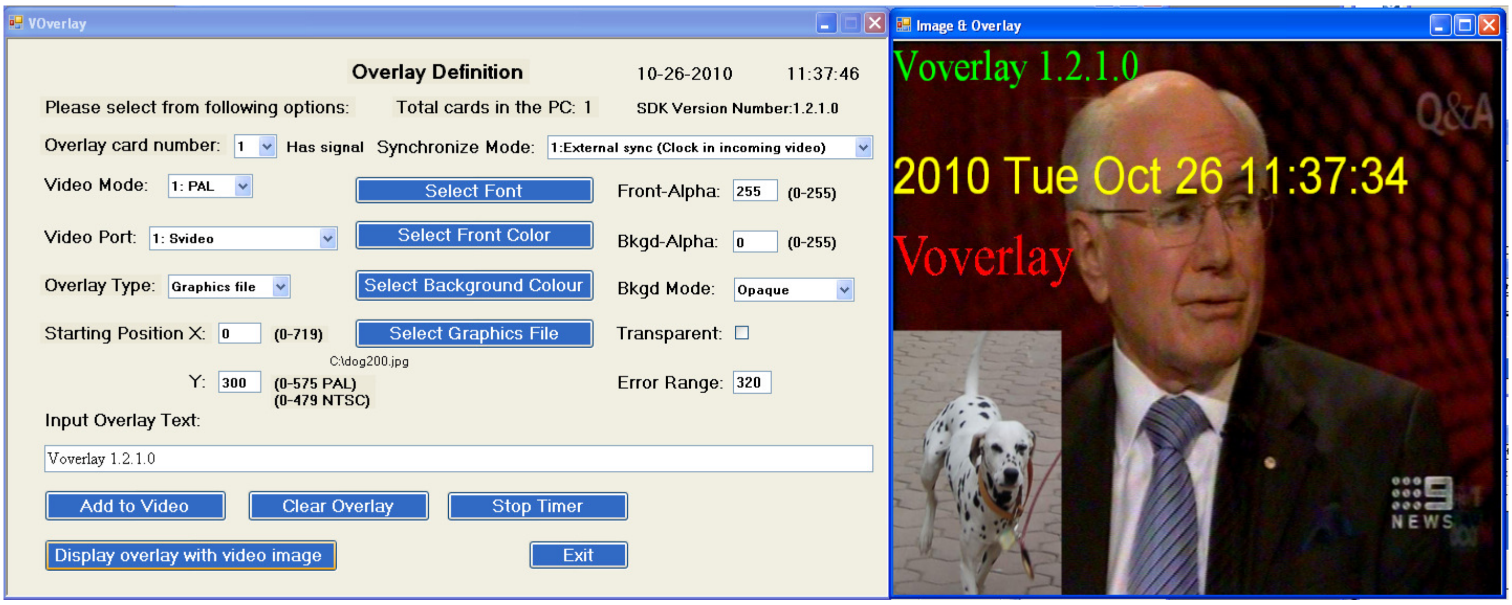
Total Cards: 1 Selected Card: Card 0 Grab Mode: Overlay on Vic Test Input: SVideo:PAL:Has Signal

Overlay Item:
Item Type: Window Alpha Start: 255 Alpha End: 160 ☐ Alpha Rotate
Step: 0 Interval(ms.): 0 Duration(sec.): 0
TKErrorRange 200 ☑ Transparent
Location:
X1: 120 Y1: 320 Width: 535
X2: 0 Y2: 0 Height: 90
Duration(sec.) 0
☑ Clear Old Overlay Value
Raster Operation: SRCCOPY
Window Handle: 040FEA Class Name: PSViewC Title:
Root Window Class Name: Photoshop Title: Adobe Photoshop
☑ Client Area Only
☑ Erase on Exit
Pause Time(ms.): 0
☐ Window Content Static

Overlay Item List: Total Items: 2
Window at (120,20), Handle 266218, Class "PSViewC
Timer at (120,320), Interval 0, Dura. 0, Alpha 255

Item File Name:

Screenshot for the **VoverlayVB.exe** application under the “**src\VB\bin\debug**” folder:



Overlay Definition 10-26-2010 11:37:46

Please select from following options: Total cards in the PC: 1 SDK Version Number:1.2.1.0

Overlay card number: 1 Has signal Synchronize Mode: 1:External sync (Clock in incoming video)

Video Mode: 1: PAL

Front-Alpha: 255 (0-255)

Video Port: 1: Svideo

Bkgd-Alpha: 0 (0-255)

Overlay Type: Graphics file

Bkgd Mode: Opaque

Starting Position X: 0 (0-719)

Transparent: ☐

C:\dog200.jpg

Y: 300 (0-575 PAL) (0-479 NTSC)

Error Range: 320

Input Overlay Text:

Voverlay 1.2.1.0