# GENERAL PROXY-DLL PACKAGE (GPP)

## INFO

GPP is intended to show 3rd party information in DirectX fullscreen mode games.
This is beta software for testing only. Be aware of the possibility of system crashes and data loss.
When testing, always reboot system after any abnormal termination of your game.

GPP does only work with games *using* DirectX8. As DX is downwards compatible, this would also work if you have a higher version of DX (e.g. DX9) installed. The game needs to be designed to use DirectX8, though.

## HISTORY

| | |
|---|---|
| 1/31/04 | V1.0: Initial release |
| 7/5/04 | V1.1: added initial startup logo (d3d8.dll), added ingame text line color, weight, size and font family. moved single function calls to one combined function. Added multiline feature. |
| 4/10/05 | V1.2: added “bold text” feature to multiline functions. Added basic ingame picture functions. |
| 4/30/05 | V1.3: added “screen size information” feature. Changed picture on startup. |

## FILES

d3d8.dll v1.3
Proxy-dll to be placed into the game's main exe directory. (DO NOT replace any d3d8.dll in your windows directories ! This will damage your DirectX config !).

gpcomms.dll v1.3
Communications wrapper for interaction of an application with the proxy-dll. Uses Microsoft MFC extensions (msvcp70.dll and msvcr70.dll need to be installed on your system).

gp_demo.exe v1.3
Demo application. Run before starting your game or do afterwards (in case you are able to alt-tab out). Set text to show ingame (single line or multiline), set position of text ingame (screen *ingame* dimension values). Select a bitmap file on disk to be shown ingame. Select background type of text (black/transparent). Show ingame.
Uses Microsoft MFC extensions (msvcp70.dll and msvcr70.dll need to be installed on your system).

GPHook.dll v1.4
Keyboard hook dll, to be used ingame. This is provided for convenience.

## CONTACT/AUTHORS

Download page: http://www.proxy.mikoweb.de
Discussions: http://www.boards.myrotacol.net
mail: gpp@mikoweb.de

Keyboard hooking: Mike “Mogar” Welch
GPP core: Michael “Miquant” Koch

# GPCOMMS.DLL EXPORTED FUNCTIONS

Single Text Line Functions

| | |
|---|---|
| **bool __stdcall GPSL_SetTextLineData(** | **WORD wTextPosX,**<br>**WORD wTextPosY,**<br>**LPCTSTR pTextLine,**<br>**DWORD dwTextColor,**<br>**bool bBlackBackground,**<br>**BYTE cSize, bool bTextBold,**<br>**BYTE cFontFamily)** |

One and only function to set up and change ingame single line text. It returns "true" on success, "false" otherwise.

**wTextPosX, wTextPosY:**
Ingame text position (upper left corner of textbox). Values are given in pixels and are based on the current screen resolution. Remember that WORD is a 16bit unsigned integer value !

**PTextLine:**
Text to show ingame. A pointer to a string. The string MUST be 0-terminated. Maximum length of string: 128 bytes (chars), including the terminating '\0'.

**DwTextColor:**
Color of the text. Format is like the DirectX D3DCOLOR value: ARGB (Alpha, Red, Green, Blue). 0xFF000000 would be black with no transparency.

**BBlackBackground:**
Defines how the background of the ingame text is handled. It can be transparent (bBlackBackground = false) or a black box (bBlackBackground = true).

**CSize:**
The size of the font in "logical device units". BYTE value.

**BTextBold:**
Sets the text to be drawn bold (bTextBold = true) or normal (bTextBold = false)

**CFontFamily:**
Two general font family specifiers are available. The system will (hopefully) select one out of the present system fonts. BYTE value. 0 = SWISS type (proportional font like Arial), 1 = MODERN type (monospace font like Courier New)

| |
|---|
| **bool __stdcall GPSL_ShowText(bool bShowIt)** |

Switches the ingame text on and off. Default is "off". Function returns "true" on success, "false" otherwise.

Multiple Text Line Functions

```
bool __stdcall GPML_SetTextMultilineData( WORD wPosX,
                                          WORD wPosY,
                                          LPCTSTR pText,
                                          DWORD dwTextColor,
                                          bool bBlackBackground,
                                          BYTE cFontSize, bool bTextBold,
                                          WORD wSizeX,
                                          WORD wSizeY,
                                          BYTE cFontFamily)
```

One and only function to set up and change ingame multiline text. It returns "true" on success, "false" otherwise.

**wPosX, wPosY:**
Ingame text position (upper left corner of textbox). Values are given in pixels and are based on the current screen resolution. Remember that WORD is a 16bit unsigned integer value !

**pText:**
Text lines to show ingame. A pointer to a string. The string MUST be 0-terminated. Maximum length of string: 1024 bytes (chars), including the terminating '\0'. Word wrap is enabled to show the text within the screen area given by the wSizeX/Y parameters (see below). <CR><LF> will show up as carriage return/line feed. A partially visible last line (not fitting completely in the bounding rectangle) will be cancelled.

**dwTextColor:**
Color of the text. Format is like the DirectX D3DCOLOR value: ARGB (Alpha, Red, Green, Blue). 0xFF000000 would be black with no transparency.

**bBlackBackground:**
Defines how the background of the ingame text is handled. It can be transparent (bBlackBackground = false) or a black box (bBlackBackground = true).

**cFontSize:**
The size of the font in "logical device units". BYTE value.

**BTextBold:**
Sets the text to be drawn bold (bTextBold = true) or normal (bTextBold = false)

**wSizeX, wSizeY:**
Size of the "bounding text box". All text will be shown within this box only. Values are given in pixels and are based on the current screen resolution. Remember that WORD is a 16bit unsigned integer value !

**cFontFamily:**
Two general font family specifiers are available. The system will (hopefully) select one out of the present system fonts. BYTE value. 0 = SWISS type (proportional font like Arial), 1 = MODERN type (monospace font like Courier New)

```
bool __stdcall GPML_ShowText(bool bShowIt)
```

Switches the ingame multiline text on and off. Default is "off". Function returns "true" on success, "false" otherwise.

Ingame Bitmap Functions

**bool __stdcall GPPIC_LoadNewPicture(LPCTSTR sPathToFile)**

Set a picture from disk to be loaded (and shown ingame later). Allowed formats: .bmp, .dds, .dib, .hdr, .jpg, .pfm, .png, .ppm, and .tga. There is no transparency.
Having a picture size (X*Y) with the powers of 2 is recommended for compatibility with some graphics cards (read: better use 256x256 pixels than e.g. 200x200).

**bool __stdcall GPPIC_ShowPicturePos(bool bShowIt, WORD wPosX, WORD wPosY)**

**bShowIt:**
Switches the ingame picture on and off. Default is "off". Function returns "true" on success, "false" otherwise.

**wPosX, wPosY:**
Ingame picture position (upper left corner of picture). Values are given in pixels and are based on the current screen resolution. Remember that WORD is a 16bit unsigned integer value !

**bool __stdcall GPSI_GetScreenSize(int& piScreenX, int& piScreenY)**

**piScreenX**, **piScreenY**
Retrieve the size of the screen in pixels. Note that this function does not work when the DirectX application is minimized. If the GPP can not determine the screen's size, both values are set to 0.

## GPHOOK.DLL EXPORTED FUNCTIONS AND USAGE

Functions

**bool __stdcall HookOn (HWND hAppWindow, bool bDebug)**

Sets the global keyboard hook. Call this on application startup. Sends a message to the window specified by hAppWindow (see below) whenever a key is pressed (or released). Provides additional debug output when bDebug = true.

**bool __stdcall HookOff (void)**

Disables the global keyboard hook. Call this on application termination.

Usage

An application (window) must provide a message handler to receive information from the global keyboard hook. This message handler must response to to a custom command number as seen here:

```
#define CM_COMMAND_KEY  (WM_USER + 0x1000)
```

The message handler routine is standard format (e.g., for MFC/Visual C++):

```
// ------------------------------------------------
BEGIN_MESSAGE_MAP(CMyMainDialog, CDialog)
      ON_MESSAGE(CM_COMMAND_KEY, OnMyHookKeyHandler)
END_MESSAGE_MAP()

LRESULT CMyMainDialog::OnMyHookKeyHandler(WPARAM wParam, LPARAM lParam)
...
// ------------------------------------------------
```

Filtering keys within this routine could look like this:

```
// ------------------------------------------------
if (wParam == VK_INSERT) // the insert key
    {
        if (GetKeyState(VK_INSERT)&(0x0001<<7)) // it is being PRESSED
        {
            // do weird things here
        }
    }
// ------------------------------------------------
```