GraphZ 1.1 Programmer's Reference

Author's E-Mail: cvzwynsv@estec.esa.nl

GraphZ is a dynamic link library (DLL) that curently implements one type of dynamically updated graph. This is a 2d-plot graph.

GraphZ is particularly suitable for displaying continuously incomming technical data in a simple way and at nearly real-time.

GraphZ is highly configurable, either from the program to which it is linked, or at run time, by the user. User's help can be found in the **User's Guide** help file. It is recommended that you read it before you start with programming matters.

GraphZ is a Dynamic Link Library (DLL).
It is not a Visual Basic Control (VBX), a static library (LIB), an object library (OBJ) or even a Pascal Unit (TPU).

GraphZ implements a global window class called `"GraphZ"`. This class remains valid as long as the library is loaded and as soon as you have called the `GZRegister` function.

In Microsoft Windows, each window has a given class. The class is identified by a name (ie. a string). The class is declared by the API *RegistereClass* function, and the class name is used for example as the first parameter of the API *CreateWindow* function.

A class can be either private (to an application) or global. In that case it is available to all the applications running at that time.
GraphZ implements one global class named `"GraphZ"`.

Real-time applications

GraphZ has been designed with real-time applications in mind.

Therefore it offers features like:

- It is a library so you can easily integrate it in your application. It is not a complete environment for off-line data analysis.
- It has been strongly optimized for speed matters.
- It displays incomming data provided by your application. Your application has the complete control on the data flow.
- It displays data in a continuous way, with a sliding window.
- It has full support for data & time scaling.
- It is small and implements only what you need with no overkill.

Still, there is **no real-time guarantee**. That is, GraphZ does not guarantees that a given transaction will be performed within a given time. Hence I advise the following if you have real-time requirements:

- Try to get a fast computer. In particular a math-coprocessor is very strongly recommended.
- Try not to be too short in memory. I believe 4M is a little bit short. But this depends a lot on the rest of your application. It also depends on how much concurrent graphs and curves you have and what is the curves buffer length.
- Another rather important speed factor seems to be the display card and/or driver. In that respect, making graph windows smaller may help significantly.
- To a smaller extent, the number of curves also has some influence.
- Make some tests about the execution speed with your application and your computer. For instance, if you have data comming in only every 30 seconds, then you probably won't have to worry about real-time aspects.
- In order to be able to guarantee the response time of your application's data handling routine, I recommend that you implement a first-in-first-out (FIFO) buffer mechanism between your data handling routine and the calls to GraphZ.

  Note: This is only if you:

  1. do have real-time requirements and have to guarantee the response time of your data handling routine; and
  2. have a high data flow rate which makes you work at the limit of your computer's capability so from time to time (not in average) GraphZ takes more time than is allowed by your requirements.

Buffer Length

Each curve has a buffer for storing historical data. The buffer length is a general setting of a graph. The default buffer length is 500 data elements. In the current version, one data element is 10 bytes long. The maximum length of the buffer is 64k.

See further details in the **User's Guide** help file.

Display card and/or driver.

I developped GraphZ on a 386, 25 Mhz, without coprocessor ! First I thought the bottleneck was the CPU-speed so I did a lot of work in optimizing the processings and in particular the floating point calculations. This of course helped a lot. Finally I observed that the bottleneck had become the speed of the low-level GDI graphics routines like *ScrollWindow* or *FillRect*.

Programming principle

In the following, I assume you are programming in **C**. If you actually program in **C++, Pascal or whatever**, then you will have to adapt this a little bit. Still, there should be no big effort involved.

The principle of using GraphZ in your application is quite simple. You have to proceed as follows:

- Register GraphZ. You will then receive a name and password.
- GraphZ needs the ControlZ run-time libraries (`CZxxxxx.dll`). THey must be located either in the same directory as GraphZ, or in the Windows (or Windows\system) directory. GraphZ also needs the `BWCC.DLL` library (Borland Windows Custom Controls).
- In either of the following cases:
  1 the GraphZ User's Guide help file (`"GraphZ.hlp"`) is located neither in your application's directory, nor in the windows (or windows\system) directory; or
  2 you have renamed this file; or
  3 you want to provide another kind of help file to your users;

  you will have to add the following lines to your WIN.INI file:

  ```
  [GraphZ]
  HelpFile=<the path and name of the GraphZ User's Guide help file>
  ```

- Include the following in your project's DEF file:

  ```
  IMPORTS
  GZRegister=GraphZ.1
  GZAddPoints=GraphZ.2
  GZGetSettings=GraphZ.3
  GZSetSettings=GraphZ.4
  GZWriteIni=GraphZ.5
  GZLoadIni=GraphZ.6
  GZSetIni=GraphZ.7
  GZMakeDate=GraphZ.8
  GZExtractTime=GraphZ.9
  GZExtractDate=GraphZ.10
  ```

  Note: Of course there are other ways to import these functions, but I find this rather convenient and portable. A popular alternative is to make an import linrary. There is probably a tool (IMPLIB, etc.) for that purpose comming along with your compiler

- Add the following line to your source files.:

  ```
  #include "GraphZ.h"
  ```

- At the beginning of your application (a good idea is to do this is the *WinMain* function), make the following call:

  ```
  GZRegister (your name, your password);
  ```

- Create one or more GraphZ window(s).

- One way for that is make a call to the API *CreateWindow* function with the string `"GraphZ"` as the first parameter (i.e. the window class name).
- Another way is to do this in your resource definition file (`*.rc`). For instance, in a dialog box definition, you can add a line like this:

  ```
  CONTROL "", ID_GRAPH, "GraphZ", WS_CHILD | WS_VISIBLE, 10, 10, 300, 200
  ```

  Possibly, your resource editor will complain about the fact that `"GraphZ"` is not a defined control calss at that time, but that's okay.

Note that the window's name (first parameter of the `CONTROL` statement, and second parameter of the *CreateWindow* API function) has no effect.

- GraphZ initializes a graph with default values. Then you may:
  1 Directly provide the settings:

  ```
  GZSetSettings (hGraphWindow, GRAPH structure);
  ```

  2 Load the settings from a given settings file:

  ```
  GZSetIni (hGraphWindow, lpszSettingsFilename);
  GZLoadIni (hGraphWindow);
  ```

  Note that you can do this at any time and repeat it any number of times.

  Note that there is a `GRAPH` structure associated with each GraphZ window. You will find this structure as parameter of a number of functions.

- Pass your data to GraphZ each time new data comes in. You'll probably want to do this in an interupt handling routine (a simple example being the `WM_TIMER` message).

  ```
  GZAddPoints (hGraphWindow, values, nNumValues);
  ```

- Make a **case-insensitive link** (and imports) of your application.

See also:

Functions, Structures and Constants.
A note about sizing.
The source code of the demo program included in the `GraphZ.zip` package.

Resizing the graph window

GraphZ provides no (re)sizing functionality (unless you create the window with the WS_OVERLAPPED style, but you probably won't want to do that). Two common cases are:

- The graphs are on a dialog box and their size is fixed.
- You make a MDI (Multiple Document Interface) application where each MDIClient window has one child: the GraphZ window. Your application takes care of always keeping the GraphZ window exactly the size of the MDIClient window client's area.

When resizing its window, GraphZ tries to make the best possible use of the available place. Still there are a couple considerations to do on that subject:

- Nothing is guaranteed when the size is really too small to contain all the items.
- The title will remain centered but will be truncated if needed.
- GraphZ will try to reserve horizontal place to display the legend completely, so don't make it (ie. the curves names) too long.
- The legend may be vertically trunctated if needed.

Functions, structures and constants

The "GraphZ.h" header file defines the following:

Functions:

- GZRegister
- GZSetIni
- GZLoadIni
- GZWriteIni
- GZGetSettings
- GZSetSettings
- GZAddPoints
- GZMakeDate
- GZExtractDate
- GZExtractTime

Structtures:

- GRAPH
- CURVE
- TITLE

Constants:

- Symbols
- Date & Time display formats

See also:

Settings file format.
Settings constraints.

GZRegister

BOOL FAR PASCAL GZRegister (LPSTR szName, LPSTR szCode)

szName is the name that you registered your copy of GraphZ with.
szCode is the password you received when you registered your copy of GraphZ.

return value:
Non-zero if the registration is okay, zero otherwise. In that case the graph title will be forced to be the string `"GraphZ: Unregistered Copy"`.

GZLoadIni

void FAR PASCAL GZLoadIni (HWND hWnd)

hWnd is the window handle of the GraphZ window for which you want to load the settings from a settings file.

The name of the settings file is contained in the `GRAPH` structure associated with the window. This is the `szIniFile` item. You may set or modify this file name by making use of either the `GZSetSettings` or, preferably the `GZSetIni` function.

If the settings file cannot be read or does not provide certain settings, default values will be provided.

If *hWnd* is not a GraphZ window, this function has no effect.

GZWriteIni

void FAR PASCAL GZWriteIni (HWND hWnd)

Dumps the current settings of a GraphZ window to its associated settings file. The settings file name must be set in the GRAPH structure.

hWnd is the GraphZ window for which to save the settings.

This function has no effect if:

- *hWnd* is not a GraphZ window, or
- the settings file name is empty or incorrect, or
- the file cannot be written (write protection, etc.)

GZSetIni

BOOL FAR PASCAL GZSetIni (HWND hWnd, LPSTR lpszIniFile)

This function sets the given settings file name in the GRAPH structure associated with the given GraphZ window. It does not modify the other member of the structure (use GZSetSettings). It does not load the settings from the file (use GZLoadIni).

hWnd is the GraphZ window for wich you want to set the settings file name.
lpszIniFile is the name of the settings file.

return value:
Zero if *hWnd* is not a GraphZ window. Non-zero otherwise.

GZGetSettings

BOOL FAR PASCAL GZGetSettings (HWND hWnd, LPGRAPH lpGraph)

Gets the current settings for the given GraphZ window

hWnd is the GraphZ window for which to retrieve the settings.
lpGraph is a far pointer to the GRAPH structure that is to receive the settings.

return value:
Zero if *hWnd* is not a GraphZ window. Non-zero otherwise.

GZSetSettings

BOOL FAR PASCAL GZSetSettings (HWND hWnd, LPGRAPH lpGraph)

This function replaces the current settings for the given GraphZ window, by the given settings. This is equivalent to user pressing the "Ok" button from the settings dialog box.

hWnd is the GraphZ window for which to replace the current settings.
lpGraph is a far pointer to a GRAPH structure specifying the new settings.

return value:
Zero if *hWnd* is not a GraphZ window. Non-zero otherwise.

Note: If some of the provided settings do not comply to the constaints, they will be automatically corrected.

Note: If you change the buffer length, the buffer will be cleaned up, which means you will loose all the historical data stored so far.

GZAddPoints

BOOL FAR PASCAL GZAddPoints (HWND hWnd, double FAR *lpValues, WORD nNumValues)

Adds new incomming data to a graph.

hWnd is the GraphZ window to add new data to.
lpValues is a long pointer on an array of values to add to the graph. You should supply one value for each curve. The number of curves can be found in the GRAPH structure.

- If you supply data for more curves than defined in the settings, GraphZ will try to allocate curves on the fly, based on default values. Note that the buffer for the allocated curves will be zero-initialized.
- If you supply data for less curves than defined in the settings, GraphZ will automatically generate the missing data (see the curves settings and alarms description in the **User's Guide**).

nNumValues is the number of values in the lpValues array. That is, for how many curves you are supplying data.

return value:
Zero if GraphZ failed to allocate some curves on the fly. Non-zero otherwise.

Note: In logarithmic scale mode, you should take care of providing strictly possitive values. However GraphZ actually does some error prevention on that subject.

GZMakeDate

double FAR PASCAL GZMakeDate(BYTE nDay, BYTE nMonth, WORD nYear, BYTE nHour, BYTE nMinute, BYTE nSecond, BYTE nHundredth)

Builds a date & time variable out of its components.

nDay is the day component of the date & time variable to build.
nMonth is the month component of the date & time variable to build.
nYear is the year component of the date & time variable to build.
nHour is the hour component of the date & time variable to build.
nMinute is the minute component of the date & time variable to build.
nSecond is the second component of the date & time variable to build.
nHundredth is the hundredth-of-a-second component of the date & time variable to build.

return value:
The resulting date & time value.

See also:

Date & Time constraints.

GZExtractDate

void FAR PASCAL GZExtractDate (double datetime, LPINT lpnDay, LPINT lpnMonth, LPINT lpnYear)

Extracts the date components out of a date & time variable.

datetime is the variable to extract the date components from.
lpnDay is a far pointer to an integer that is to receive the day component.
lpnMonth is a far pointer to an integer that is to receive the month component.
lpnYear is a far pointer to an integer that is to receive the year component.

See also:

Date & Time constraints

Date & Time variables

GraphZ uses the **double** C variable type to represent date & time values. This has the advantage that calculations such as adding time can be performed rapidly. Another advantage is that there is virtually no difference between normal numbered x-axis and a date & time x-axis.

The integral part of the floating point value represents the number of days since 1/1/1900.
The fractionnal part represents the fraction of a day. For instance 1h is 1/24, 1m is 1/1440, etc.

For your convenience GraphZ provides 3 functions for dealing with date & time values:

- GZMakeDate
- GZExtractDate
- GZExtractTime

Date & Time constraints

The date and time variables and components have the following boundaries:

variable	minimum value	maximum value
datetime	0.0 (1/1/1900)	73050.0 (1/1/2100)
day	1	28, 29, 30 or 31
month	1	12
year	0 (prefered) or 1990	200 (prefered) or 2100
hour	0	23
minute	0	59
second	0	59
hundredth	0	99

GZExtractTime

void FAR PASCAL GZExtractTime (double datetime, LPINT lpnHour, LPINT lpnMinute, LPINT lpnSecond, LPINT lpnHundredth)

Extracts the time components out of a date & time variable.

datetime is the variable to extract the time components from.
lpnHour is a far pointer to an integer that is to receive the hour component.
lpnMinute is a far pointer to an integer that is to receive the minute component.
lpnSecond is a far pointer to an integer that is to receive the second component.
lpnHundredth is a far pointer to an integer that is to receive the hundredth-of-a-second component.

See also:

Date & Time constraints

Curve

A GraphZ graph displays a number of curves simultaneously. This is fixed by the settings and is represented by the `numCurves` item of the `GRAPH` structure associated with the GraphZ window.

In order to define the settings for each curve, and array of `CURVE` srtuctures is present in the `GRAPH` structure. A curve is a series od points provided by your application (`GZAddPoints`). All the curves are plotted inside the curves box. The curves box is subject to a sliding window mechanism (that is, it is scroll forth whever your application adds new data, and the usr may scroll it back and forth as desired). Depending on the curve settings, each curve is plotted with a given pen. Each point is linked to the previous and next one with a line, and a symbol is plotted to represent the point. A curve also has a name (legend).

A detailled description of all the settings can be found in the **User's Guide**.

The **`CURVE`** structure is defined as follows in the `"GraphZ.h"` header file:

```
typedef struct
    {
    LOGPEN  pen;               /* logical color pen */
    BYTE    nSymbol;           /* symbol for use in the normal case */
    BOOL    bNoLine;           /* plot no line, only symbols */
    BOOL    bAlarms;           /* use alarm effects ? */
    double  alarmLow;          /* low alarm limit */
    BYTE    nSymbolLow;        /* if bAlarms, symbol to plot if value < alarmLow */
    double  alarmHigh;         /* high alarm limit */
    BYTE    nSymbolHigh;       /* if bAlarms, symbol to plot if value > alarmHigh */
    char    szLegend[31];      /* name for this curve (ie. legend) */
    BYTE     reserved[10];     /* for internal and future use. Do not modify ! */
    } CURVE, NEAR *PCURVE, FAR *LPCURVE;
```

See also:

Settings constraints.

More details can de found about alarms in the Curves Settings section of the **User's Guide**.

Graph

A GraphZ window stores all the settings related the its graph in a **GRAPH** structure. This structure is also used as parameter of a number of functions.

A detailled description of all the settings can be found in the **User's Guide**.

The **GRAPH** structure is defined as follows in the "GraphZ.h" header file:

```
#define NUM_CURVES            16
#define MAXFONTNAME           41

typedef struct
    {
    char        szIniFile[81];            /* the path and filename of the settings file */
    TITLE       title;                    /* the graph's title */
    char        szXLabel[41];             /* label for the X axis (horizontal) */
    char        szYLabel[41];             /* label for the Y axis (vertical) */
    double      xRate;                    /* real interval between two successive entries */
    double      xStart;                   /* real x-value of the first entry */
    double      xRange;                   /* real x-value range that has to be displayed */
    double      yMin, yMax;               /* y-scale boundaries when not in autoscale mode */
    double      logBase;                  /* base for logarithm scaling (eg. 10 for decimal log)
                                          */
    BOOL        bAutoscale;               /* autoscale the Y axis */
    BOOL        bLogarithmic;             /* logarithmic Y-scale */
    BOOL        bGrid;                    /* display a grid ? */
    BOOL        bTics;                    /* display tics and tic labels */
    BOOL        bMouseTrack;              /* display the real under mouse point coordinates */
    WORD        wTimeFormat;              /* date/time display format code for the x axis */
    WORD        bufferLen;                /* number of buffered values for each curve */
    WORD        numCurves;                /* number of curves to be displayed */
    char        szFontName[MAXFONTNAME];  /* font family name for all but the title */
    int         nFontSize;                /* font size in pts */
    COLORREF    background;               /* curves-box background color */
    LOGPEN      penGrid;                  /* the pen to use for the grid */
    LOGPEN      penTics;                  /* the pen to use for the tics */
    CURVE       curves[NUM_CURVES];       /* the curves definition */
    BYTE        reserved[500];            /* for indernal and future use. Do not modify ! */
    } GRAPH, NEAR *PGRAPH, FAR *LPGRAPH;
```

See also:

Settings constraints.

See for further details the description of the horizontal scale in the **User's Guide**.

Number of displayed curves

The number of displayed curves in a given GraphZ graph is fixed by the `numCurves` item of the associated `GRAPH` structure. There is a maximum of `NUM_CURVES` (16) curves per graph.

Your application supplies data (`GZAddPoints`) for a certain number of curves. When this differs from `numCurves`, then GraphZ either:

- tries to allocate the missing curves on the fly (which implies that the curve settings and buffer will be reset), or
- generates data for the curves that your application does not feed. This feature is realated to alarm (see further details in the **User's Guide**).

Title

A GraphZ graph has a title which is displayed and printed above the graph itself. The title will always be centered on the curves box. Possibly it will have to be truncated. There the **User's Guide** for a more detailed description.

The title can have a number of formatting attributes that make the graph more attractive. Hence a `TITLE` structure has been defined to store those attributes.

The **`TITLE`** structure is defined as foollows in the `"GraphZ.h"` header file:

```
typedef struct
    {
    char        szTitle[81];              /* the graph's tile */    char
    szFontName[MAXFONTNAME];              /* title font family name */
    int         nFontSize;                /* title font size in pts */
    BOOL        bBold;                    /* bold or regular */
    BOOL        bItalic;                  /* italic or regular */
    BOOL        bUnderline;               /* underline or not */
    BOOL        bGraved;                  /* graved or not (display only) */
    BOOL        bEmbossed;                /* embossed or not (display only) */
    BOOL        bBorder;                  /* border or not (display only) */
    COLORREF    color;                    /* RGB text color (ignored for sculpted styles ) */
    } TITLE, NEAR *PTITLE, FAR *LPTITLE;
```

Notes:

- When your GraphZ copy is unregistered (call the `GZRegister` function), the text `"GraphZ: Unregistered Copy"` will appear instead.
- The `bGraved` and `bEmbossed` attributes are mutually exclusive. When boths are set, `bGraved` has precedence.
- When `bBorder` is non-zero, a graved border is drawn. The exception is when `bGraved` is also non-zero, in which case an embossed border is drawn.
- When either `bGraved` or `bEmbossed` is non-zero, the color has no effect.
- `bBorder, bGraved` and `bEmbossed` are ignored for printing. The color is used instead.
- The title text is given by the `szTitle` attribute of the `TITLE` structure. The window name (ie, second parameter of the API *CreateWindow* function) has no effect.

Settings file format

The settings file has a Windows "*.ini" file compatible format. It is devided in sections, each of them having a number of items and values. Below is a list of those sections, items and default value.

When one or more setting cannot be loaded from a settings file, default values are provided as follows. This also happens when a graph is created (ie. initialized).

Section	Item	Default Value	Description
[GraphZ]	xLabel	(none)	the horizontal scale name
	yLabel	(none)	the vertical scale name
	Logarithmic	0 10.0	(0 / 1) = (true / false), base
	yScale	0 1	min max
	xRate	1	the "Rate" attribute
	xStart	0	the "First" attribute
	xRange	100	the "Range" attribute
	MouseTrack	1	(0 / 1) = (true / false)
	TimeFormat	0	a number indicating the display format for the date & time scale, 0 means normal numbered scale.
	Autoscale	1	(0 / 1) = (true / false)
	Grid	1	(0 / 1) = (true / false)
	Tics	1	(0 / 1) = (true / false)
	Font	Arial,8	general font name, point size
	Curves	0	number of curves (0 to 16).
	BufferLen	500	length of the history buffer
	Background	0 0 0	curves box background color (red green blue)
	GridPen	255 255 255 2 1	red green blue dotted=2/solid=0 size
	TicsPen	0 0 0 0 1	red green blue dotted=2/solid=0 size
[Title]	Title	(none)	
	FontName	Times New Roman	
	FontSize	16	title font size in points
	Bold	0	(0 / 1) = (true / false)
	Italic	0	(0 / 1) = (true / false)
	Underline	0	(0 / 1) = (true / false)
	Graved	0	(0 / 1) = (true / false)
	Embossed	1	(0 / 1) = (true / false)
	Border	0	(0 / 1) = (true / false)
	Color	0 0 0	red green blue
[CurveN]	Legend	"Curve N"	the legend for this curve
	Symbol	0 0	SymbolCode DrawLine=0/DrawOnlySymbol=1
	Color	r g b 0 1	red green blue dotted=2/solid=0 size. (r,g,b default values depend on the curve number N).
	AlarmEffects	0	(0 / 1) = (true / false)
	AlarmLow	0 0	LowAlarmLimit LowAlarmSymbol
	AlarmHigh	100 0	HighAlarmLimit HighAlarmSymbol

See also:

Settings constraints.

Symbols

Each curve has symbols atributes. There are three symbols for each curve: One for the normal case, one in case the value is beneath the low alarm limit, and one in case the value is above the high alarm limit. For that purpose the GraphZ.h include file defines the following codes.

#define	value	meaning
SYM_NONE	0	No symbol. Just the line.
SYM_CIRCLE	1	A circle (○) filled with the curves box background brush
SYM_SQUARE	2	A square (•) filled with the curves box background brush
SYM_PLUS	3	A plus sign (+)
SYM_CROSS	4	A cross (x)
SYM_DIAMOND	5	A diamond (◊) filled with the curves box background brush
SYM_DOT	6	A dot (.)
SYM_START	7	A start (*)
SYM_TRIANGLE	8	A triangle (▽) filled with the curves box background brush

Date & Time display formats

As described in the User's Guid, GraphZ extensively supports time scaling, on the horizontal scale (x-axis).
GraphZ can display dates and times in a number of formats. The `wTimeFormat` item of the `GRAPH` structure tells GraphZ which display format to use.
`wTimeFormat` is composed of two parts: the data and the time parts, which you must combine with the bitwise OR operator (|).

Date formats:

#define	value	meaning
DATE_NO	0x0000	no date is displayed
DATE_DMY	0x0001	"DD/MM/YY"
DATE_YMD	0x0002	"YY/MM/DD"
DATE_MDY	0x0003	"MM/DD/YY"
DATE_DmY	0x0004	"DD mon YY"
DATE_YmD	0x0005	"YY mon DD"
DATE_mDY	0x0006	"mon DD, YY"

Time Formats:

#define	value	meaning
TIME_NO	0x0000	no time is displayed
TIME_HM	0x0100	"HH:MM"
TIME_MS	0x0200	"MM:SS"
TIME_HMS	0x0300	"HH:MM:SS"
TIME_S99	0x0400	"SS.99"
TIME_MS99	0x0500	"MM:SS.99"
TIME_HMS99	0x0600	"HH:MM:SS.99"

Note: When both date and time are specified, the date is displayed first and then the time on the following line.

Note: When `(DATE_NO | TIME_NO)` is specified, then no date & time scale is used. A normal numbered scale is used instead.

Note: In these tables, DD, MM, YY, mon, HH, MM, SS and 99 stand respectively for the day, month, year, month name, hour, minute, second and hundredth of a second. They are all formatted on 2 charecters excet the month name which occupy three.

Note: The slash (/) and colon (:) characters used in these tables are only (very common) examples. The actual charecters used by GraphZ are those configured in the WIN.INI file. One can set those by the Control Panel.

Settings constraints

There are some constraints related to the settings, that is, the elements of the GRAPH, CURVE and TITLE structures. In addition, there are also constraints for date & time variables.

Here are the restrictions:

structure	item	restriction
GRAPH	xRate	> 0.0. Date & time constraints when relevant
	xStart	Date & time constraints when relevant
	xRange	> 0.0. Date & time constraints when relevant
	yMin, yMax	yMax > yMin. log error prevention in log scale. if (yMin == yMax) then yMax += 1.0
	logBase	log error prevention. != 1.0
	wTimeFormat	Date & time display format
	bufferLen	> 0 < 6553 Look at your available memory.
	numCurves	> 0 < NUM_CURVES (16)
	szFontName	Specify an existing (on your system) scalable font.
	nFontSize	> 0 In points. Making this > 10 quickly becomes ugly.
	background	color constraints.
	penGrid, penTics	pen constraints.
CURVE	szLegend	GraphZ will try to reserve enough place on the right of the graph to display the legend entirely. Thus, making this too long migh go at costs of the graph readability.
	pen	pen constraints.
	nSymbol, nSymbolLow, nSymbolHigh	A valid symbol code. Otherwise SYM_NONE is substituted.
	alarmLow, alarmHigh	alarmLow < alarmHigh When alarmLow is used to generate data that is not provided by the application (see the User's Guide), and when a logarithmic scale is used, alarmLow is subject to the logarithmic error prevention mechanism.
TITLE	szTitle	If you have not registered your GraphZ copy, the text "GraphZ: Unregistered Copy" will be displayed and printed instead.
	bGraved, bEmbossed	Mutually exclusive.
	bBorder	Normally a graved border is displayed, except in combination with bGraved where an embossed border is substituted. The bBorder, bEmbossed and bGraved attributes will have no effect for printing.
	color	color constraints.

See also:

Default settings.

Pen Constraints

You specify pens by means of the **LOGPEN** structure. This structure is defined in `"windows.h"` as follows:

```
typedef struct tagLOGPEN {
    UINT     lopnStyle;
    POINT    lopnWidth;
    COLORREF lopnColor;
} LOGPEN;
```

with the following restrinctions:

- a number of styles (like PS_SOLID, PS_DOT, etc.) are defined for **lopnStyle**.
- **lopnWidth.y** is not used. **lopnWidth.x** must be > 0. Only with the PS_SOLID style, **lopnWidth.x** can be > 1.

GraphZ adds the following restrinctions:

- **lopnStyle** may be only `PS_DOT` (2) or `PS_SOLID` (0). Any other style will be replaced by `PS_SOLID`.
- **lopnColor** must comply to the colors constraints.

Colors constraints

When you must specify a COLORREF variable, GraphZ puts the restriction that it must be either an RGB or a PALETTERGB value, but not a PALETTEINDEX.

Log(.) error prevention

In logarithmic scale, any value for which the **log(.)** function has to be calculated undergoes the following tests:

- value = abs(value)
- if (value == 0.0) then value = 1.0

Settings file name

The settings file name is stored in the `szIniFile` item of the `GRAPH` structure associated with a GraphZ window. You may set or modify this name by the `GZSetSettings` or preferably the `GZSetIni` function.

Note: The settings file name shoud either:

- include a complete path.
- be preceeded by `".\"` in order to specify the current directory.
- not include any path, in which case the window directory will be searched.

See also:

Settings file format
`GZLoadIni`
`GZWriteIni`

Product Information

This is GraphZ version 1.1.

All the fancy objects are from the ControlZ library.

ControlZ is already registered for GraphZ.

Note: If you would like to get both ControlZ and GraphZ, I can make some nice arrangements. Please e-mail me at: `cvzwynsv@estec.esa.nl`

GraphZ has to be registered for your application. When you register you will get a name and password that you'll have to pass to the `GZRegister` function at the begining of your program. Please do not tell anybody about these strings. It is even recommended that you do some encryption of them inside your application.

- **Registering GraphZ will prevent you from having the text `"GraphZ: Unregistered Copy"` substituted to the graph title.**
- **It will also give you the right to distribute GraphZ along with your application, without any limitation.**

Questions:
If you have any question, desire, remark or bug report, please e-mail me at:
`cvzwynsv@estec.esa.nl`

See also:

How to register.
Registration form.

How to register

The GraphZ registration fee is US$ 195.

You register GraphZ only once and hereby get the right to distribute it along with your application without limitation.

To register, proceed as follows:

- Fill in the registration form.

- Send it to:
  ```
  C. van Zwynsvoorde.
  Zeestraat 21,
  2201 KH Noordwijk.
  The Netherlands.
  ```
  or fax it to:
  ```
  (+31) 1719-85659
  ```

- Transfer the registration fee to the following account:
  ```
  Bank account number: 56.79.21.395
  Bank: ABN-AMRO bank in The Netherlands.
  Bank account owner: C.S van Zwijnsvoorde.
  ```
  I'm sorry I can't take credit cards.
  Possibly e-mail me when you have done so, so that I can check it more quickly.

- As soon as I have notice of your payment from the bank, I will send you back your name and password. You will get nothing more (argh!) since the distributed library is already fully featured and the completed documentation consists of this heelp faile, together with the User's Guide help file.

GraphZ 1.1 Registration Form

To register GraphZ 1.1, please first deposit US$ 195 on the bank account number 56.79.21.395 of the ABN-AMRO bank in The Netherlands under the name "C.S. van Zwijnsvoorde". Then fax or mail me a printed copy of this page. As soon as I have notification of your payment from the bank you will be retruned a registration name and a password.

Terms/Conditions:

I agree that GraphZ is distributed as shareware. No warranty exists, either express or implied. No liability is assumed for any damage or loss resulting from the use of this program. No claims are made regarding the accuracy of this program. The author reserves the right to change pricing in future versions. Your signature below indicates your acceptance of these terms and conditions.

Registration Name: _____
(Name you will supply to the GZRegister function.
This is recommended to be your application's name)

Your Complete Name:

_____

Postal Address:

_____

_____

_____

City: _____ Country: _____

Phone No: _____ Fax No: _____

Internet E-mail address: _____

How do you prefer to receive your user name and password ?
O. E-Mail O. Post O. Fax

Signature: _____ Date: _____

****
* Mail address: C.v.Zwynsvoorde. Zeestraat 21. *
* 2201 KH Noordwijk zh. The Netherlands *
* Fax number: (+31) 1719-85659 *
****