CD

Canvas Draw, A 2D Graphics Library

Version 4.4

(cd@tecgraf.puc-rio.br)

CD (Canvas Draw) is a platform-independent graphics library. It is implemented in several platforms using native graphics libraries: Microsoft Windows (GDI) and X-Windows (XLIB).

The library contains functions to support both vector and image applications, and the visualization surface can be either a canvas or a more abstract surface, such as Clipboard, Metafile, PS, and so on.

This work was developed at Tecgraf/PUC-Rio by means of the partnership with PETROBRAS/CENPES.

CD Project Management:

Antonio Escaño Scuri

Tecgraf - Computer Graphics Technology Group, PUC-Rio, Brazil
http://www.tecgraf.puc-rio.br/cd

Overview

CD is a platform-independent graphics library. It is implemented in several platforms using native graphics libraries: Microsoft Windows (GDI and GDI+) and X-Windows (XLIB).

The library contains functions to support both vector and image applications, and the visualization surface can be either a canvas or a more abstract surface, such as Clipboard, Metafile, PS, and so on.

To make the Application Programmers Interface (API) simple, all data are standard C types (int, double or char). Thus the application program does not have to maintain parallel data structures to deal with the graphic library.

Furthermore, the list of parameters of the CD primitive functions contains only the geometrical descriptions of the objects (line, circle, text, etc.). Where these objects should appear and what is the their color, thickness, etc. are defined as current state variables stored in the visualization surfaces. That is, the library is visualization-surface oriented, meaning that all attributes are stored in each visualization surface.

To control where the primitives are going to be drawn there we use the active canvas (visualization surface) concept. This means that you have to activate a given surface if you want to draw on it. As was pointed out, this approach simplifies the drawing for many primitives, so you do not have to pass the canvas as a parameter in each primitive. On the other hand, if there is no active canvas a call to a primitive does nothing and may cause an invalid memory access.

CD is free software, can be used for public and commercial applications.

Availability

The library is available for several **compilers**:

- GCC and CC, in the UNIX environment
- Visual C++, Borland C++, Watcom C++ and GCC (Cygwin and MingW), in the Windows environment

The library is available for several **operating systems**:

- UNIX (SunOS, IRIX, AIX, FreeBSD and Linux)
- Microsoft Windows NT/2K/XP

Support

The official support mechanism is by e-mail, using **cd AT tecgraf.puc-rio.br** (replace " AT " by "@"). Before sending your message:

- Check if the reported behavior is not described in the user guide.
- Check if the reported behavior is not described in the specific driver characteristics.
- Check the History to see if your version is updated.
- Check the To Do list to see if your problem has already been reported.

After all of the above have been checked, report the problem, including in your message: **function, element, driver, platform, and compiler.**

We host **CD** support features at **LuaForge**. It provides us Tracker, Lists, News, CVS and Files. The **CD** page at **LuaForge** is available at: http://luaforge.net/projects/cdlib/.

The discussion list is available at: http://lists.luaforge.net/mailman/listinfo/cdlib-users.
You can also submit *Bugs*, *Feature Requests* and *Support Requests* at:
http://luaforge.net/tracker/?group_id=88.
Source code, pre-compiled binaries and samples can be downloaded at:
http://luaforge.net/frs/?group_id=88.
The CVS can be browsed at: http://luaforge.net/scm/?group_id=88.

If you want us to develop a specific feature for the library, Tecgraf is available for partnerships and cooperation. Please contact **tcg AT tecgraf.puc-rio.br**.

Lua documentation and resources can be found at http://www.lua.org/.

Credits

This work was developed at Tecgraf by means of the partnership with PETROBRAS/CENPES.

We thank the people at the SEPROC department at CENPES and the library creators *Marcelo Gattass, Luiz Henrique de Figueiredo, Luiz Fernando Martha and Carlos Henrique Levy.*

Thanks to the people that worked in the library:
- *Antonio Scuri* (all since version 3.0)
- *Carlos Cassino* (X-Windows Platform Driver, Postscript Driver and WD functions),
- *Renato Borges* (X-Windows Platform Driver and WD functions),
- *Marcelo Gattass* (Microsoft Windows Platform Drivers),
- *Carlos Augusto Mendes* (DOS Platform Drivers),
- *Diego Fernandes Nehab* (DOS Platform Drivers, Lua binding),
- *Danilo Tuler* (Lua 5 binding),
- *Vinicius da Silva Almendra* (DGN Driver),
- *Milton Jonathan* (DXF Driver),
- *Pedro Miller* (DirectX Driver),
- *Erick de Moura Ferreira* (RGB Client Image Driver and Simulation,
- *Carolina Alfaro* (revision and translation of the user guide in Portuguese and English),
- *Camilo Freire* (CGM Driver),
- *André Derraik* (Macintosh Platform Beta Drivers),
- *Alexandre Ferreira* (Direct X Beta Driver)

We also thank the developers of the FreeType and Mesa libraries and of the XV program, for making the source code available, which helped us improve our implementation of the Simulation driver and of the X-Windows driver. Thanks to Jason Perkins for the Premake tool.

The CD distribution includes the FreeType library, this is a third party library not developed at Tecgraf. But its license is also free and have the same freedom as the Tecgraf Library License. You can read the Free Type license and copyright in the file freetype.txt. FreeType is copyright David Turner, Robert Wilhelm, and Werner Lemberg.

Documentation

This library is available at http://www.tecgraf.puc-rio.br/cd.

The full documentation can be downloaded from the Download by choosing the "Documentation Files" option.

The documentation is also available in Adobe Acrobat (cd.pdf ~400Kb) and Windows HTML Help (cd.chm ~200Kb) formats.

The HTML navigation uses the WebBook tool, available at http://www.tecgraf.puc-rio.br/webbook.

Tecgraf Library License

All the products under this license are free software: they can be used for both academic and commercial purposes at absolutely no cost. There are no royalties or GNU-like "copyleft" restrictions. They are licensed under the terms of the MIT license reproduced below, and so are compatible with GPL and also qualifies as Open Source software. They are not in the public domain, Tecgraf and Petrobras keep their copyright. The legal details are below.

The spirit of this license is that you are free to use the libraries for any purpose at no cost without having to ask us. The only requirement is that if you do use them, then you should give us credit by including the copyright notice below somewhere in your product or its documentation. A nice, but optional, way to give us further credit is to include a Tecgraf logo in a web page for your product.

The libraries are designed and implemented by a team at Tecgraf/PUC-Rio in Brazil. The implementation is not derived from licensed software. The library was developed by request of Petrobras. Petrobras permits Tecgraf to distribute the library under the conditions here presented.

The Tecgraf products under this license are: IUP, CD and IM.

CD Download

The main download site is the **LuaForge** site available at:

http://luaforge.net/project/showfiles.php?group_id=88

When LuaForge is offline, the **Tecgraf Download Center** is activated to offer a mirror site.

http://www.tecgraf.puc-rio.br/download

Before downloading any precompiled binaries, you should read before the Tecgraf Library Download Tips.

Some other files are available directly at the **CD** download folder:

http://www.tecgraf.puc-rio.br/cd/download/

History of Changes

Version 4.4 (12/Dec/2005)

- NEW: CDLua for Lua 5. The CDLua for Lua 3 is now also totally compatible with the "cd." name space used in the CDLUA for Lua 5. So the documentation now reflects only the new nomenclature although the old CDLua 3 names are still valid.
- NEW: attribute `"WINDOWRGN"` for the Native Windows and IUP drivers to set the shape of a window to the current complex clipping region.
- NEW: **`cdlua_close`** function to release the memory allocated by the **`cdlua_open`**.
- NEW: `"ROTATE"` attribute for PS driver, GDI+ base driver and GDI base driver.
- NEW: `CD_FILLSPLINE` and `CD_SPLINE` parameters for cdBegin in GDI+ base driver.
- NEW: support for complex regions for clipping using: **`cdBox`**, **`cdSector`**, **`Polygons`** and **`cdText`**. New parameter `CD_REGION` for **`cdBegin`** to create the region, new parameter `CD_CLIPREGION` for **`cdClip`** to select the region for clipping. New funtions to control regions: **`cdPointInRegion`**, **`cdOffsetRegion`**, **`cdRegionBox`** and **`cdRegionCombineMode`**. Valid only for the Windows GDI, GDI+ and X-Windows base drivers and their derived drives.
- NEW: mode for **`cdBegin`**, `CD_BEZIER`.
- NEW: filled primitive **`cdChord`**.
- NEW: polygon fill rule control using **`cdFillMode`** with `CD_EVENODD` (the default) and `CD_WINDING` parameters.
- NEW: line cap and line join styles using **`cdLineCap`** and **`cdLineJoin`**.
- NEW: typeface `CD_NATIVE` to indicate that a native font has been selected.
- NEW: custom line style using **`cdLineStyleDashes`** and **`cdLineStyle`**(`CD_CUSTOM`). This replaces the attribute "`USERLINESTYLE`".
  (All NEW, when not specified the divers, are valid for all the drivers, except DXF, DGN e CGM.)
- NEW: text utility function **`cdTextBounds`** that returns the oriented bounding rectangle.
- NEW: `"IMAGEFORMAT"` and `"IMAGEALPHA"` attributes for the Windows base driver.
- NEW: In GDI+, the `CD_CLIPBOARD` driver supports EMF and BMP formats.
- NEW: function **`cdReleaseState`** to release the memory allocated by a state. The **`cdRestoreState`** does not release the memory anymore so it can be used several times for the same state.
- IMPROVEMENT: Optimization flags now are ON when building the library in all platforms.
- IMPROVEMENT: Upgraded Freetype to version 2.1.10. The CD library file size increased because of this. But we gain a better text rendering for images.
- IMPROVEMENT: Better organization of the documentation.
- CORRECTION: Invalid cdKillImage in X-Windows when active canvas is not the canvas where the image was created.
- CORRECTION: Text clipping for `CD_IMAGERGB` driver.
- CORRECTION: fixed size in milimeter of **`cdGetScreenSize`** in Win32.
- CORRECTION: fixed size of the EMF picture.
- CORRECTION: fixed the parse of filenames with spaces for all file based drivers. The filename must be inside double quotes (") if it has spaces.

- CORRECTION: **cdSetAttribute** in Lua now can set nil values.
- CORRECTION: fixed **cdSector** when interior style is CD_HOLLOW, to include two lines connecting to the center point.
- CORRECTION: In GDI+, the NATIVEWINDOW driver ignored other data pointer configurations in **cdCreateCanvas**.
- CORRECTION: In GDI+, **cdStipple** was not updated when the foreground or background colors where changed.
- CORRECTION: In GDI+, **cdSector** and **cdArc** have incorrect angles.
- CORRECTION: "simple.c" and "simple.zip" were outdated. Now new makefiles were added.
- CORRECTION: in Windows base driver small incompatibility in cdNativeFont with the IUP FONT attribute.
- IMPROVEMENT: In Windows the NATIVEWINDOW driver now accepts a NULL pointer to draw in the entire screen.
- IMPROVEMENT: Optimized **cdPutImageRGBARect** in Windows base driver.
- CHANGE: Now by default CD will not print X-Windows messages. To enable you must set the CD_XERROR environment variable.
- CHANGE: The default fill rule for polygons in CD_PS is now the Even-Odd rule. Matching the other drivers.
- CHANGE: Line Styles in GDI+ was corrected again to match GDI line styles when line width is not 1.
- CHANGE: The native WC support in GDI+ was removed because of alignment and size problems, simulation will be used.
- CHANGE: the EMF drivers now ignore the resolution parameter. For EMFs, the resolution is always the screen resolution.
- CHANGE: the value of following attributes were changed to strings "IMAGEMASK", "IMAGEPOINTS", "ROTATE", "GRADIENTCOLOR", "IMAGETRANSP" and "IMAGEFORMAT".
- CHANGE: in GDI+ base driver, the **cdBegin** modes CD_IMAGEWARP and CD_GRADIENT were moved to attributes "IMAGEPOINTS" and "LINEGRADIENT". Mode CD_PATHGRADIENT was renamed to CD_FILLGRADIENT, and "PATHGRADIENT" attribute was renamed to "GRADIENTCOLOR". Their definition was also changed.
- CHANGE: **cdImageEx** was renamed to **cdBitmap**, and now supports only client images. This will cause a conflict with a macro definition in "im_image.h" header of the IM toolkit. Include this header before "cd.h" and inbetween set "#undef cdPutBitmap". The IM macro will be changed in the next IM version.
- CHANGE: **cdText** is not dependent on the **cdBackOpacity** anymore. Text now is always transparent. If you need a box under the text use **cdTextBox** to calculate the dimensions for **cdBox**.

Version 4.3.3 (25/Aug/2004)

- NEW: new "USERLINESTYLE" attribute for the base GDI and X11 drivers.
- NEW: new "GC" attribute for the base X11 driver.
- IMPROVEMENT: in the Native Window driver for the Windows system, the creation using a HDC can have an addicional parameter for the canvas size.
- IMPROVEMENT: in cdTextSize for the GDI+ base driver we now compensates the height in -10% to match the GDI height.
- IMPROVEMENT: The GDI+ printer driver now returns the HDC attribute.
- CORRECTION: fixed a bug in **cdNativeFont** for the GDI+ base driver.
- CORRECTION: again fixed a rounding error in **cdPutImage*** for big zooms.

Version 4.3.2 (14/Apr/2004)

- CORRECTION: in the Win32 and X-Win drivers the **cdPutImageRGB** and **cdPutImageMap** functions when zooming bigger then the canvas where incorrectly positioning the image by some pixels because of round errors.

Version 4.3.1 (07/Nov/2003)

- CORRECTION: in the Win32 driver the clipping of **cdPutImage*** functions when zooming was wrong. In the DoubleBuffer driver the main canvas **cdOrigin** can be used to move the image in the swap operation (**cdFlush**). In the GDI+ DoubleBuffer driver there was an error in the **cdFlush** when some primitive used world coordinates directly in the main canvas.

Version 4.3 (06/Mar/2003)

- NEW: the function **cdlua_getcanvas** retreives the pointer of a canvas created in Lua.
- NEW: in Win32 the function **cdUseContextPlus** change the behavior of the Windows drivers NativeWindow,

IUP, Image, Printer, EMF and Double Buffer to make them use the GDI+ for drawing. GDI+ does not have support for XOR Write Mode, but it has other resources like: transparency, anti-aliasing, gradient filling, bezier lines and filled cardinal splines. WC functions are directly implemented in the base driver. Two new functions were created to support transparency in the CD color coding: **cdEncodeAlpha** and **cdDecodeAlpha.** Check the documentation for more information.
- IMPROVEMENT: the Lua binding is now distributed in the same package. There is only one version number.
- CORRECTION: the PS header had same flaws, the character ":" was missing in some DCS attributes.
- CORRECTION: screen resolution was wrong in the Win32 driver, this afects the size of the canvas in milimeters.
- CORRECTION: in the Win32 driver the creation of a polygon for clipping does not activate the clipping.
- CORRECTION: in the Win32 driver the function **cdNativeFont** using "-d" parameter need some ajusts. Also the returned string does not contains all the used parameters.
- CORRECTION: in the Win32 driver the function **cdPutImageRectRGBA** had a positioning error.

Version 4.2 (20/July/2001)

- IMPROVEMENT: in driver Win32, **cdNativeFont** accepts parameter "-d" on the font name to show the font-selection dialog.
- IMPROVEMENT: the whole code can now be compiled as C++.
- IMPROVEMENT: functions **wdPattern** and **wdStipple** were changed to make pattern deformation more uniform.
- CORRECTION: in the Clipboard driver on Win32, when parameter "-b" was used the image was not correctly copied.
- CORRECTION: in certain moments, color vectors were being allocated with size 4 and should be "sizeof (long)". This was done to improve the compatibility with 64-bit systems.
- CORRECTION: **cdPutImageRectRGB** in driver ImageRGB had a memory-invasion error in some cases when the image was placed in a negative coordinate.

Version 4.1.10 (04/May/2000)

- IMPROVEMENT: the driver Native Windows in Win32 now also accepts an already created HDC handle as a parameter.
- IMPROVEMENT: in the **cdPutImageMap*** functions, in case the color vector is null, a vector with 256 gray shades in assumed.
- CORRECTION: **cdRegisterAttribute** was not verifying whether the attribute had already been registered.
- CORRECTION: function **cdArc** in the simulation driver (includes **ImageRGB**) was returning without drawing anything in an incorrect test.
- CORRECTION: function **cdTextBox** was returning incorrect values when the text had an orientation different from the default one in some alignment instances.
- CORRECTION: in function **cdRGB2Map** there was a memory invasion.
- CORRECTION: the vector text simulation was not freeing the memory used for fonts loaded from files.
- CORRECTION: in the Doubled Buffer driver in X-Windows there was an invalid memory liberation.
- CORRECTION: in the Lua binding, in several functions receiving or returning tables, the first index was being considered as 0, but in Lua they must be 1. This correction includes **cdVectorTextTransform, cdGetVectorTextBounds, wdGetVectorTextBounds, cdGetClipPoly** and **wdGetClipPoly**.
- CORRECTION: when the PS driver generated EPS, it did not correctly add the description of the bounding box (a line break was missing).
- CORRECTION: the vector text drawing functions did not take into account the fact that the default font and the GKS fonts were in ASCII standard. Now a conversion from ANSI to ASCII is made before these fonts are used for drawing.
- CORRECTION: in the X-Win driver, an error in the X-Vertex library caused the texts in 90/270 degrees to be drawn incorrectly.
- CORRECTION: in the X-Win driver, the **cdPutImageMap** functions were generating a memory invasion when the X was in 16 bits.
- CORRECTION: in the Win32 driver, very large non-filled polygons were not being drawn in Windows 9x. To correct that, they were divided into smaller polygons.

Version 4.1 (24/Nov/99)

- NEW: new basic Windows driver attributes that allow controling the internal simulation of pattern/stipple, XOR text, and filled polygon ("SIMXORTEXT", "SIMPATTERN8X8", "SIMPENFILLPOLY"). New attribute for returning the HDC of the Windows canvas.
- NEW: the PS driver accepts landscape orientation as a parameter. New "POLYHOLE" attribute allows controling the number of holes in a closed polygon. New "-1" parameter forces a level 1 Postscript. New "-g" parameter

adds comments to the PS file in order to better explain what is done. New "CMD" attribute saves a string to the file.
- NEW: new environment variable, CD_QUIET, does not display in *stdout* the library's version information.
- NEW: two new exclusive functions for the Native Window driver: **cdGetScreenColorPlanes** and **cdGetScreenSize**.
- NEW: new CD_DBUFFER driver implements a *double buffer* using a server image.
- NEW: new attributes in the ImageRGB driver: "REDIMAGE", "GREENIMAGE" and "BLUEIMAGE".
- NEW: new functions **wdGetVectorTextBounds** and **cdGetVectorTextBounds** to obtain the bounding box of the vector text.
- NEW: new **wdGetFont** function. It is equivalent to **cdGetFont**, but the returned size is in millimeters.
- CORRECTION: the management of WD functions was incomplete for functions **cdPixel**, **cdVertex** and **cdPutImage***. This resulted in a wrong or out of the canvas positioning. It only affects drivers PS and METAFILE.
- CORRECTION: function **cdActivate** in Lua was not returning the correct values.
- CORRECTION: when the image was partially out of the window, above or below, functions **cdPutImageMap** and **RGB** were drawing a wrong portion of the image.
- CORRECTION: in the CGM driver, after opening the file of the **cdPlay** function, the check to see if the opening had been successful was not being done.
- CORRECTION: when the active canvas was already NULL, the activation of a NULL canvas was generating a memory invasion.
- CORRECTION: in the creation of EPS, the PS driver was adding a wrong call to setpagedevice. The **cdPutImageMap** function was modifying the wrong PS parameter in the file. The margin clipping was not saved when the drawing's clipping area was changed. The clipping area, when drawing in WD, was being incorrectly modified.
- CORRECTION: in the IMAGERGB driver, functions **cdRedImage**, **cdBlueImage** and **cdGreenImage** were returning invalid pointers.
- CORRECTION: when initializing text simulation functions, the opened font file was not being closed. This affected all CD drivers, but was only apparent in the application that opened and closed many drivers.
- CORRECTION: the approximate computation of the text size was not accepting sizes in pixels.
- CORRECTION: the creation of the IMAGERGB driver in Lua was incorrect when the resolution parameter (which is optional) was not specified.
- CORRECTION: functions **cdGetClipPoly** and **wdGetClipPoly** in Lua were causing memory invasion.
- IMPROVEMENT: in the PS driver, when the Map image is actually a grayscale, function **cdPutImageMap** uses an 8 bit image, thus saving memory. Level 2 Postscript functions rectfill, rectstroke and rectclip are now used. The comments in DCS were updated to DCS version 3 and were increased to improve the document's portability.
- IMPROVEMENT: in driver X-Windows, the text drawing attribute was implemented with any orientation.
- CHANGE: function **cdVersion** in Lua now behaves just like in C. A global Lua variable, **CDLUA_VERSION**, was created containing the version of the Lua binding library - for example: "CDLua 1.3.0".
- CHANGE: function **cdVectorTextTransform** now returns the previsous transformation matrix.
- TIP: when the creation of a printer canvas returns NULL, this means the user has cancelled the print dialogue.
- TIP: in case the foreground and the background colors are modified, the hatched and monochromatic fillings must be changed again in order to be updated.
- TIP: the Native Window driver works regularly in a window of the OpenGL library.
- TIP: in the Guide topic of this user guide, a comparison between the CD library and other graphics libraries available for free in the Web was included.

Version 4.0.1 (05/Mar/99)

- CORRECTION: in the Windows driver, the polygon simulation with pattern was corrected to polygons with repeated points.
- CORRECTION: in the Windows driver, function **cdNativeFont** was corrected for IUP fonts. It was affecting the Matrix's visualization.
- CORRECTION: function **cdNativeFont** was wrongly testing its input parameter and always returning.
- CORRECTION: in the drivers IUP and Native Window, the **cdGetCanvasSize** function was corrected. When the window size was changed, the values in millimeters were not updated to **cdActivate**.
- CORRECTION: in the CGM driver, function **cdPlay** was generating problems in reading and displaying cell arrays. When the **cdCreateCanvas** function used the default values for dimensions and resolution, it generated files with errors.
- IMPROVEMENT: in the X-Windows driver, function **cdPixel** was optimized. It now compares the color to the foreground color and reuses the value.

Version 4.0 (18/Feb/99)

- **Summary**: (necessary due to the great number of changes).
  - Update of the Lua binding.
  - Several changes in the internal structure (most bringing benefits only to the driver developer).
  - Countless corrections.
  - Small changes in the functions **cdHatch**, **cdScrollImage**, **cdFont** and **cdPlay**.
  - Optimization of functions **wdVectorFont** and **cdMark**.
  - New functions:
    cdCreateCanvasf, cdGetContext, cdContextCaps, cdSaveState, cdRestoreState, cdSetAttribute, cdGetAttribute
    cdOrigin, cdRect, wdRect, cdGetFont, cdGetStipple, cdGetPattern, cdTextBox
    cdPutImageRectRGB, cdPutImageRectRGBA, cdPutImageRectMap,
    cdCreateImageEx, cdKillImageEx, cdPutImageEx, cdGetImageEx.
  - New WD functions: **wdHardcopy, wdPattern, wdStipple, wdPixel, wdPutImageRect, wdPutImageRectRGB, wdPutImageRectRGBA** and **wdPutImageRectMap**.
  - New vector text functions: **cdVectorFont, cdVectorTextDirection, cdVectorTextTransform, cdVectorTextSize, cdGetVectorTextSize, cdVectorCharSize, cdVectorText** and **cdMultiLineVectorText**.
  - **wdActivate** is no longer necessary.
  - Driver IMAGERGB complete.
  - Driver SIMULATE no longer exists; now function **cdSimulate** must be used.
  - New driver DIRECTDRAW.
  - Policy change of **cdPalette** in the X-Windows driver
  - IUP driver is now in a separate library.

IMPORTANT NOTE: This version is not totally compatible to the previous one. The applications using the driver IUP must be relinked, as this driver is now in a separate library, "**cdiup**". The Lua applications must also be modified to include a call to function **cdluaiup_open** after **cdlua_open**, and must be linked with the "**cdluaiup**" library. The SIMULATE driver no longer exists, therefore the applications that used it must be modified to use the new function, **cdSimulate**, without the need for creating a new driver.

- IMPROVEMENT: the internal structure of the library was changed once again. One of the purposes is to make the drivers become independent from the function table. With this change, adding a new function to the function table does not imply editing the old drivers. We also allowed the drivers not to implement functions that do not make sense in their context. Another modification simplifying the work in the drivers was to bring the attribute query mechanism to the library's control part, freeing the drivers from this obligation. Taking the chance, we determined that a change in an attribute to a value equal to the current one will not be effective, thus saving calls to the driver. Now, the value of an attribute is changed even if the driver function is not implemented, as the driver can query this attribute later on. The management of default values of the attributes is also done by the library's control part. All these changes prepare the library to a new philosophy: before, if a driver did not contain a certain feature, it simply did nothing. The new philosophy will be: if a driver does not contain a certain feature, then the simulation of this feature will be activated.
- IMPROVEMENT: when a canvas which is already active is activated again, an internal driver function is now called, notifying an update instead of an activation.
- IMPROVEMENT: the use of the CD canvas with a IUP canvas is better described in the manual, showing the various ways of treating the canvas creation prooblem.
- IMPROVEMENT: all functions in the control module now have ASSERT directives. Thus, using the library with depuration information, one can better detect simple errors.
- IMPROVEMENT: in order to use the IUP driver, it must be linked with the "**cdiup**" library, apart from the "**cd**" library (`cdiup.lib` in Windows, `cdiuplib.a` in UNIX).
- IMPROVEMENT: the IMAGERGB driver is now implemented using the simulation functions.
- IMPROVEMENT: the **cdMark** function is back to the function table, so that the drivers in which the primitive can be implemented can profit from it.
- IMPROVEMENT: in order to assure that the use of server images is done only between canvases of the same driver, or of the same basic driver, an internal structure was defined for the server image containing the functions of the driver from which the image was created. Thus, if the **cdKillImage** function is called with an active canvas of a different kind from that in which the image was created, the **KillImage** function of the correct driver will be called.
- IMPROVEMENT: in the X-Windows driver, the XV code was used to optimize functions **cdPutImageRectRGB** and **cdPutImageRectMap**.
- IMPROVEMENT: the Lua binding was updated. Now the user guide contains Lua function together with C functions.
- CHANGE: in the X-Windows driver, **cdPalette**'s policy was changed to fulfill the requirements of some applications, which wanted to force a palette. Please see the function's documentation in the driver.
- IMPROVEMENT: the CGM driver used to always store the **cdForeground** attribute before drawing a primitive;

now it stores the attribute only when it is changed. The **cdBackOpacity** function was not implemented. The **cdFlush** function was not preserving the canvas attributes. Now when the canvas size is not specified in the **cdCreateCanvas** function, the VDC Extension saved to the file is the figure's bounding rectangle. The patterns and/or stipples selected were being stored in a way so that only the first one was valid.

- CHANGE: the documentation of the old DOS driver was removed from the user guide.
- CHANGE: the default resolution for drivers DGN, DXF, METAFILE, CGM and ImageRGB is no longer 1 but 3.8 points per mm (96 DPI).
- CHANGE: in the **cdInteriorStyle** function, if stipple or pattern are not defined, the state of the attribute is not modified. There is no longer a default 32x32 pattern or stipple.
- CHANGE: in functions **cdFontDim** and **cdTextSize**, if the driver does not support this kind of query, the values are estimated.
- CHANGE: function **cdHatch** now returns the previous value.
- CHANGE: function **cdScrollImage** is now called **cdScrollArea**, better reflecting its functionality, since it does not require any explicitly defined image to be performed. The old function is maintained to assure compatibility with old applications.
- CHANGE: the **cdPlay** function now accepts all window parameters null. In this case, the primitives in the file are interpreted without scaling.
- CHANGE: **cdFont**now accepts font sizes in pixels when negative values are used as a parameter.
- CHANGE: the WD functions were included in the library's function table, so that the drivers can use floating point precision when storing primitives. Presently, only the drivers PS and METAFILE have this resource directly implemented. With this change, the **wdActivate** function became obsolete and is no longer necessary. For purposes of compatibility with other applications, it was maintained only as a call to function **cdActivate**.
- CHANGE: drivers EMF and WMF now accept the resolution as a parameter.
- NEW: internal modification of the Windows driver to allow the creation of the DirectDraw driver.
- NEW: DirectDraw driver to accelerate video access to high-performance applications.
- NEW: function **cdInteriorStyle** now accepts style CD_HOLLOW. When this style is defined, the **cdBox** and **cdSector** functions behave like their equivalents **cdRect** and **cdArc**, and the polygons with the CD_FILL style behave like CD_CLOSED_LINES.
- NEW: new functions:
  - **cdCreateCanvasf** accepts parameters equivalent to **sprintf**, helping in the creation of some canvases.
  - **cdOrigin** allows translating the origin - for instance, to the center of the canvas.
  - **cdGetContext** returns the context of a given canvas; it can be compared with predefined contexts, such as "CD_PS".
  - **cdSaveState** and **cdRestoreState** allow saving and restoring the state of attributes of the active canvas.
  - **cdSetAttribute** and **cdGetAttribute** allow passing customized attributes to the driver, which are ignored if the driver does not have the attribute defined.
  - **cdContextCaps** returns a combination of several flags informing the available resources of the canvas in that context.
  - Driver SIMULATE no longer exists. Now function **cdSimulate** must be used. The simulation can be activated and deactivated at any moment.
  - **cdRect** and **wdRect** allow drawing a box with no filling.
  - **cdGetFont** returns the values of the font modified by function **cdFont** and ignores those modified by **cdNativeFont**.
  - **cdTextBox** returns the horizontal bounding rectangle of the text box, even if it is bended.
  - **cdGetStipple** and **cdGetPattern** return current stipple and pattern. With these functions and with function **cdGetFont**, one can now totally change and restore the attributes of a given canvas.
  - **wdPattern** and **wdStipple** allow specifying the style in world coordinates. The size of the image is modified to the specified size in millimeters.
  - functions **cdPutImageRectRGB**, **cdPutImageRectRGBA** and **cdPutImageRectMap** allow specifying a rectangle in the image to be used for the drawing instead of the whole image.
  - **wdPixel**, **wdPutImageRect**, **wdPutImageRectRGB**, **wdPutImageRectRGBA** and **wdPutImageRectMap** are equivalent to **cdPixel**, **cdPutImageRect**, **cdPutImageRectRGB**, **cdPutImageRectRGBA** and **cdPutImageRectMap**, respectively, but the target coordinates are specified in world coordinates.
  - New vector text functions: **cdVectorFont**, **cdVectorTextDirection**, **cdVectorTextTransform**, **cdVectorTextSize**, **cdGetVectorTextSize**, **cdVectorCharSize**, **cdVectorText**, **cdMultiLineVectorText**. The vector text can now be used without the need of world coordinates. Functions **wdVectorFont** and **wdVectorTextTransform** have become obsolete, though they still exist for compatibility reasons.
  - **wdHarcopy** helps drawing WD primitives in devices, adjusting Window and Viewport.
  - Auxiliary functions were created to manipulate all sorts of images in a single way, being either client, RGB, RGBA, MAP, or server images: **cdCreateImageEx**, **cdKillImageEx**, **cdPutImageEx**, **cdGetImageEx**, etc.

- CORRECTION: the documentation of function **cdFont** was confusing, causing errors in the conversion from pixels to points.
- CORRECTION: function **wdFont** was making a wrong conversion of the font size parameter from millimeters to points.
- CORRECTION: functions **wdVectorText** and **wdMultiLineVectorText** were generating an extra polygon when the text contained blank spaces in certain positions.
- CORRECTION: the PS driver was not prepared for marked texts. Function **cdFlush** did not preserve current attributes. The interior style was affecting line drawing. The text alignment now takes into account an estimation for the baseline. Function **cdTextOrientation** was implemented. The world coordinate functions were implemented directly in the driver. Hatch and stipple interior styles were implemented, but they are still only opaque.
- CORRECTION: in the X-Windows driver, function **cdGetColorPlanes** was returning 8 bpp even if the canvas was 24 bbp when the default visualization was different from the canvas' visualization in that X server. Text position on the screen was above the one entered. Function **cdFont** was looping in certain conditions. Function **cdEnd** in the X-Windows driver in the AIX system was generating an error and aborting the program if only one point of the polygon was specified. Dashed lines were always opaque, ignoring the **cdBackOpacity** attribute.
- CORRECTION: in the Clipboard driver for X-Windows, a parameter was missing which prevented it from working properly. Before the update, it used that of the IUP/Native Window active canvas.
- CORRECTION: in the Windows driver, the text position on the screen was above the position provided. Filled polygons had a one pixel error to the right and below due to the small `NULL` used. Fillings with hatch, pattern and stipple still contain errors. The internal simulation of polygons filled with pattern and stipple was also corrected; they had one additional pixel to the right and below. Text alignment treatment was improved.
- CORRECTION: driver WMF now has text alignment.
- CORRECTION: in the PRINTER (Windows) driver, function **cdFlush** was not preserving current attributes.
- CORRECTION: in the CGM driver, the text align interpretation was corrected. The **cdMark** function is implemented directly in the driver. Function **cdBackOpacity** was implemented. Mark interpretation was also corrected.
- OPTIMIZATION: function **wdVectorFont** only loads the new font if it is different from the current one.
- OPTIMIZATION: function **cdMark** now modifies fill and line attributes only if they are different from the one needed to draw the mark.

Version 3.6 (05/May/98)

- CORRECTION / Win32: every time the clipping region changed the old region was not deleted.
- NEW: new function **cdRGB2Map**, which converts an RGB image into a 256 indexed-colors image. It is the same algorithm used in the IM library - in fact, it is the same code.
- IMPROVEMENT: the **cdMark** function now uses the **cdPixel** function when drawing a mark of 1 pixel size.
- IMPROVEMENT / Win32: the **cdPixel** function now uses the **SetPixelV** function when not under Win32s. This function is faster than the **SetPixel** function because it does not return the old value.
- IMPROVEMENT / Win32: the polygon used for clipping is now optimized to not include 3 points that are in the same horizontal or vertical line.
- CORRECTION / WD: the **wdVectorText** function was not drawing correctly when extended characters (>128) were used.
- CORRECTION / X: the **cdPalette** function and the color management for canvases with only 256 colors were wrong. Each canvas had its own management, now all canvases in the same aplication use the same management.
- CORRECTION / X: several resource and memory leaks were corrected.
- CORRECTION / IMAGERGB: functions **cdRedImage**, **cdGreenImage** and **cdBlueImage** were not returning the correct pointer.
- CORRECTION / SunOS: drivers IMAGERGB, SIMULATE and NATIVEWINDOW use the "%p" format string, but in the SunOS they use "%d" because of an internal bug of the run time library of this OS.
- IMPROVEMENT / IUP: driver IUP sets the **cdCanvas** function as an attribute of the **IupCanvas** passed as a parameter using the name "_CD_CANVAS".
- MANUAL: the manual appearance was changed to match the new Tecgraf standard.

Version 3.5 (07/Jan/98)

- NEW: the **cdTextDirection** function allows raster text to be drawn in any direction. Up to now it is implemented only in the basic Win32 driver.
- CORRECTION / X / NativeWindow: the canvas was not created if the screen was 0.
- CORRECTION / Win32 / NativeWindow: now the driver considers the existence of non owner-draw device contexts.
- CORRECTION / Win32: function **cdClipArea** was not including `xmax` and `xmin` in the clipping area.
- IMPROVEMENT: the **cdCallback** typedef was defined, being useful for type casting when calling the

cdRegisterCallback function.
- CORRECTION / Win32: a compatibility problem with the **cdNativeFont** string and the WINFONT IUP attribute was corrected.
- IMPROVEMENT / Win32: the **cdPutImageRGB** and **cdPutImageMap** functions use a cache memory for best performance.
- CORRECTION: text size estimation for the CGM and PS drivers now uses Courier New as the "System" font. As it was, it was causing a memory invasion.

Version 3.4 (12/Nov/97)

- IMPROVEMENT / X: memory use of the **cdPutImageRGB**, **cdPutImageRGBA** and **cdPutImageMap** functions was optimized, as well as the performance of the **cdPutImageMap** function.
- IMPROVEMENT / X and Win32: when the canvas has bpp <= 8, function **cdPutImageRGB** converts the image into Map before displaying it.
- IMPROVEMENT / X and Win32: if a font's parameters are the same as the current parameters, the **cdFont** function does nothing.
- DOC / PS: the "-d" parameter for the EPS option was not documented.
- CORRECTION / PS: parameters "-w" and "-h" were incorrectly interpreted.
- CORRECTION / X: the internal function names were generating an error in the VMS plataform.
- CORRECTION / X: the **cdKillCanvas** function was freeing some pointers of the active canvas.
- IMPROVEMENT / Win32: the **cdVertex** function now ignores duplicate points.
- IMPROVEMENT / Win32: the **cdNativeFont** function also accepts the font string of the WINFONT IUP attribute.
- CORRECTION / DXF: corrections in color conversion and in the **cdArc** function for small radius were made, and an unnecessary identation was removed.

Version 3.3 (19/Sep/97)

- IMPROVEMENT / X: the **cdFont** function now has a better heuristic to find a closer font if the requested font does not match an available one.
- IMPROVEMENT / X: the **cdPattern** and **cdStipple** functions now use a bitmap cache to store the *pixmap* and do not recreate it if the size is not changed.
- CORRECTION / X and Win32: the **cdPutImageRect** function was placing the bitmap in a wrong position.
- CORRECTION / Win32: the **cdCreateImage** function did not return NULL when the creating failed.
- IMPROVEMENT / Win32: the **cdPutImageRGB**, **cdPutImageRGBA** and **cdPutImageMap** functions were largely optimized when the picture displayed is larger than the screen.
- IMPROVEMENT / WMF: using the **cdPlay** function we discovered that the size of the picture was incorrect in the header file, so we first had to calculate the bounding box and then interpret the picture.
- IMPROVEMENT / PS and CGM: now the **cdFontDim** and **cdTextSize** functions return approximate dimensions, instead of nothing.
- CORRECTION / PS: the default font was not being set when the canvas was created.
- CORRECTION / PS: text alignment was incorrect in the vertical direction.
- CORRECTION / SIM: the clipping algorithm of the **cdPixel** function of the Simulation driver was corrected.
- CORRECTION / CD: now you can activate a NULL canvas. When you get the active canvas and restore it, if it was NULL the **cdActivate** function was accessing an invalid pointer.
- MANUAL: several changes were made on the online manual structure, and were added to the CDLua page.

Version 3.2

- A problem in the **cdFlush** function in the Postscript driver was corrected. It was not setting the scale.
- Functions **wdFontDim** and **wdTextSize** now check if the return pointers are not NULL.
- An internal function in the DGN driver was drawing an ellipse with two times the axis size.
- The **cdFont** function was corrected to store the font names in the CGM driver.

Version 3.1

- Several minor bugs in the Win32 Printer driver and in the Postscript driver were corrected. The EPS mode of the PS driver now generates a "showpage" PS function so it can be printed.
- The Clipboard driver was implemented in Motif. The **cdPlay** function was implemented in the Motif and Win32 Clipboard drivers.
- The **cdRegisterCallback** function was added to allow the customization of the **cdPlay** function's behavior.

- The **wdVectorTextTransform** function allows a 2D transformation applied to any vector text.
- Now the Simulation driver has several levels of simulation, and the simulation was improved with pattern and clipping simulation.

Version 3.0

- The library's architecture underwent several changes. The function tables are no longer public, only the drivers know its structure. This means that we have eliminated the need to recompile applications that use the dynamic library when we change something in the function table. There are some other benefits, like the fact that the Windows DLL can now be implemented and that it is more simple to explain the library to new users, so they will not be confused by the `cdprivat.h` header.
- Corrections to the text alignment of the **wdVectortext** function were made.
- Memory allocation of the **cdPattern** and **cdStipple** functions in the basic Windows driver was corrected.
- Memory release of the **cdKillCanvas** function in the basic Windows driver was corrected.
- The **cdPattern** function was implemented in the Postscript driver, and the **cdPutImageRGB** and **cdPutImageMap** functions now write color images.
- The **cdPattern** function was corrected in the basic X-Windows driver for use with clipping.
- The compiler directive `#include<malloc.h>` was changed to `#include<stdlib.h>` in several modules for better compatibility with other compilers.
- The **cdPlay** function now accepts the viewport rectangle where the drawing will be rendered.
- Several navigation changes were made to the user guide pages.
- A **new** CD_SIMULATE driver was created. Use it to replace some primitives and to handle attributes of another driver. It can be used with any other driver. Basically, it substitutes the interior style of dependent primitives: box, sector and filled polygons. It also substitutes the clipping methods of these primitives.
- The Windows DLL version of the library was created.

Version 2.2.1

- Interrnal macros that affect **wdArc** and **wdSector** were corrected.
- The CGM driver now supports some client image functions.
- Hatch styles in the Image RGB driver were corrected.

Version 2.2.0

New Functions:

cdVersion - returns the current library version.
cdNativeFont - sets a native font.
cdPutImageRect - same as **cdPutImage** but you may specify a part of the image.
cdPutImageRGBA - **cdPutImageRGB** with transparency.
wdFont - **cdFont** for the WD client, the size parameter is in millimeters.

New Drivers:

NativeWindow - now the library can work with other Interface libraries.
DGN - MicroStation Design File.
EMF - Windows Enhanced Metafile.
CD Metafile - our own metafile.
Client Image RGB - now you can write in an RGB image.

- **DGN**, **CGM** and **DXF** file-based drivers now have a default size in pixels (**INT_MAX** = 2.147.483.647) and are optional. In fact the size is irrelevant in these file formats. The **cdCreateCanvas** data string may contain the size in millimeters and the resolution in pixels per millimeters. Both are real values. The default resolution is 1.
- The **cdPlay** function now works on the CGM and on the CD Metafile drivers.
- The interior style attributes were implemented in the **CGM** driver.
- On the Clipboard driver: limitations are considered if creating a WMF under Win32s.
- Now the Printer Driver shows the Printer's Dialog Box (Win32 & Mac) if the parameter "-d" is specified.
- On the PS driver: all the dimensions in the Data parameter string are now in millimeters.
- On the WMF driver: several functions were disabled because of WMF limitations. Picture size was corrected.
- On the basic X-Windows driver: **cdLineWidth**`(1)` uses `width 0` for better performance. Stipple was being incorrectly interpreted. **cdGetImageRGB** was swapping red and blue channels on true color canvas.

- The clipping region now can be a polygon on some systems/drivers (Win32, Mac, X-Win and PS). Use **cdClip**(CD_CLIPPOLYGON) to use the polygon defined by a **cdBegin**(CD_CLIP), **cdVertex**(...), **cdEnd**() sequence.
- The functions **wdMM2Pixel** and **wdPixel2MM** became **cdMM2Pixel** and **cdPixel2MM**, respectively.
- Minor bugs in the **wdFontDim**, **wdLineWidth** and **wdMarkSize** functions were corrected.
- **wdVectorCharSize** now returns the previous value.

Up to Version 2.1

- The **cdActiveCanvas**, **cdPlay** and the **wdVectorFont** functions were added, and the **cdKillCanvas** function was corrected when destroying the current canvas.
- The **cdMark** function no longer depends on line style, line width and interior style attributes, and it is the same for all drivers because it is implemented only with CD functions.
- The **wdLineWidth** and **wdMarkSize** functions now use millimeters.
- The functions **cdEncodeColor** and **cdDecodeColor** now can be called without an active canvas. The DXF driver was added.
- WD can now access files with vector font definitions.
- Minor bugs in the **wdTextSize** function were corrected.

To Do's

CD

- A new SVG driver.

MAC

- Build a native Mac OS X driver using Quartz 2D.
- Macintosh Picture (PICT) file.

X-WIN

- A GDK driver in Linux.
- Xp: X Printer Extension driver
- XShm: Double Buffering and MIT-Shared Memory extensions for server images ?
- XIE: X Imaging Extensions ?
- XComposite, XRender and XFullScreen libraries from www.freedesktop.org ?

Simulation

- cdChord primitive
- ERROR in the Sector simulation.
  The error occurs sometimes when the drawing of the round part of the sector begins, because the algorithm skips an horizontal line.
- Enhance mark and text clipping.
- Implement a polygon clipping for text.
- Implement line styles for lines with thickness over 1.
- Anti-Aliasing option when possible.
- Implement winding fill mode for polygons.
- Implement line cap and line join styles.

ImageRGB

- Support for Regions, Anti-aliased lines, Alpha in color coding. Rewrite simulation primitives with a more modular and flexible approach.
- Polygon clipping is not performed for image functions.

PS

- Allow functions **cdPutImageMap...** to be implemented using *indexed color space*.
- Check the possibility of **cdHatch** and **cdStipple**, which are always opaque, having transparency, using *shading* from Version 3 or *mask images*.Same for **cdPutImageRGBA**.

Not likely to be updated anymore, although they are still supported.

DXF

- Implement World Coordinate functions directly in the driver.
- Implement Arch and Sector functions as DXF primitives, and not as polygons. Update all other primitives according to the new DXF manual, as there are several limitations in the current implementation.

CGM

- Make **cdPlay** treat the possibility of `xmax` and `ymax` being 0.
- Check the possibility of implementing function **cdTextOrientation**.
- Implement World Coordinate functions directly in the driver.
- Correct the **cdPlay** function, which is generating several extra lines.
- Correct the **cdPlay** function, which should not preserve the aspect ratio.
- Allow **cdPutImageRGBA** to be partially implemented using *transparent cell color*.

DGN

- Improve the driver using the DGNlib third party library.
- Implement the interior style attributes: *hatch*, *stipple* and *pattern*. They depend on the new DGN specification, which we do not have yet.
- Check the possibility of implementing functions **cdTextOrientation** and **cdRect**.
- Implement World Coordinate functions directly in the driver.
- Correct function **cdKillCanvas**, which generates "*assertion failed*" when the library is used with debug information and the Seed file is not included.

Comparing CD with Other Graphics Libraries

There are other graphics libraries, with some portability among operational systems, available on the Internet. Among them we can highlight:

- **VOGL** - A Very Ordinary GL-Like Library. It is very similar to the CD library, but it has no longer been updated since 1995. It has several drivers, 2D and 3D routines, and illumination. http://www.cs.kuleuven.ac.be/~philippe/vogl/.
- **SRGP** - Based on Foley's book, the code has not been found. It is aimed only at display. http://www.micg.et.fh-stralsund.de/~pohlers/srgp.html.
- **GGI** - 2D graphics library aimed only at display. http://www.ggi-project.org/.
- **GKS** - Very complete 2D and 3D graphics library, but with limited image resources. It is an ISO standard, and it implementations are usually commercial. Tecgraf has an implementation of GKS which is no longer used, being replaced by CD. http://www.bsi.org.uk/sc24/.
- **Mesa** - 3D graphics library with support to the OpenGL standard. Implemented in C. Aimed only at display, with attribute functions for illumination and shading features. http://www.mesa3d.org/.
- **OpenGL** - 3D graphics library with some 2D support. Aimed only at display. A window CD canvas can coexist with an OpenGL canvas at the same time. Note: When Double Buffer is used, do not forget to swap buffer before redrawing with the CD library. http://www.opengl.org.
- **GD** - Library only for drawing on images, saves PNG files. Implemented in C. http://www.boutell.com/gd/.
- **GDK** - Used by the GTK user interface toolkit. Implemented in C. Aimed only at display, and contains several functions for managing windows, keyboard and mouse. http://www.gtk.org/.
- **CAIRO** - A vector graphics library designed to provide high-quality display and print output. Very interesting, lots of functions, seems to render in bitmaps on native systems. http://cairographics.org/.

Most of them are aimed only at one type of driver, usually display or images, and sometimes user interface routines were also included. Others add 3D drawing routines, as well as scene illumination routines. All this

unnecessarily increases their complexity and does not make them more complete as 2D graphic libraries.

There are also several Graphics User Interface libraries that contain drawing functions.

As to performance, CD is as good as any other, in some cases having a better performance. Thus, the CD library offers unique features and quality as a portable 2D graphic library.

Guide

Getting Started

The CD library is a basic graphic library (GL). In a GL paradigm you use **primitives**, which have **attributes**, to draw on a **canvas**. All the library functions reflect this paradigm.

The **canvas** is the basic element. It can have several forms: a paper, a video monitor, a graphic file format, etc. The virtual drawing surface where the canvas exists is represented by a **driver**. Only the driver knows how to draw on its surface. The user does not use the driver directly, but only the canvas.

To make the library simple we use the concept of an active canvas, over which all the primitives are drawn. This also allows the use of an expansion mechanism using function tables. Unfortunately if a function is called without an active canvas a memory invasion will occur. On the other hand, the mechanism allows the library to be expanded with new drivers without limits.

The **attributes** are also separated from the primitives. They reside in the canvas in a state mechanism. If you change the attribute's state in the canvas all the primitives drawn after that canvas and that depend on the attribute will be drawn in a different way.

The set of **primitives** is very small but complete enough to compose a GL. Some primitives are system dependent for performance reasons. Some drivers (window and device based) use system functions to optimally implement the primitives. Sometimes this implies in a in small different behavior of some functions. Also some primitives do not make sense in some drivers, like server images in file-based drivers.

The set of available functions is such that it can be implemented in most drivers. Some drivers have sophisticated resources, which cannot be implemented in other drivers but can be made available using a generic attribute mecanism.

Building Applications

All the CD functions are declared in the `cd.h` header file; World Coordinate functions are declared in the `wd.h` header file; and each driver has a correspondent header file that must be included to create a canvas. It is important to include each driver header after the inclusion of the `cd.h` header file.

To link the application you must add the **cd.lib/libcd.a/libcd.so** library to the linker options. If you use an IUP Canvas then you must also link with the **cdiup.lib/libcdiup.a/libcdiup.so** library.

In UNIX, CD uses the Xlib (`X11`) libraries. To link an application in UNIX, add also the `"-lX11"` option in the linker call.

The download files list includes the Tecgraf/PUC-Rio Library Download Tips document, with a description of all the available binaries.

Building the Library

The easiest way to build the library is to install the Tecmake tool into your system. It is easy and helps a lot. The Tecmake configuration files (*.mak) available at the "src" folder are very easy to understand also.

Tecmake is a command line multi compiler build tool available at http://www.tecgraf.puc-rio.br/tecmake. Tecmake is used by all the Tecgraf libraries and many applications.

In **CD**'s main source directory there is a file named *make_uname* (*make_uname.bat* in Windows) that build the libraries using **Tecmake**. To build the **CD** libraries for Windows using Visual C 7.0 for example, just execute *make_uname.bat vc7* in the source folder.

But we also provide a stand alone makefile for Linux systems and a Visual Studio workspace with the respective projects. The stand alone makefile is created using Premake and a configuration file in lua called "premake.lua".

The library does not impose any specific compiler directive. Therefore, using it with the default compiler options would be enough to make things work. The library always has a static linking module, but on some platforms a dynamic linking library is also available. To compile the library it is necessary to define the symbol "__CD__". Internaly we use the definitions: "WIN32" and "SunOS", that must be defined in the respective systems.

Environment Variables

`CDDIR` - This environment variable is used by some drivers to locate useful data files, such as font definition files. It contains the directory path without the final slash.
`CD_QUIET` - In UNIX, if this variable is defined, it does not show the library's version info on *sdtout*.
`CD_XERROR` - In UNIX, if this variable is defined, it will show the X-Windows error messages on *sdterr*.

Implementing a Driver

The best way to implement a new driver is based on an existing one. For this reason, we provide the code of the simplest driver in the library, CD_METAFILE. This code is well commented on, in order to make this process easy and to elliminate some doubts. See CDXX.H and CDXX.C.

Also see topic "Internal Architecture" in this user guide.

Intercepting Primitives

To fill data structures of library primitives during a `cdPlay` call you must implement a driver and activate it before calling `cdPlay`. Inside your driver primitives you can fill your data structure with the information interpreted by the `cdPlay` function.

IUP Compatibility

The **IupCanvas** element of the IUP interface toolkit can be used as a visualization surface for a CD canvas. There are two moments in which one must be careful when an application is using both libraries: when creating the CD canvas, and when changing the size of the IUP canvas.

Creating the CD Canvas

The creation of the CD canvas must be made always after the **IupCanvas** element has been mapped in the system's native control. This happens when the application calls function `IupShow` or when the function `IupMap` is explicitally called.

Since a call to `IupShow` generates a call to the ACTION callback of the IUP canvas, we have a peculiar situation. The CD canvas cannot be created before `IupShow`, but if it is created after it one cannot draw on the first time the redrawing callback of the IUP canvas is called.

We can address this problem in several ways:

- We can force the mapping prior to `IupShow` by calling the `IupMap` function before creating the CD canvas.
- We can create the CD canvas after `IupShow`, but associating the canvas' redrawing callback also after `IupShow` and forcing a call to this function.
- We can create the CD canvas during the redrawing callback or during the size change callback, which is also called during a `IupShow`.

- We can create the canvas during the **MAP_CB** callback, which is called after the **IupCanvas** element has been mapped in the native control.

Any of the above solutions works perfectly. The most elegant solution seems to be the one that uses the **MAP_CB** callback.

Creating the CD canvas also requires some parameters to be passed to the Native Window driver. These parameters are obtained from the IUP canvas by means of the `CONID` attribute. Therefore, the canvas creation is:

```
myCdCanvas = cdCreateCanvas(CD_NATIVEWINDOW,
IupGetAttribute(myIupCanvas, "CONID"));
IupSetAttribute(myIupCanvas, "_CD_CANVAS",
myCdCanvas);
```

The **CD_IUP** driver can still be used, but it must be linked with the **cdiup** library.

Resizing the IUP Canvas

If the application always activates the canvas before drawing, even if it is already active, then it is not necessary to worry about this situation. If this is not so, then the CD canvas mut be activated in the IUP canvas resize callback.

Internal Architecture

Modularity

Apart from the several drivers, the CD library is composed of a few modules, the public header files **cd.h** and **wd.h**, those which implement the functions independently from drivers, **cd.c** and **wd.c**, and the header file **cdprivat.h**, apart from some other modules which implement non-exported specific functions. Such modules are totally independent from the implemented drivers, as well as every driver independs from one another, unless there is an intentional dependency.

Linking

Since the drivers independ from one another, we could create a library for each of them. For the drivers provided with CD it was easy to include them in their own library, thus simplifying the application's linking process. Note: Internally, the drivers are called "context".

In order to establish this dependency, when creating a canvas in a given driver the user must specify the driver to be used. This specification is done by means of a macro which is actually a function with no parameter, which passes the function table from that driver to the canvas creation function. For instance:

```
CD_PS => cdContext* cdContextPS();
cdCreateCanvas(CD_PS, "teste.ps"); => novo_canvas->cdLine = context->cdLine
```

Since each primitive is called without the canvas as a parameter, an active canvas is assumed. That is, after the canvas is created it must be activated in order to be used. Therefore, we have to internally maintain an object containing the active canvas.

When there is a call to a function, the active canvas function is called, which is actually the driver function from where that canvas was created. For example:

```
cdLine => canvas_ativo->cdLine => context->cdLine
```

Structures

The control part has only 3 structures. Two of them are private structures which correspond to the **cd.h** public structures: `cdPrivateContext` and `cdPrivateCanvas`. The other one is only intern, used for the WD functions: `wdCanvas`. The `cdPrivateContext` structure is used only by the drivers;

`cdPrivateCanvas` contains the function table and the values of all attributes which can be queried.

The drivers need not implement all functions from the function table, because those which are not implemented for not making sense in that driver will not run without generating an error for the user.

Each driver has a private internal canvas structure. Apart from this structure, they have to define the `cdContext` structure to be returned by function `cdContextXX()` (where `XX` varies according to the driver, as described previously). Thus the drivers must also define a `cdPrivateContext` structure to be included into this driver's `cdContext` structure.

The table below illustrates this mechanism:

```
typedef struct _cdContext
{
  void *ctx;   ============>
} cdContext;
```

```
typedef struct _cdPrivateContext
{
  void* (*CreateCanvas)(cdCanvas* canvas, void *data);
  int  (*Play)(int xmin, int xmax, int ymin, int ymax,
  int  (*RegisterCallback)(int cb, cdCallback func);
} cdPrivateContext;
```

```
typedef struct _cdCanvas
{
  void *cnv;   ============>
} cdCanvas;
```

```
typedef struct _cdPrivateCanvas
{
  ...
  void (*Line)(int x1, int y1, int x2, int y2);
  void (*Rect)(int xmin, int xmax, int ymin, int ymax)
  void (*Box)(int xmin, int xmax, int ymin, int ymax);
  ...

  ...
  int mark_type, mark_size;
  int line_style, line_width;
  int interior_style, hatch_style;
  ...

  void* wd_canvas;       ============>  wdCanvas
  void* context_canvas;  ============>  cdCanvasXX
  void* context;         ============>  cdContext
} cdPrivateCanvas;
```

To simplify driver administration, the context structure's linking is done as follows:

```
/* In the header file */
#define CD_METAFILE cdContextMetafile()
cdContext* cdContextMetafile(void)


/* In the implementation file */
static cdPrivateContext private_context_metafile =
{
  cdMFcreatecanvas,
  cdMFplay,
  cdMFregistercallback
};

static cdContext context_metafile =
{
  &private_context_metafile
};

cdContext* cdContextMetafile(void)
{
  return &context_metafile;
}
```

In CDLua, in order to create a new driver, one must use the internal function `cdluaAddContext`. All predefined drivers are also included this way. A static structure `cdContextLUA` is created containing all necessary information, and the driver initialization function, as in `cdluaiup_open`, calls function `cdluaAddContext` and registers values, if necessary. See, for instance, CDLUAMF.C, which illustrates this process.

Attributes

The query mechanism of an attribute is done still in the control part and does not reach the driver. That is, the drivers do not need to focus on the query mechanism and can use `cdPrivateCanvas` relative to the active canvas at any moment. Due to this fact, the attributes which are modified several times for the same value are not updated in the drivers, thus saving processing. Similarly, if an attribute modification in a driver was not successful, its value is not updated. Nevertheless, the fact that a driver does not implement the attribute's modification function does not mean that it rejects that attribute - the driver just does not need to do anything with this attribute on that moment and will query it later, before drawing the primitive.

The creation of customized attributes for each driver is made generically, using string-like attributes. A structure with the attribute's name and its *set* and *get* functions must be declared, as in the example below:

```
static void set_fill_attrib(char* data)
{
  CurrentCanvas->fill_attrib[0] = data[0];
}

static char* get_fill_attrib(void)
{
  return CurrentCanvas->fill_attrib;
}

static cdAttribute fill_attrib =
{
  "SIMPENFILLPOLY",
  set_fill_attrib,
  get_fill_attrib
};
```

At *createcanvas* in the driver:

```
new_canvas->fill_attrib[0] = '1';
new_canvas->fill_attrib[1] = 0;

cdRegisterAttribute(private_canvas, &fill_attrib);
```

, for instance, must exist, thus initializing the attribute and registering it in the canvas' attribute list.

Nomenclature

All directly or indirectly public functions, variables or numberings have the prefix "CD" ("cd" for functions and variables, and "CD_" for numberings). For drivers, one must add, after the prefix, a one- or two-letter identification relative to the driver (ex.: "cdps" for the Postscript driver). The same rule applies for the modules; modules from different platforms have more letters added to the prefix indicating the platform (ex.: "cdwiup", IUP driver in Winndows). This helps solving conflicts for RCS, the version control program.

There are no global variables in the control part, but some drivers, due to historical reasons, may contain some global variables, which attempt to follow this same nomenclature and should not cause any problem.

History

The first version of the library used macros to replace function names by pointers in a function pointer structure, that is, a function table which was indirectly visible to the user. This caused several problems, especially when using the dynamic library and due to the fact that it was not by any means possible to control the case of a non-active canvas. Indirectly, the users also depended on the header file `cdprivat.h`.

We have then decided to hide the function table, creating an intermediate level between the public functions and the function table. However, the drivers still knew the table's structure, therefore a change in the table meant the need to modify all drivers.

Thus, once again, the library's structure was modified so that the drivers would become independent from the function table. With this change we were able to create a method for replacing any primitive in a given driver by the simulation driver's primitive, as well as to allow the user to change any driver primitive to a specified primitive.

Together with these new changes, we approached a problem related to the WD functions, which were necessarily clients of the current drivers. This generated some precision losses and unnecessary conversions. Including the WD functions in the function table and using the already implemented WD functions as default functions for the drivers with no WD functions, we corrected this problem, because wherever it is possible to implement these functions the primitives are more precisely executed, and can be called from the `cdPlay` function when interpreting a metafile.

Sample Codes

Simple Draw

This is an example of a simple drawing program using a IUP canvas:

```
cdCanvas* Canvas = cdCreateCanvas(CD_NATIVEWINDOW, IupGetAttribute
(IupCanvas,"CONID"));
cdActivate(Canvas);
cdLineStyle(CD_DASHED);
cdLine(0, 0, 100, 100);
cdKillCanvas(Canvas);
```

If you want to use **World Coordinates**:

```
cdCanvas* Canvas = cdCreateCanvas(CD_NATIVEWINDOW, IupGetAttribute
(IupCanvas,"CONID"));
cdActivate(Canvas);
wdViewport(0, 100, 0, 100);
wdWindow(-1.5, 1.5, -3000, 3000);
cdLineStyle(CD_DASHED);
wdLine(-0.5, -500, 1.0, 1000);
cdKillCanvas(Canvas);
```

Off Screen Drawing (Double Buffering)

To draw in the background and later on transfer the drawing to the screen, use:

```
cdCanvas* Canvas = cdCreateCanvas(CD_NATIVEWINDOW, IupGetAttribute
(IupCanvas,"CONID"));
cdActivate(Canvas);
void* Image = cdCreateImage(100, 100);
cdCanvas* ImageCanvas = cdCreateCanvas(CD_IMAGE, Image);
cdActivate(ImageCanvas);
cdLineStyle(CD_DASHED);
cdLine(10, 10, 50, 50);
cdActivate(Canvas);
cdPutImage(Image, 0, 0);
cdKillImage(Image);
cdKillCanvas(ImageCanvas);
cdKillCanvas(Canvas);
```

For a more easier use of double buffering see the driver CD_DBUFFER.

To draw in a RGB image, use:

```
unsigned char* red = malloc(width * height);
unsigned char* green = malloc(width * height);
unsigned char* blue = malloc(width * height);

cdCanvas* canvas = cdCreateCanvasf(CD_IMAGERGB, "%dx%d %p %p %p", width, height,
cdActivate(canvas);
....
cdLineStyle(CD_DASHED);
cdLine(10, 10, 50, 50);
...
cdKillCanvas(canvas);
...
free(red);
free(green);
free(blue);
```

Lua Samples

To draw in a RGB image in CDLua for Lua 3 (old names):

```
canvas_buf=cdCreateImageRGB(200,200)
canvas=cdCreateCanvas(CD_IMAGERGB,canvas_buf)
cdActivate(canvas)
cdFont(CD_TIMES_ROMAN, CD_BOLD, 8)
cdText(10, 10, "Test")
cdKillCanvas(canvas)
```

and in CDLua for Lua 3 (new names) or Lua 5:

```
canvas_buf=cd.CreateImageRGB(200,200)
canvas=cd.CreateCanvas(cd.IMAGERGB,canvas_buf)
cd.Activate(canvas)
cd.Font(cd.TIMES_ROMAN, cd.BOLD, 8)
cd.Text(10, 10, "Test")
cd.KillCanvas(canvas)
```

Check the file samples_cdlua3.zip or samples_cdlua5.zip for several samples in Lua and samples in C to test them. You will need the CD, IUP and Lua libraries to compile and link the applications.

Screen Capture in Windows

Using a NULL parameter to the NATIVEWINDOW driver you can get access to the entire screen:

```
cdCanvas *cd_canvas = cdCreateCanvas(CD_NATIVEWINDOW, NULL);
cdActivate(cd_canvas);

cdGetCanvasSize(&width, &height, NULL, NULL);
// allocate red, green and blue pointers

cdGetImageRGB(red, green, blue, 0, 0, width, height);

cdKillCanvas(cd_canvas);
```

Complete Example

We have created an application called Simple Draw that illustrates the use of all functions in the CD library (including WD). You can see the source code in the simple.c file, or take the file simple.zip for a complete set of files including makefiles for all platforms. Extract the files creating subfolders, using parameter "-d".

Example for Tests

The CDTEST example is actually one of the applications used to test virtually all functions of the CD library. Its interface uses the IUP library, and it can run in several platforms. You can take either the .EXE files or the source code. Extract the files creating subfolders, using parameter "-d". Warning: This application is not didactic.

Lua Binding

Overview

CDLua was developed to make all functionalities of the CD library available to Lua programmers. To use the CDLua bindings, your executable must be linked with the CDLua library, and you must call the initialization function **cdlua_open** declared in the header file **cdlua.h**, as seen in the example below:

in Lua 3	in Lua5

`#include <lua.h>`
`#include <lualib.h>`
 	
`#include <cdlua.h>`
 	
`void main(void)`
`{`
`  lua_open();`
 	
`  iolib_open();`
`  strlib_open();`
`  mathlib_open();`
 	
`  cdlua_open();`
 	
`  lua_dofile("myprog.lua");`
 	
`  cdlua_close();`
`  lua_close();`
`}`
	`#include <lua.h>`
	`#include <lualib.h>`
	`#include <lauxlib.h>`
	`#include <cdlua.h>`
	
	`void main(void)`
	`{`
	`  lua_State *L = lua_open();`
	
	`  luaopen_string(L);`
	`  luaopen_math(L);`
	`  luaopen_io(L);`
	
	`  cdlua_open(L);`
	
	`  lua_dofile("myprog.lua");`
	
	`  cdlua_close(L);`
	`  lua_close(L);`
	`}`

The **cdlua_open()** function registers all CD functions and constants your Lua program will need. The use of the CDLua functions in Lua is generally identical to their equivalents in C. Nevertheless, there are several exceptions due to differences between the two languages. Notice that, as opposed to C, in which the flags are combined with the bitwise operator OR, in Lua the flags are added arithmetically.

As in C, it is important to make sure there is a currently active canvas before calling any CDLua function. Otherwise, the Lua program will be aborted with an error message. Canvases are created and activated as in C:

```
cnv = cd.CreateCanvas(cd.IUP, "test.ps")
if cnv == nil then
  -- deal with error
  ...
else
  cd.Activate(cnv)
end
```

The CDLua dynamic libraries are also compatible with the Lua "require" function.

Function Names and Definitions

In Lua, because of the name space "cd" all the functions and definitions have their names prefix changed. The general rule is quite simple:

```
cdXxx  -> cd.Xxx
wdXxx  -> cd.wXxx
CD_XXX -> cd.XXX
```

In Lua 3, because of backward compatibility the old names are also available.

Data Types

Lua native data types were not enough for the CDLua bind, because CD deals with a series of data types other than *strings* and *numbers*. Therefore, to provide the eficiency required, *usertag*s or *metatables* were used to implement such data types. They are:

"CDLUA_COLOR_TAG" (color_tag): type used by functions `cdEncodeColor`, `cdDecodeColor`, `cdPixel`, `cdForeground` and `cdBackground`. Except in Lua 5 where the **`color_tag`** is a light user data.
"CDLUA_IMAGEEX_TAG" (**`imageex_tag`**): type used by functions `cdCreateImageEx`, `cdKillImageEx`, `cdGetImageEx`, `cdPutImage` and `cdPutImageEx`.
"CDLUA_IMAGE_TAG" (**`image_tag`**): type used by functions `cdCreateImage`, `cdKillImage`, `cdGetImage`, `cdPutImage` and `cdPutImage`.
"CDLUA_IMAGERGB_TAG" (**`imagergb_tag`**): type used by functions `cdCreateImageRGB`, `cdKillImageRGB`, `cdGetImageRGB` and `cdPutImageRGB`.
"CDLUA_IMAGERGBA_TAG" (**`imagergba_tag`**): type used by functions `cdCreateImageRGBA`, `cdKillImageRGBA` and `cdPutImageRGBA`.
"CDLUA_IMAGEMAP_TAG" (**`imagemap_tag`**): type used by functions `cdCreateImageMap`, `cdKillImageMap` and `cdPutImageMap`.
"CDLUA_PALETTE_TAG" (**`palette_tag`**): type used by functions `cdCreatePalette`, `cdKillPalette`, `cdPalette` and `cdPutImageMap`.
"CDLUA_PATTERN_TAG" (**`pattern_tag`**): type used by functions `cdCreatePattern`, `cdKillPattern` and `cdPattern`.
"CDLUA_STIPPLE_TAG" (**`stipple_tag`**): type used by functions `cdCreateStipple`, `cdKillStipple` and `cdStipple`.
"**CDLUA_CHANNEL_TAG**" (**`channel_tag`**): type used by channel image access methods. Internal use only.
"CDLUA_CANVAS_TAG" (**`canvas_tag`**): type used by `cdCreateCanvas`, `cdActivate`, `cdKillCanvas`, `cdGetContext` and `cdContextCaps`.
"CDLUA_STATE_TAG" (**`state_tag`**): type used `bycdSaveState` and `cdRestoreState`.

All the *usertags* are available to the C programmer in Lua 3 global variables or Lua 5 metatables with the given names. The respective structures are defined in the **`cdluapvt.h`** header.

The structure `cdContext` is stored as a number, and did not need a *usertag*.

New Functions

New functions (without equivalents in C) were implemented to create and to destroy objects of the new data types. Functions were developed to create and remove images, pattern, stipple and palette.

In case of error (for example, lack of memory), the creation functions return `nil`. The user must verify its return value. See the example below:

```
pattern = cd.CreatePattern(16, 16)
if pattern == nil then
  ...
end
```

Modified Functions

Some functions were modified to receive parameters of the types `imagergb_tag`, `imagergba_tag`, `imagemap_tag`, `palette_tag`, `stipple_tag` and `pattern_tag`. These objects already have their dimensions stored internally and, therefore, the user does not need to pass them as parameters.

The functions which receive values by referencing to C were also modified in Lua. Generally, the values of parameters that would have their values modified are now returned by the function in the same order.

The functions still have the same functionality, but they are now used differently:

```
w, h = cd.GetCanvasSize()
red, green, blue = cd.DecodeColor(CD_DARK_MAGENTA)
x, y = cd.Canvas2Raster(x, y)
```

Garbage Collection

All the objects are garbage collected by the Lua garbage collector, with the execption of the Canvas. This means that all the `cd.Kill*` functions are optional, with the exception of the `cd.KillCanvas`.

Exchanging Values between C and Lua

Because of some applications that interchange the use of CD canvases in Lua and C, we build a simple C function to retreive the pointer of a CD canvas created in Lua: **`cdlua_getcanvas`**. It is declared in the **`cdlua.h`** header. The canvas to be retreived must be in the Lua stack before calling this function. Usually you will do a `lua_pushobject(lua_getglobal("mycanvas"))` call before it.

Error Checking

Rigorous parameter checking is performed by CDLua functions before passing the parameters to CD. When an error is considered fatal, the library interrupts the Lua program and shows an explanatory error message. Errors in the number and types of parameters, and inconsistency in the values of parameters are considered as fatal errors. In general, fatal errors are those that require a change in the Lua code and would cause an equivalent C program to crash.

All fatal errors result in a call to `lua_error` with a message with format "*function:* message", where *function is* the name of the function that detected the fatal error and *message* is a message that identifies the error. Some of the most important errors are seen in the examples below:

```
"cdlua: there is no active canvas!"
"cdActivate: attempt to activate a killed canvas!"
"cdCreateCanvas: unknown driver!"
"cdKillStipple: attempt to kill a killed stipple!"
"cdCreateCanvas CD_IUP: data should be an iuplua canvas object!"
"cdCreateCanvas CD_IMAGE: data is a NIL image!"
"cdCreateCanvas CD_PRINTER: data should be of type string!"
"cdPlay: CD_SIZECB: invalid return value!"
"cdCreateCanvas CD_SIMULATE: too many parameters!"
"cdRegisterCallback: unknown callback!"
"cdPlay: driver does not implement the play function or unknown driver!"
"cdKillCanvas: attempt to kill a NIL canvas!"
"cdEncodeColor: color components values should be in range [0, 255]!"
"cdCreateStipple: stipple dimensions should be positive integers!"
```

The fallbacks used to modify and to check the values of types `imagergb_tag, imagergba_tag, imagemap_tag, palette_tag, pattern_tag` and `stipple_tag` also perform error checking and can detect fatal errors. The methods check for vector bounds, and for parameter types and values. The messages have the format "*type* '*fallback':* message", as seen below:

```
"pattern_tag "settable": index should be a number!"
"imagemap_tag "gettable": index out of bounds!"
"stipple_tag "settable": value should belong to {0, 1}!"
"pattern_tag "settable": value should be of type color_tag!"
```

Integration with IMLUA

In IMLUA there is an additional library providing simple functions to map the `imImage` structure to the `cdBitmap` structure. And some facilities to draw an image in a CD canvas. See the IM documentation.

Color values and palettes can be created and used transparently in both libraries. Palettes and color values are 100% compatible between CD and IM.

Canvas

The canvas represents the drawing surface. It could be anything: a file, a client area inside a window in a Window System, a paper used by a printer, etc. Each canvas has its own attributes.

Initialization

You must call `cdCreateCanvas` to create a canvas, and `cdKillCanvas` when you do not need the canvas anymore. You must activate a canvas using `cdActivate`. If you call a function without an active canvas a memory invasion will occur. The `cdActiveCanvas` function returns the currently active canvas. You can use this function to retreive the active canvas before activating your own, so you can restore it after drawing on your canvas.

To know if a feature is supported by a driver, use function `cdContextCaps` or see the driver's documentation.

Control

Some canvases are buffered and need to be flushed; for that, use the `cdFlush` function. In some drivers, this function can also be used to change to another page, as in drivers `CD_PRINTER` and `CD_PS`.

You can clear the drawing surface with the `cdClear` function, but in some drivers the function may just draw a rectangle using the background color.

Coordinate System

You may retrieve the original canvas size using the `cdGetCanvasSize` function. The canvas' origin is at the bottom left corner of the canvas, but an origin change can be simulated with function `cdOrigin`. Usually user interface libraries have their origin at the upper right corner, oriented top down. In this case, the function `cdUpdateYAxis` converts the Y coordinate from this orientation to CD's orientation and vice-versa.

Other

Some canvas contents can be interpreted; the `cdPlay` function interprets the contents of a canvas and calls library functions for the contents to be displayed in the active canvas.

Canvas Initialization

`cdCanvas *cdCreateCanvas(cdContext* ctx, void *data); [in C]`
`cd.CreateCanvas(ctx: `*`number`*`, data: `*`string`*` or `*`userdata`*`) -> (canvas: `*`canvas_tag`*`) [in Lua]`

Creates a CD canvas for a virtual visualization surface (VVS). A VVS may be the canvas of a user-interface window, the page of a document sent to a printer, an offscreen image, the clipboard, a metafile, and so on. To create the canvas, it is necessary to specify the driver in which each canvas is implemented.

The driver is set by the **`driver`** variable with additional information provided in the **`data`** parameter. Even though it is possible to create more than one canvas with the same **`driver/data`** pair, this is not recommended, and its behavior is not specified. Each canvas maintains its own features.

In case of failure, a **`NULL`** value is returned. The following predefined drivers are available:

Window-Base Drivers

- **`CD_IUP`** = IUP Canvas (**`cdiup.h`**).
- **`CD_NATIVEWINDOW`** = Native Window (**`cdnative.h`**).

Device-Based Drivers

- **`CD_CLIPBOARD`** = Clipboard (**`cdclipbd.h`**).

- **CD_PRINTER** = Printer (**cdprint.h**).

Image-Based Drivers

- **CD_IMAGE** = Server-Image Drawing (**cdimage.h**).
- **CD_IMAGERGB** = Client-Image Drawing (**cdirgb.h**).
- **CD_DBUFFER** = Offscreen Drawing (**cddbuf.h**).

File-Based Drivers

- **CD_CGM** = Computer Graphics Metafile ISO (**cdcgm.h**).
- **CD_DGN** = MicroStation Design File (**cddgn.h**).
- **CD_DXF** = AutoCad Drawing Interchange File (**cddxf.h**).
- **CD_EMF** = Microsoft Windows Enhanced Metafile (**cdemf.h**). Works only in MS Windows systems.
- **CD_METAFILE** = Metafile Canvas Draw (**cdmf.h**).
- **CD_PS** = PostScript File (**cdps.h**).
- **CD_WMF** = Microsoft Windows Metafile (**cdwmf.h**).

cdCanvas* cdCreateCanvasf(cdContext *context, char* format, ...); [in C]
[There is no equivalent in Lua]

Same as **cdCreateCanvas**, used in the case that the parameter **data** is a string composed by several parameters. This function can be used with parameters equivalent to the **printf** function from the default C library.

void cdKillCanvas(cdCanvas *canvas); [in C]
cd.KillCanvas(canvas: *canvas_tag*) [in Lua]

Destroys a previously created canvas.

int cdActivate(cdCanvas *canvas); [in C]
cd.Activate(canvas: *canvas_tag*) -> (status: *number*) [in Lua]

Activates a canvas for drawing. There is no explicit **cdDeactivate**. When a new canvas is activated, the current one is deactivated (there can be only one active canvas at a given moment). The function returns a status, CD_OK or CD_ERROR, indicating whether the target canvas was successfully created or not. A NULL canvas can be activated, but any function call will cause an invalid memory access.

cdCanvas* cdActiveCanvas(void); [in C]
cd.ActiveCanvas() -> (canvas: *canvas_tag*) [in Lua]

Returns the active canvas. Returns NULL if there is no active canvas.

---

cdContext* cdGetContext(cdCanvas *canvas); [in C]
cd.GetContext(canvas: *canvas_tag*) -> (ctx: *number*) [in Lua]

Returns the context of a given canvas, which can be compared with the predefined contexts, such as "CD_PS".

int cdContextCaps(cdContext* ctx); [in C]
cd.ContextCaps(ctx: *number*) -> (caps: *number*) [in Lua]

Returns the resources available for that context. To verify if a given resource is available, perform a binary AND ('&') with the following values:

```
CD_CAP_FLUSH
CD_CAP_CLEAR
```

CD_CAP_PLAY
CD_CAP_YAXIS - The Y axis has the same orientation as the CD axis.
CD_CAP_CLIPAREA
CD_CAP_CLIPPOLY - Usually is not implemented.
CD_CAP_MARK - Marks are implemented directly in the driver (they are usually simulated).
CD_CAP_RECT - Rectangles are implemented directly in the driver (they are usually simulated).
CD_CAP_VECTORTEXT - Vector text is implemented directly in the driver (it is usually simulated).
CD_CAP_IMAGERGB
CD_CAP_IMAGERGBA - If this is not implemented, but cdGetImageRGB is, then it is simulated using cdGetImageRGB and cdPutImageRGB.
CD_CAP_IMAGEMAP
CD_CAP_GETIMAGERGB
CD_CAP_IMAGESRV - Usually is only implemented in contexts of window graphics systems (Native Window and IUP).
CD_CAP_BACKGROUND
CD_CAP_BACKOPACITY
CD_CAP_WRITEMODE
CD_CAP_LINESTYLE
CD_CAP_LINEWITH
CD_CAP_WD - Functions of world coordinates are implemented directly in the driver (they are usually simulated).
CD_CAP_HATCH
CD_CAP_STIPPLE
CD_CAP_PATTERN
CD_CAP_FONT
CD_CAP_FONTDIM - If not defined, the function is implemented using an internal heuristics of the library.
CD_CAP_TEXTSIZE - If not defined, the function is implemented using an internal heuristics of the library.
CD_CAP_TEXTORIENTATION - Usually is not implemented.
CD_CAP_PALETTE - Usually is only implemented in contexts of window graphics systems (Native Window and IUP).

int cdSimulate(int mode); [in C]
cd.Simulate(mode: *number*) -> (old_mode: *number*) [in Lua]

Activates the simulation of one or more primitives and clipping for the active canvas. It is ignored for the canvas in the ImageRGB context, for in this case everything is already simulated. It also has no effect for primitives that are usually simulated. It returns the previous simulation, but does not include primitives that are usually simulated. The simulation can be activated at any moment. For instance, if a line simulation is required only for a situation, the simulation can be activated for the line to be drawn, and then deactivated.

ATTENTION: Using the clipping simulation causes some primitives to be internally simulated, such as Polygons, Arcs and Sectors. Be careful when activating the clipping simulation.

See in the Simulation sub-driver the information on how each simulation is performed.

To activate a given simulation, perform a binary OR ('|') using one or more of the following values (in Lua, the values must be added '+'):

CD_SIM_NONE - Deactivates all kinds of simulation.
CD_SIM_CLIPAREA
CD_SIM_CLIPPOLY
CD_SIM_TEXT
CD_SIM_MARK
CD_SIM_LINE
CD_SIM_RECT
CD_SIM_ARC
CD_SIM_POLYLINE
CD_SIM_BOX
CD_SIM_SECTOR

CD_SIM_POLYGON
CD_SIM_WD
CD_SIM_VECTORTEXT
CD_SIM_ALL - Activates all simulation options.
CD_SIM_LINES - Combination of CD_SIM_LINE, CD_SIM_RECT, CD_SIM_ARC and CD_SIM_POLYLINE.
CD_SIM_FILLS - Combination of CD_SIM_BOX, CD_SIM_SECTOR and CD_SIM_POLYGON.
CD_SIM_CLIP - Combination of CD_SIM_CLIPAREA and CD_SIM_CLIPPOLY.

Extras

void cdlua_open(void); [for Lua 3]
int cdlua_open(lua_State * L); [for Lua 5]

Initializes the CDLua binding. In Lua 5 the binding is lua state safe, this means that several states can be initialized any time.

void cdlua_close(void); [for Lua 3]
int cdlua_close(lua_State * L); [for Lua 5]

Releases the memory allocated by the CDLua binding.

cdCanvas* cdlua_getcanvas(void); [for Lua 3]
cdCanvas* cdlua_getcanvas(lua_State * L); [for Lua 5]

Returns the canvas in the Lua stack at index 1. The function will call lua_error if there is not a valid canvas in the stack.

Canvas Control

void cdClear(void); [in C]
cd.Clear() [in Lua]

Cleans the active canvas using the current background color. This action is interpreted very differently by each driver. Many drivers simply draw a rectangle with the current background color. It is NOT necessary to call cdClear when the canvas has just been created, as at this moment it is already clean. Most file-based drivers do not implement this function.

void cdFlush(void); [in C]
cd.Flush() [in Lua]

Has a different meaning for each driver. It is useful to send information to buffered devices and to move to a new page or layer. In all cases, the current canvas attributes are preserved.

---

cdState* cdSaveState(void); [in C]
cd.SaveState() -> (state: state_tag) [in Lua]

Saves the state of attributes of the active canvas. It does not save callbacks, polygon creation states (begin/vertex/vertex/...) and the palette.

void cdRestoreState(cdState* state); [in C]
cd.RestoreState(state: state_tag) [in Lua]

Restores the attribute state of the active canvas. It can be used between canvases of different contexts. It can be used several times for the same state.

void cdReleaseState(cdState* state); [in C]
cd.ReleaseState(state: state_tag) [in Lua]

Releases the memory allocated by the **cdSaveState** function.

---

void cdSetAttribute(char* name, char* data); [in C]
cd.SetAttribute(name, data: string) [in Lua]

Modifies a custom attribute directly in the driver of the active canvas. If the driver does not have this attribute, the call is ignored.

void cdSetfAttribute(char* name, char* format, ...); [in C]
[There is no equivalent in Lua]

Same as **cdSetAttribute**, used for the case in which the parameter **data** is a string composed by several parameters. It can be used with parameters equivalent to those of the **printf** function from the standard C library.

char* cdGetAttribute(char* name); [in C]
cd.SetAttribute(name: string) -> (data: string) [in Lua]

Returns a custom attribute from the driver of the active canvas. If the driver does not have this attribute, it returns NULL.

Coordinate System

void cdGetCanvasSize(int *width, int *height, double *width_mm, double *height_mm); [in C]
cd.GetCanvasSize() -> (width, heigth, mm_width, mm_height: number) [in Lua]

Returns the canvas size in pixels and in millimeters. You can provide only the desired values and NULL for the others.

void cdUpdateYAxis(int *y); [in C]
cd.UpdateYAxis(yc: number) -> (yr: number) [in Lua]

In some graph systems, the origin is at the upper left corner of the canvas, with the direction of the Y axis pointing down. In this case, the function converts the coordinate system of the CD library into the internal system of the active canvas' driver, and the other way round. If this is not the case, nothing happens. This is just "y = height-1 - y".

void cdMM2Pixel(double mm_dx, double mm_dy, int *dx, int *dy); [in C]
cd.MM2Pixel(mm_dx, mm_dy: number) -> (dx, dy: number) [in Lua]

Converts sizes in millimeters into pixels (canvas coordinates). You can provide only the desired values and NULL for the others.

void cdPixel2MM(int dx, int dy, double *mm_dx, double *mm_dy); [in C]
cd.Pixel2MM(dx, dy: number) -> (mm_dx, mm_dy: number) [in Lua]

Converts sizes in pixels (canvas coordinates) into millimeters. You can provide only the desired values and NULL for the others. Use this function to obtain the horizontal and vertical resolution of the canvas by passing 1 as parameter in dx and dy. The resolution value is obtained using the formula **res=1.0/mm**.

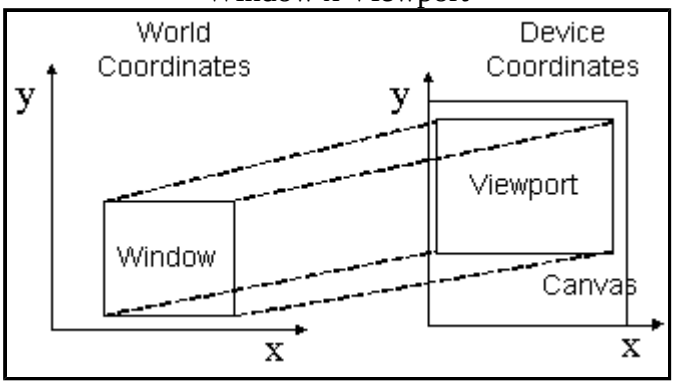
void cdOrigin(int x, int y); [in C]
cd.Origin(x, y: number) [in Lua]

Allows translating the origin - for instance, to the center of the canvas. The function profits from the architecture of the library to simulate a translation of the origin, which in fact is never actually passed to the active canvas in the respective driver. Default values: (0, 0)

World Coordinates

Allows the use of a World Coordinate System. In this system you can attribute coordinates to any unit you want. After you define a window (rectangular region) in your world, each given coordinate is then mapped to canvas coordinates to draw the primitives. You can define a viewport in your canvas to change the coordinate mapping from world to canvas. The image below shows the relation between Window and Viewport.

Window x Viewport



If you want to map coordinates from one system to another, use the `wdWorld2Canvas` e `wdCanvas2World` functions.

The quality of the picture depends on the conversion from World to Canvas, so if the canvas has a small size the picture quality will be poor. To increase picture quality create a canvas with a larger size, if possible.

`void wdWindow(double xmin, double xmax, double ymin, double ymax); [in C]`
`cd.wWindow(xmin, xmax, ymin, ymax: number) [in Lua]`

Configures a window in the world coordinate system to be used to convert world coordinates (with values in real numbers) into canvas coordinates (with values in integers). The default window is the size in millimeters of the whole canvas.

`void wdGetWindow(double *xmin, double *xmax, double *ymin, double *ymax); [in C]`
`cd.wGetWindow() -> (xmin, xmax, ymin, ymax: number) [in Lua]`

Queries the current window in the world coordinate system being used to convert world coordinates into canvas coordinates (and the other way round). It is not necessary to provide all return pointers, you can provide only the desired values.

`void wdViewport(int xmin, int xmax, int ymin, int ymax); [in C]`
`cd.wViewport(xmin, xmax, ymin, ymax: number) [in Lua]`

Configures a viewport in the canvas coordinate system to be used to convert world coordinates (with values in real numbers) into canvas coordinates (with values in integers). The default viewport is the whole canvas `(0,w-1,0,h-1)`. If the canvas size is changed, the viewport will not be automatically updated.

`void wdGetViewport(int *xmin, int *xmax, int *ymin, int *ymax); [in C]`
`cd.wGetViewport() -> (xmin, xmax, ymin, ymax: number) [in Lua]`

Queries the current viewport in the world coordinate system being used to convert world coordinates into canvas coordinates (and the other way round). It is not necessary to provide all return pointers, you can

provide only the desired values and NULL for the others.

void wdWorld2Canvas(double xw, double yw, int *xv, int *yv); [in C]
cd.wWorld2Canvas(xw, yw: *number*) -> (xv, yv: *number*) [in Lua]

Converts world coordinates into canvas coordinates. It is not necessary to provide all return pointers, you can provide only the desired values and NULL for the others.

void wdCanvas2World(int xv, int yv, double *xw, double *yw); [in C]
cd.wCanvas2World(xv, yv: *number*) -> (xw, yw: *number*) [in Lua]

Converts canvas coordinates into world coordinates. It is not necessary to provide all return pointers, you can provide only the desired values and NULL for the others.

Extra

void wdHardcopy(cdContext* ctx, void *data, cdCanvas *cnv, void(*draw_func)(void)); [in C]
cd.wHardcopy(ctx: *number*, data: *string* or *userdata*, cnv: *canvas_tag*, draw_func: *function*) [in Lua]

Creates a new canvas, activates it, prepares Window and Viewport according to the provided canvas, maintaining the aspect ratio and making the drawing occupy the largest possible area of the new canvas, calls the drawing function (which must use routines in WC) and, finally, removes the new canvas.

General Attributes

long int cdForeground(long int color); [in C]
cd.Foreground(color: *color_tag*) -> (old_color: *color_tag*) [in Lua]

Configures a new current foreground color and returns the previous one. This color is used in all primitives (lines, areas, marks and text). Default value: **CD_BLACK**. Value **CD_QUERY** simply returns the current value.

long int cdBackground(long int color); [in C]
cd.Background(color: *color_tag*) -> (old_color: *color_tag*) [in Lua]

Configures the new current background color and returns the previous one. However, it does not automatically change the background of a canvas. For such, it is necessary to call the cdClear function. The background color only makes sense for cdClear and for primitives affected by the background opacity attribute. Default value: **CD_WHITE**. Value **CD_QUERY** simply returns the current value.

int cdWriteMode(int mode); [in C]
cd.WriteMode(mode: *number*) -> (old_mode: *number*) [in Lua]

Defines the writing type for all drawing primitives. Values: **CD_REPLACE**, **CD_XOR** or **CD_NOT_XOR**. Returns the previous value. Default value: **CD_REPLACE**. Value **CD_QUERY** simply returns the current value.

Note: operation XOR is very useful, because, using white as the foreground color and drawing the same image twice, you can go back to the original color, before the drawing. This is commonly used for mouse selection feedback.

Clipping

The clipping area is an area that limits the available drawing area inside the canvas. Any primitive is drawn only inside the clipping area. It affects all primitives.

You can set the clipping area by using the function `cdClipArea`, and retrieve it using `cdGetClipArea`. The clipping area is a rectangle by default, but it can has other shapes. In some drivers a polygon area can be defined, and in display based drivers a complex region can be defined. The complex region can be a combination of boxes, polygons, sectors, chords and texts.

The `cdClip` function activates and deactivaes the clipping.

`int cdClip(int mode); [in C]`
`cd.Clip(mode: number) -> (old_mode: number) [in Lua]`

Activates or deactivates clipping. Returns the previous status. Values: **`CD_CLIPAREA, CD_CLIPPOLYGON`** or **`CD_CLIPOFF`**. The value **`CD_QUERY`** simply returns the current status. Default value: **`CD_CLIPOFF`**.

The value **`CD_CLIPAREA`** activates a rectangular area as the clipping region.

The value **`CD_CLIPPOLYGON`** activates a polygon as a clipping region, but works only in some drivers (please refer to the notes of each driver). The clipping polygon must be defined before activating the polygon clipping; if it is not defined, the current clipping state remains unchanged. See the documentation of cdBegin/cdVertex/cdEnd to create a polygon.

The value **`CD_CLIPREGION`** activates a complex clipping region. See the documentation of Regions.

`void cdClipArea(int xmin, int xmax, int ymin, int ymax); [in C]`
`void wdClipArea(double xmin, double xmax, double ymin, double ymax); (WC) [in C]`
`cd.ClipArea(xmin, xmax, ymin, ymax: number) [in Lua]`
`cd.wClipArea(xmin, xmax, ymin, ymax: number) (WC) [in Lua]`

Defines a rectangle for clipping. Only the points in the interval: *`x_min`*`<= x <=` *`x_max`*`, ` *`y_min`*` <= y <= ` *`y_max`* will be printed. Default region: (0, w-1, 0, h-1).

`int cdGetClipArea(int *xmin, int *xmax, int *ymin, int *ymax); [in C]`
`int wdGetClipArea(double *xmin, double *xmax, double *ymin, double *ymax); (WC) [in C]`
`cd.GetClipArea() -> (xmin, xmax, ymin, ymax, status: number) [in Lua]`
`cd.wGetClipArea() -> (xmin, xmax, ymin, ymax, status: number) (WC) [in Lua]`

Returns the rectangle and the clipping status. It is not necessary to provide all return pointers, you can provide only the desired values and *`NULL`* for the others.

Polygons

A polygon for clipping can be created using **`cdBegin(`**`CD_CLIP`**`)/cdVertex(x,y)/.../cdEnd()`**.

See the documentation of cdBegin/cdVertex/cdEnd.

`int* cdGetClipPoly(int *n); [in C]`
`double * wdGetClipPoly(int *n); (WC) [in C]`
`cd.GetClipPoly() -> (n: number, points: table) [in Lua]`
`cd.wGetClipPoly() -> (n: number, points: table) (WC) [in Lua]`

Returns the number of points in the clipping polygon and the polygon itself as a sequence of points, each with its respective x and y coordinates (E.g.: x1,y1,x2,y2,x3,y3,...).

Complex Clipping Regions

A complex region can composed of boxes, sectors, chords, polygons and texts.

It is implemented only in the Windows GDI, GDI+ and X-Windows base drivers.

Creating

Complex clipping regions can be created using **cdBegin(CD_REGION)/(filled primtives)/.../cdEnd()**. For more about cdBegin and cdEnd see Polygons.

Between a **cdBegin(CD_REGION)** and a **cdEnd()**, all calls to **cdBox**, **cdSector**, **cdChord**, **cdBegin(CD_FILL)/cdVertex(x,y)/.../cdEnd()** and **cdText** will be composed in a region for clipping. This is the only exception when you can call a **cdBegin** after another **cdBegin**.

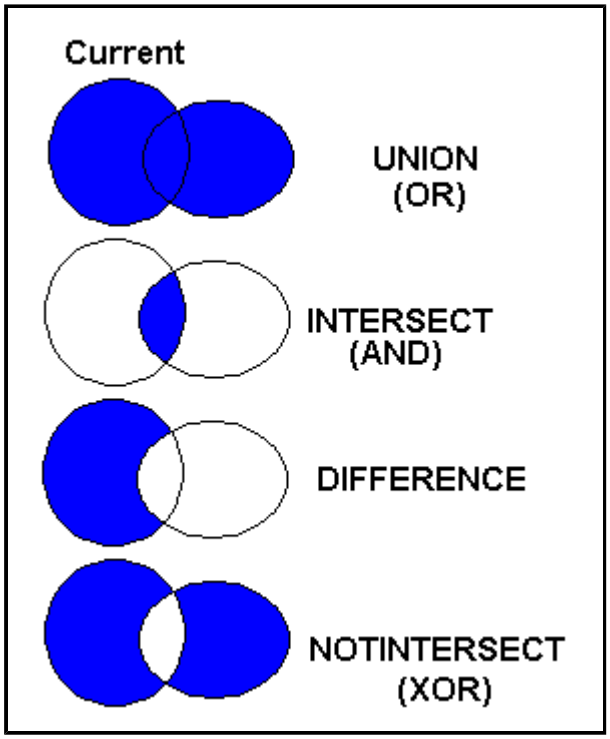
When you call **cdBegin(CD_REGION)** a new empty region will be created. So for the first operation you should use **CD_UNION** or **CD_NOTINTERSECT** combine modes. When you finished to compose the region call **cdEnd()**.

To make the region active you must call **cdClip(CD_CLIPREGION)**. For other clipping regions see Clipping.

```
int cdRegionCombineMode(int mode); [in C]
cd.RegionCombineMode(mode: number) -> (old_mode: number) [in Lua]
```

Changes the way regions are combined when created. Returns the previous status. Values: **CD_UNION, CD_INTERSECT, CD_DIFFERENCE or CD_NOTINTERSECT**. The value **CD_QUERY** simply returns the current status. Default value: **CD_UNION**.

Combine Modes




```
int cdPointInRegion(int x, int y); [in C]
cd.PointInRegion(x, y: number) -> (status: number) [in Lua]
```

Returns a non zero value if the point is contained inside the current region.

```
void cdOffsetRegion(int dx, int dy); [in C]
```

```
void wdOffsetRegion(double dx, double dy); (WC) [in C]
cd.OffsetRegion(dx, dy: number) [in Lua]
cd.wOffsetRegion(dx, dy: number) (WC) [in Lua]
```

Moves the current region by the given offset. In X-Windows, if the region moves to outside the canvas border, the part moved outside will be lost, the region will need to be reconstruted.

```
void cdRegionBox(int *xmin, int *xmax, int *ymin, int *ymax); [in C]
void wdRegionBox(double *xmin, double *xmax, double *ymin, double *ymax); (WC) [in C]
cd.RegionBox() -> (xmin, xmax, ymin, ymax, status: number) [in Lua]
cd.wRegionBox() -> (xmin, xmax, ymin, ymax, status: number) (WC) [in Lua]
```

Returns the rectangle of the bounding box of the current region. It is not necessary to provide all return pointers, you can provide only the desired values and *NULL* for the others.

Marks

A mark is a punctual representation. It can have different sizes and types. All types are affected only by mark attributes and by the foreground color.

```
void cdPixel(int x, int y, long int color); [in C]
void wdPixel(double x, double y, long int color); (WC) [in C]
cd.Pixel(x, y: number, color: color_tag) [in Lua]
cd.wPixel(x, y: number, color: color_tag) (WC) [in Lua]
```

Configures the pixel **(x,y)** with the color defined by **color**. It is the smallest element of the canvas. It depends only on global attributes of the canvas.

```
void cdMark(int x, int y); [in C]
void wdMark(double x, double y); (WC) [in C]
cd.Mark(x, y: number) [in Lua]
cd.wMark(x, y: number) (WC) [in Lua]
```

Draws a mark in **(x,y)** using the current foreground color. It is not possible to use this function between a call to functions **cdBegin** and **cdEnd** if the type of mark is set to **CD_DIAMOND**. If the active driver does not include this primitive, it will be simulated using other primitives from the library, such as **cdLine**.

If you will call function **cdMark** or **wdMark** several times in a sequence, then it is recommended that the application changes the filling and line attributes to those used by the **cdMark** function:

```
cdInteriorStyle(CD_SOLID);
cdLineStyle(CD_CONTINUOUS);
cdLineWidth(1);
```

This will greatly increase this function's performance. Also in this case, if the mark is very small, we suggest using the `cdPixel` function so that the application itself draws the mark. In many cases, this also increases this function's performance.

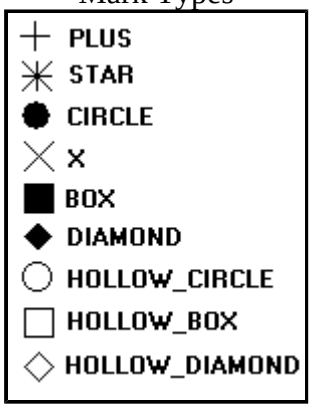
Attributes

```
int cdMarkType(int type); [in C]
cd.MarkType(type: number) -> (old_type: number) [in Lua]
```

Configures the current mark type for: **CD_PLUS**, **CD_STAR**, **CD_CIRCLE**, **CD_X**, **CD_BOX**, **CD_DIAMOND**, **CD_HOLLOW_CIRCLE**, **CD_HOLLOW_BOX** or **CD_HOLLOW_DIAMOND**. Returns the previous value. Default

value: **CD_STAR**. Value **CD_QUERY** simply returns the current value.

Mark Types




```
int cdMarkSize(int size); [in C]
double cdMarkSize(double size); (WC) [in C]
cd.MarkSize(size: number) -> (old_size: number) [in Lua]
cd.wMarkSize(size: number) -> (old_size: number) (WC) [in Lua]
```

Configures the mark size in pixels. Returns the previous value. Default value: 10. Value **CD_QUERY** simply returns the current value. Valid width interval: >= 1.

In WC, it configures the current line width in millimeters.

Lines

Line are segments that connects 2 or more points. The `cdLine` function includes the 2 given points and draws the line using the foreground color. Line thickness is controlled by the `cdLineWidth` function. By using function `cdLineStyle` you can draw dashed lines with some variations. Lines with a style other than continuous are affected by the back opacity attribute and by the background color.

```
void cdLine(int x1, int y1, int x2, int y2); [in C]
void wdLine(double x1, double y1, double x2, double y2); (WC) [in C]
cd.Line(x1, y1, x2, y2: number) [in Lua]
cd.wLine(x1, y1, x2, y2: number) (WC) [in Lua]
```

Draws a line from **(x1,y1)** to **(x2,y2)** using the current foreground color and line width and style. Both points are included in the line.

Polygons and Bezier Lines

Open polygons can be created using **cdBegin(CD_OPEN_LINES)/cdVertex(x,y)/.../cdEnd()**.

Closed polygons use the same number of vertices but the last point is automatically connected to the first point. Closed polygons can be created using **cdBegin(CD_CLOSED_LINES)/cdVertex (x,y)/.../cdEnd()**.

Bezier lines can be created using **cdBegin(CD_BEZIER)/cdVertex(x,y)/.../cdEnd()**. At least 4 vertices must be defined. The two vertices of the middle are the control vertices. A sequence of bezier lines can be defined using more 3 vertices, two control points and an end point, the last point of the previous bezier will be used as the start point.

See the documentation of cdBegin/cdVertex/cdEnd.

```
void cdRect(int xmin, int xmax, int ymin, int ymax); [in C]
```

```
void wdRect(double xmin, double xmax, double ymin, double ymax); (WC) [in C]
cd.Rect(xmin, xmax, ymin, ymax: number) [in Lua]
cd.wRect(xmin, xmax, ymin, ymax: number) (WC) [in Lua]
```

Draws a rectangle with no filling. All points in the limits of interval **x_min<=x<=x_max, y_min<=y<=y_max** will be painted. It is affected by line attributes and the foreground color. If the active driver does not include this primitive, it will be simulated using the **cdLine** primitive.

```
void cdArc(int xc, int yc, int w, int h, double angle1, double angle2); [in C]
void wdArc(double xc, double yc, double w, double h, double angle1, double
angle2); (WC) [in C]
cd.Arc(xc, yc, w, h, angle1, angle2: number) [in Lua]
cd.wArc(xc, yc, w, h, angle1, angle2: number) (WC) [in Lua]
```

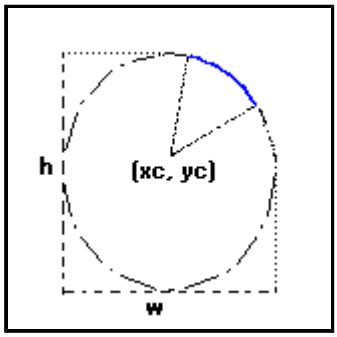
Draws the arc of an ellipse aligned with the axis, using the current foreground color and line width and style. It is drawn counter-clockwise. The coordinate **(xc,yc)** defines the center of the ellipse. Dimensions **w** and **h** define the elliptic axes X and Y, respectively.

Angles **angle1** and **angle2**, in degrees define the arc's beginning and end, but they are not the angle relative to the center, except when w==h and the ellipse is reduced to a circle. The arc starts at the point **(xc+(w/2)*cos(angle1),yc+(h/2)*sin(angle1))** and ends at **(xc+(w/2)*cos(angle2),yc+(h/2)*sin(angle2))**. A complete ellipse can be drawn using 0 and 360 as the angles.

The angles are specified so if the size of the ellipse (w x h) is changed, its shape is preserved. So the angles relative to the center are dependent from the ellipse size. The actual angle can be obtained using **rangle = atan2((h/2)*sin(angle),(w/2)*cos(angle))**.

The angles are given in degrees. To specify the angle in radians, you can use the definition **CD_RAD2DEG** to multiply the value in radians before passing the angle to CD.

Arc Parameters

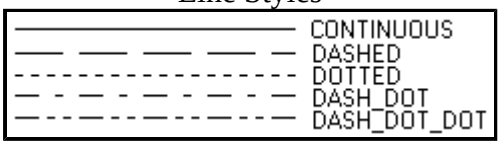


Attributes

```
int cdLineStyle(int style); [in C]
cd.LineStyle(style: number) -> (old_style: number) [in Lua]
```

Configures the current line style for: **CD_CONTINUOUS**, **CD_DASHED**, **CD_DOTTED**, **CD_DASH_DOT, CD_DASH_DOT_DOT,** or **CD_CUSTOM**. Returns the previous value. Default value: **CD_CONTINUOUS**. Value **CD_QUERY** simply returns the current value. When **CD_CUSTOM** is used the **cdLineStyleDahes** function must be called before to initialize the custom dashes.

Line Styles



void cdLineStyleDashes(int* dashes, int count); [in C]
cd.LineStyleDashes(dashes: *table*, count: *number*) -> (old_style: *number*) [in Lua]

Defines the custom line style dashes. The first value is the lenght of the first dash, the second value is the leght of the first space, and so on. For example: "10 2 5 2" means dash size 10, space size 2, dash size 5, space size 2, and repeats the pattern.

int cdLineWidth(int width); [in C]
double cdLineWidth(double width); (WC) [in C]
cd.LineWidth(width: *number*) -> (old_width: *number*) [in Lua]
cd.wLineWidth(width: *number*) -> (old_width: *number*) (WC) [in Lua]

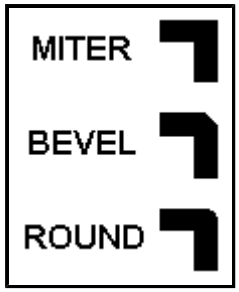
Configures the width of the current line (in pixels). Returns the previous value. Default value: 1. Value **CD_QUERY** simply returns the current value. Valid width interval: >= 1.

In WC, it configures the current line width in millimeters.

int cdLineJoin(int style); [in C]
cd.LineJoin(style: *number*) -> (old_style: *number*) [in Lua]

Configures the current line style for: **CD_MITER**, **CD_BEVEL** or **CD_ROUND**. Returns the previous value. Default value: **CD_MITER**. Value **CD_QUERY** simply returns the current value.

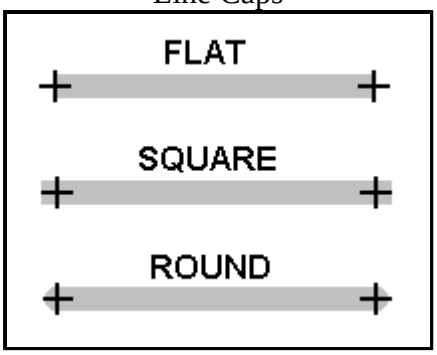
Line Joins



int cdLineCap(int style); [in C]
cd.LineCap(style: *number*) -> (old_style: *number*) [in Lua]

Configures the current line style for: **CD_CAPFLAT**, **CD_CAPSQUARE** or **CD_CAPROUND**. Returns the previous value. Default value: **CD_CAPFLAT**. Value **CD_QUERY** simply returns the current value.

Line Caps



Filled Areas

It is an area filled with the foreground color, but it depends on the current interior style. The SOLID style depends only on the foreground color. The HATCH style depends on the foreground color and on the back opacity attribute. The hatch lines drawn with this style do not depend on the line attributes. The STIPPLE style depends on the foreground color, the background color and the back opacity attribute. The PATTERN style depends only on global canvas attributes.

The fillings can have only one color, using the foreground color, they can be hatched with several styles, and can be made with a color or monochromatic pattern using either the foreground or the background color. The hatched and monochromatic fillings are affected by the back opacity - if it is transparent, then the background is not drawn with the background color. If either the background or the foreground color are modified, the hatched and monochromatic fillings must be modified again in order to be updated.

Note that when a Filling Attribute is modified, the active filling style is now that of the modified attribute (hatch, stipple or pattern). Notice that this is not true for the clipping area. When the clipping area is modified, the clipping is only affected if it is active.

Filled Polygons

Filled polygons can be created using **cdBegin(**CD_FILL**)/cdVertex(x,y)/.../cdEnd()**.

See the documentation of cdBegin/cdVertex/cdEnd.

void cdBox(int xmin, int xmax, int ymin, int ymax); [in C]
void wdBox(double xmin, double xmax, double ymin, double ymax); (WC) [in C]
cd.Box(xmin, xmax, ymin, ymax: *number*) [in Lua]
cd.wBox(xmin, xmax, ymin, ymax: *number*) (WC) [in Lua]

Fills a rectangle according to the current interior style. All points in the interval **x_min<=x<=x_max, y_min<=y<=y_max** will be painted. When the interior style CD_HOLLOW is defined, the function behaves like its equivalent **cdRect.**

void cdSector(int xc, int yc, int w, int h, double angle1, double angle2); [in C]
void wdSector(double xc, double yc, double w, double h, double angle1, double angle2); (WC) [in C]
cd.Sector(xc, yc, w, h, angle1, angle2: *number*) [in Lua]
cd.wSector(xc, yc, w, h, angle1, angle2: *number*) (WC) [in Lua]

Fills the arc of an ellipse aligned with the axis, according to the current interior style, in the shape of a pie. It is drawn counter-clockwise. The coordinate **(xc,yc)** defines the center of the ellipse. Dimensions **w** and **h** define the elliptic axes X and Y, respectively.

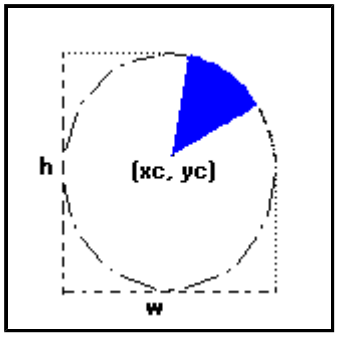
Angles **angle1** and **angle2**, in degrees, define the arc's beginning and end, but they are not the angle relative to the center, except when w==h and the ellipse is reduced to a circle. The arc starts at the point **(xc+(w/2)*cos(angle1),yc+(h/2)*sin(angle1))** and ends at **(xc+(w/2)*cos(angle2),yc+(h/2)*sin(angle2))**. A complete ellipse can be drawn using 0 and 360 as the angles.

The angles are specified so if the size of the ellipse (w x h) is changed, its shape is preserved. So the angles relative to the center are dependent from the ellipse size. The actual angle can be obtained using **rangle = atan2((h/2)*sin(angle),(w/2)*cos(angle))**.

The angles are given in degrees. To specify the angle in radians, you can use the definition **CD_RAD2DEG** to multiply the value in radians before passing the angle to CD.

When the interior style CD_HOLLOW is defined, the function behaves like its equivalent **cdArc**, plus two lines connecting to the center.

Sector Parameters

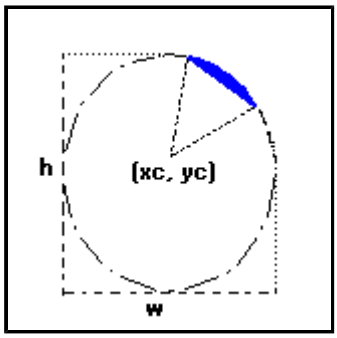


void cdChord(int xc, int yc, int w, int h, double angle1, double angle2); [in C]
void wdChord(double xc, double yc, double w, double h, double angle1, double angle2); (WC) [in C]
cd.Chord(xc, yc, w, h, angle1, angle2: *number*) [in Lua]
cd.wChord(xc, yc, w, h, angle1, angle2: *number*) (WC) [in Lua]

Fills the arc of an ellipse aligned with the axis, according to the current interior style, the start and end points of the arc are connected. The parameters are the same as the **cdSector**.

When the interior style CD_HOLLOW is defined, the function behaves like its equivalent **cdArc**, plus a line connecting the arc start and end points.

Chord Parameters

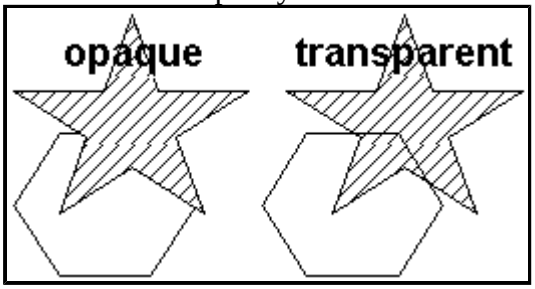


Attributes

int cdBackOpacity(int opacity); [in C]
cd.BackOpacity(opacity: *number*) -> (old_opacity: *number*) [in Lua]

Configures the background opacity to filling primitives based on the foreground and background colors. Values: **CD_TRANSPARENT** or **CD_OPAQUE**. If it is opaque, the text primitive, for instance, will erase whatever is in the bounding box with the background color. If it is transparent, only the foreground color is painted. The same occurs for the interior styles of *hatch* and *stipple*, and for lines with a style different from CD_CONTINUOUS. It returns the previous value. Default value: **CD_TRANSPARENT**. Value **CD_QUERY** simply returns the current value.

Back Opacity Attribute



int cdFillMode(int mode); [in C]
cd.FillMode(mode: *number*) -> (old_mode: *number*) [in Lua]

Selects a predefined polygon fill rule (**CD_EVENODD** or **CD_WINDING**). Returns the previous value. Default value: **CD_EVENODD**. Value **CD_QUERY** simply returns the current value.

Fill Modes

int cdInteriorStyle(int style); [in C]
cd.InteriorStyle(style: *number*) -> (old_style: *number*) [in Lua]

Configures the current style for the area filling primitives: **CD_SOLID**, **CD_HOLLOW**, **CD_HATCH**, **CD_STIPPLE** or **CD_PATTERN**. Note that **CD_HATCH** and **CD_STIPPLE** are affected by the opacity of the current background. It returns the previous value. Default value: **CD_SOLID**. Value **CD_QUERY** simply returns the current value.

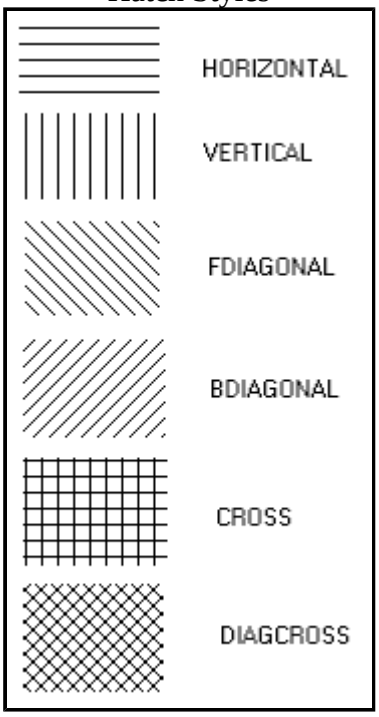
If *a stipple* or *a pattern* were not defined, the state of the attribute is not changed.

When the style **CD_HOLLOW** is defined, functions **cdBox** and **cdSector** behave as their equivalent **cdRect** and **cdArc+Lines**, and the polygons with style **CD_FILL** behave like **CD_CLOSED_LINES**.

int cdHatch(int style); [in C]
cd.Hatch(style: *number*) -> (old_style: *number*) [in Lua]

Selects a predefined *hatch* style (**CD_HORIZONTAL**, **CD_VERTICAL**, **CD_FDIAGONAL**, **CD_BDIAGONAL**, **CD_CROSS** or **CD_DIAGCROSS**) and sets the interior style to **CD_HATCH**. Returns the previous value. Default value: **CD_HORIZONTAL**. Value **CD_QUERY** simply returns the current value.

Hatch Styles



void cdStipple(int w, int h, unsigned char *fgbg) [in C]
cd.Stipple(stipple: *stipple_tag*) [in Lua]

Defines a **wxh** matrix of zeros and ones. The zeros are mapped to the background color or are transparent, according to the background opacity attribute. The ones are mapped to the foreground color. The function sets the interior style to **CD_STIPPLE**. To avoid having to deal with matrices in C, the element **(i,j)** of **fgbg** is stored as **fgbg[j*w+i]**. The origin is the left bottom corner of the image. It cannot be queried. It does not need to be stored by the application, as it is internally replicated by the library.

void wdStipple(int w, int h, unsigned char *fgbg, double w_mm, double h_mm); [in C]
cd.wStipple(stipple: *stipple_tag*, w_mm, h_mm: *number*) [in Lua]

Allows specifying the stipple in world coordinates. Another stipple will be created with the size in pixels corresponding to the specified size in millimeters. The use of this function may produce very large or very small stipples.

unsigned char * cdGetStipple(int* w, int* h); [in C]
cd.GetStipple() - > (stipple: *stipple_tag*) [in Lua]

Returns the current *stipple* and its dimensions. Returns NULL if no *stipple* was defined.

void cdPattern(int w, int h, long int *color); [in C]
cd.Pattern(pattern: *pattern_tag*) [in Lua]

Defines a new **wxh** color matrix and sets the interior style to **CD_PATTERN**. To avoid having to deal with matrices in C, the color element **(i,j)** is stored as **color[j*w+i]**. The origin is the left bottom corner of the image. It cannot be queried. It does not need to be stored by the application, as it is internally replicated by the library.

void wdPattern(int w, int h, long int *color, double w_mm, double h_mm); [in C]
cd.wPattern(pattern: *pattern_tag*, w_mm, h_mm: *number*) [in Lua]

Allows specifying the pattern in world coordinates. Another pattern will be created with the size in pixels corresponding to the specified size in millimeters. The use of this function may produce very large or very small patterns.

long int * cdGetPattern(int* w, int* h); [in C]
cd.GetPattern() - > (pattern: *pattern_tag*) [in Lua]

Returns the current *pattern* and its dimensions. Returns NULL if no *pattern* was defined.

Extras in Lua

cd.CreatePattern(width, height: number) -> (pattern: pattern_tag)

Creates a pattern in Lua.

cd.KillPattern(pattern: pattern_tag)

Destroys the created pattern and liberates allocated memory.

cd.CreateStipple(width, height: number) -> (stipple: stipple_tag)

Creates a stipple in Lua.

cd.KillStipple(stipple: stipple_tag)

Destroys the created stipple and liberates allocated memory.

Data Access

Data access in Lua is done directly using the operator "`[y*width + x]`".

All new types can have their values checked or changed directly as if they were Lua tables:

```
pattern[y*16 + x] = cd.EncodeColor(r, g, b)
...
color = pattern[y*16 + x]
r, g, b = cd.DecodeColor(color)
...
cd.Pattern(pattern)
```

Notice that the type of value returned or received by `pattern[i]` is a `color_tag`, the same type used with functions `cdEncodeColor`, `cdDecodeColor`, `cdPixel`, `cdForeground` and `cdBackground`. The value returned or received by `stipple[i]` is a number.

Text

A raster text using a font with styles. The position the text is drawn depends on the text alignment attribute.

The library has only 4 standard typefaces: System (which depends on the driver and platform), Courier (mono spaced with serif), Times Roman (proportional with serif) and Helvetica (proportional without serif). Each typeface can have some styles: Plain, **Bold**, *Italic* and a combination of ***Bold and Italic***. As an alternative to the standard typefaces, you can use native driver typefaces with the function `cdNativeFont`.

You may retrieve the dimensions of the selected font with function `cdFontDim`. Also you may retrieve the bounding box of a specific text before drawing by using the `cdTextSize` and `cdTextBox` functions.

The text is drawn using a reference point; you can change the alignment relative to this point using the `cdTextAligment` function.

---

void cdText(int x, int y, char *text); [in C]
void wdText(double x, double y, char *s); (WC) [in C]
cd.Text(x, y: *number*, text: *string*) [in Lua]
cd.wText(x, y: *number*, text: *string*) (WC) [in Lua]

Inserts a text in **(x,y)** according to the current font and text alignment. It expects an ANSI string with no line breaks. A string with codes over 128 may display wrong characters.

Attributes

void cdFont(int typeface, int style, int size); [in C]
void wdFont(int typeface, int style, double size); (WD) [in C]
cd.Font(typeface, style, size: *number*) [in Lua]
cd.wFont(typeface, style, size: *number*) (WD) [in Lua]

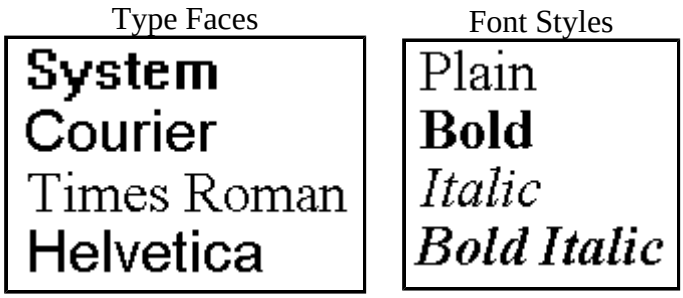
Selects a text font. The font type can be: **CD_SYSTEM**, **CD_COURIER**, **CD_TIMES_ROMAN, CD_HELVETICA, or CD_NATIVE**. The style can be: **CD_PLAIN**, **CD_BOLD**, **CD_ITALIC** or **CD_BOLD_ITALIC**. The size is provided in points (1/72 inch), and, to make the selection easier, CD provides three constants: **CD_SMALL**=8, **CD_STANDARD**=12 and **CD_LARGE**=18. Points were used here because it is a common unit to define font sizes.

Default values: **CD_SYSTEM**, **CD_PLAIN**, **CD_STANDARD**.

If you wish to specify a value in pixels, simply pass the size value in pixels providing a negative value. This way, the application will know that such value is in pixels instead of points. If you wish to specify the size in pixels but want the function to keep the value in points, use function **cdPixel2MM** to convert pixels into millimeters, then use the formula "**(value in *points*) = 2.84 * (value in millimeters)**" to convert from millimeters into points. Instead of 2.84, you can use the definition **CD_MM2PT**.

The CD_NATIVE typeface is valid only if there was a previous call to cdNativeFont. The style and size parameters are ignored.

In WC, the size is specified in millimeters, but is internally passed in points.




```
void cdGetFont(int *typeface, int *style, int *size); [in C]
void wdGetFont(int *typeface, int *style, double *size); (WC) [in C]
cd.GetFont() -> (typeface, style, size: number) [in Lua]
cd.wGetFont() -> (typeface, style, size: number) (WC) [in Lua]
```

Returns the values of the font modified by function **cdFont**, ignoring the values modified by function **cdNativeFont**. It is not necessary to provide all return pointers; you can provide only the desired values. If the current typeface is CD_NATIVE, then style and size may not reflect the actual style and size.

In WC, the size is returned in millimeters.

```
char* cdNativeFont(char* font); [in C]
cd.NativeFont(font: string) -> (old_font: string) [in Lua]
```

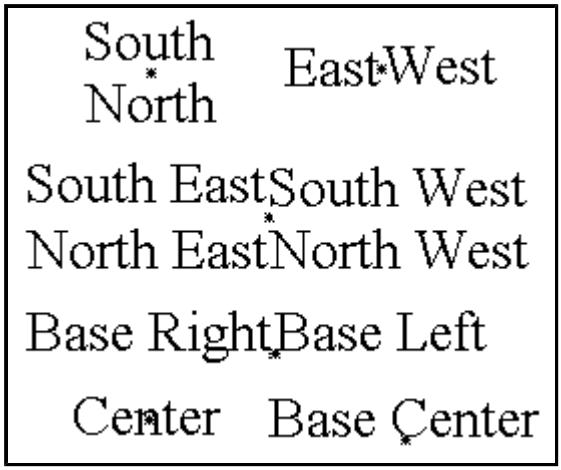
Selects a native text font. The font description depends on the driver and the platform. It does not need to be stored by the application, as it is internally replicated by the library. It returns the previous string. Calling this function will set the current typeface to CD_NATIVE.

Passing NULL as a parameter, it returns only the previous string and does not change the font. The value returned is the last attributed value, which may not correspond exactly to the font selected by the driver.

```
int cdTextAlignment(int alignment); [in C]
cd.TextAlignment(alignment: number) -> (old_alignment: number) [in Lua]
```

Defines the vertical and horizontal alignment of a text as: **CD_NORTH**, **CD_SOUTH**, **CD_EAST**, **CD_WEST**, **CD_NORTH_EAST**, **CD_NORTH_WEST**, **CD_SOUTH_EAST**, **CD_SOUTH_WEST**, **CD_CENTER**, **CD_BASE_LEFT**, **CD_BASE_CENTER**, or **CD_BASE_RIGHT**. Returns the previous value. Default value: **CD_BASE_LEFT**. Value **CD_QUERY** simply returns the current value.

Text Alignment



double cdTextOrientation(double angle); [in C]
cd.TextOrientation(angle: *number*) -> (old_angle: *number*) [in Lua]

Defines the text orientation, which is an angle provided in degrees relative to the horizontal line according to which the text is drawn. Returns the previous value. Value **CD_QUERY** simply returns the current value. The default value is 0.

Properties

void cdFontDim(int *max_width, int *line_height, int *ascent, int *descent); [in C]
void wdFontDim(double *max_width, double *height, double *ascent, double *descent); (WC) [in C]
cd.FontDim() -> (max_width, max_height, ascent, descent: *number*) [in Lua]
cd.wFontDim() -> (max_width, max_height, ascent, descent: *number*) (WC) [in Lua]

Returns the maximum width of a character, the line's height and the *ascent* and *descent* of the characters of the currently selected font. The line's height is the sum of the *ascents* and *descents* of a given additional space (if this is the case). All values are given in pixels. If the driver does not support this kind of query, the values will be given 0 (zero). It is not necessary to provide all return pointers, you can provide only the desired values and *NULL* for the others.

Font Dimension Attributes

void cdTextSize(char *text, int *width, int *height); [in C]
void wdTextSize(char *s, double *width, double *height); (WC) [in C]
cd.TextSize(text: *string*) -> (width, heigth: *number*) [in Lua]
cd.wTextSize(text: *string*) -> (width, heigth: *number*) (WC) [in Lua]

Returns the width and height of a text's minimum box with the currently selected font. If the driver does not support this kind of query, the values will be given 0 (zero). It is not necessary to provide all return pointers, you can provide only the desired values and *NULL* for the others.

void cdTextBox(int x, int y, char *text, int *xmin, int *xmax, int *ymin, int

***ymax); [in C]**
void wdTextBox(double x, double y, char *s, double *xmin, double *xmax, double *ymin, double *ymax); (WC) [in C]
cd.TextBox(x, y: *number*, text: *string*) -> (xmin, xmax, ymin, ymax: *number*) [in Lua]
cd.wTextBox(x, y: *number*, text: *string*) -> (xmin, xmax, ymin, ymax: *number*) (WC) [in Lua]

Returns the horizontal bounding rectangle of a text box, even if the text has an orientation. It is not necessary to provide all return pointers, you can provide only the desired values and *NULL* for the others.

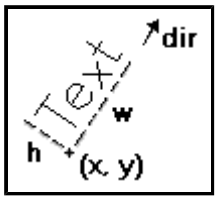
void cdTextBounds(int x, int y, char *text, int *rect); [in C]
void wdTextBounds(double x, double y, char *s, double *rect); (WC) [in C]
cd.TextBounds(x, y: *number*, text: *string*) -> (rect0, rect1, rect2, rect3, rect4, rect5, rect6, rect7: *number*) [in Lua]
cd.wTextBounds(x, y: *number*, text: *string*) -> (rect0, rect1, rect2, rect3, rect4, rect5, rect6, rect7: *number*) (WC) [in Lua]

Returns the oriented bounding rectangle of a text box. The rectangle corners are returned in counter-clock wise order starting with the bottom left corner, (x,y) arranged (x0,y0,x1,y1,x2,y2,x3,y3,x4,y4,x5,y5,x6,y6,x7,y7).

Vector Text

It is a text that uses a font created only with line segments. It is very useful to be scaled. You must set the text size before drawing any text. The functions ignore the new line character "\n"; only the `wdMultiLineVectorText` function will consider this character. The default direction is horizontal from left to right.

Vector Text Parameters



---

void cdVectorText(int x, int y, const char *text); [in C]
void wdVectorText(double x, double y, char * s); (WC) [in C]
cd.VectorText(x, y: *number*, text: *string*) [in Lua]
cd.wVectorText(x, y: *number*, text: *string*) (WC) [in Lua]

Draws a vector text in position **(x,y)**, respecting the alignment defined by **cdTextAlignment**. It ignores the configuration **cdBackOpacity**, being always transparent. It also ignores strings with multiple lines. It is ESSENTIAL to call **cdVectorTextSize** or **cdVectorCharSize** before using **cdVectorText** or **cdMultiLineVetorText**.

void cdMultiLineVectorText(int x, int y, const char *text); [in C]
void wdMultiLineVectorText(double x, double y, char * s); (WC) [in C]
cd.MultiLineVectorText(x, y: *number*, text: *string*) [in Lua]
cd.wMultiLineVectorText(x, y: *number*, text: *string*) (WC) [in Lua]

Draws a vector text with several lines in position **(x,y)**, respecting the alignment defined by **cdTextAlignment**. It ignores the configuration **cdBackOpacity**, being always transparent. Lines are broken by characters `"\n"`. Each line respects the scale defined in **cdVectorTextSize** or **cdVectorCharSize**. This function's purpose is to make function **cdVectorText** more efficient, not being concerned with multiple lines.

Attributes

void cdVectorTextDirection(int x1, int y1, int x2, int y2); [in C]
void wdVectorTextDirection(double x1, double y1, double x2, double y2); (WC) [in C]
cd.VectorTextDirection(x1, y1, x2, y2: *number*) [in Lua]
cd.wVectorTextDirection(x1, y1, x2, y2: *number*) (WC) [in Lua]

Defines the text direction by means of two points, **(x1,y1)** and **(x2,y2)**. The default direction is horizontal from left to right.

double* cdVectorTextTransform(double* matrix); [in C]
cd.VectorTextTransform(matrix: *table*) -> (old_matrix: table) [in Lua]

Defines a transformation matrix with 6 elements. If the matrix is NULL, no transformation is set. The default direction is no transformation. The origin is the left bottom corner of matrix. It returns the previous matrix, and the returned vector is only valid until the following call to the function.

The matrix contains rotation and translation elements. It is applied after computing the position and orientation normal to the vector text. We can describe the elements as follows:

```
|x'|   | cos(ang)  -sin(ang)  trans_x |   |x|                | 3   4   5|
|y'| = | sin(ang)   cos(ang)  trans_y | * |y|  with indices   | 0   1   2|
```

void cdVectorTextSize(int w, int h, char *text); [in C]
void wdVectorTextSize(double size_x, double size_y, char *s); (WC) [in C]
cd.VectorTextSize(w, h: *number*, text: *string*) [in Lua]
cd.wVectorTextSize(w, h: *number*, text: *string*) (WC) [in Lua]

Modifies the scale of the vector text so that it corresponds to the string of the bounding box defined by **w** and **h**. It ignores strings with multiple lines.

double cdVectorCharSize(int size); [in C]
double wdVectorCharSize(double size); (WC) [in C]
cd.VectorCharSize(size: *number*) -> (old_size: *number*) [in Lua]
cd.wVectorCharSize(size: *number*) -> (old_size: *number*) (WC) [in Lua]

Sets the height of the characters and adjusts the width according to it. Returns the previous value. **CD_QUERY** returns the current value.

char* cdVectorFont(char *filename); [in C]
cd.VectorFont(filename: *string*) -> (font_name: *string*) [in Lua]

Replaces the current vector font with a font stored in a file with a given name. Returns the name of the font loaded or NULL, if it fails. If `filename` is NULL, it activates the default font "**Simplex II**". There is no file associated to this font, it is an embedded font. The library will attempt to load a font from the directory defined by the environment variable "**CDDIR**", apart from the `filename` parameter. If it fails, it will attempt to load it using only the `filename` parameter. The file format is compatible with the GKS file format (text mode).

Properties

void cdGetVectorTextSize(char *text, int *w, int *h); [in C]
void wdGetVectorTextSize(char * s, double *x, double *y); (WC) [in C]
cd.GetVectorTextSize(text: *string*) -> (w, h: *number*) [in Lua]
cd.wGetVectorTextSize(text: *string*) -> (w, h: *number*) (WC) [in Lua]

Queries the string's bounding box. Ignores strings with multiple lines. It is not necessary to provide all return pointers, you can provide only the desired values and NULL for the others.

void cdGetVectorTextBounds(char *s, int px, int py, int *rect); [in C]
void wdGetVectorTextBounds(char *s, double x, double y, double *rect); (WC) [in C]
cd.GetVectorTextBounds(s: *string*, px,py: number) -> (rect: table) [in Lua]
cd.wGetVectorTextBounds(s: string, px,py: number) -> (rect: table) (WC) [in Lua]

Returns the bounding rectangle of the text specified in the current vector font, alignment and direction. Eight values are returned, corresponding to pairs (x,y) of the rectangle's vertices ordered conter-clockwise, starting by the bottom left corner.

Client Images

There are 2 kinds of client images: RGB and Indexed RGB (or MAP). The RGB image is composed by 3 buffers: red, green and blue (more colors, more memory). The MAP image is composed by 1 buffer of indices for a table and one table of encoded RGB values (less colors, less memory).

The image buffer is described by its width and height in pixels. The starting point of the buffer is the origin of the image, which is located at its bottom left corner. To retrieve a pixel in the image, use the formula `pixel(x,y)=buffer[y*width + x]`.

The Put functions may do zoom in or out; zero order interpolation is used to scale the image. It is not possible to specify a part of the image to be drawn.

---

void cdGetImageRGB(unsigned char *r, unsigned char *g, unsigned char *b, int x, int y, int w, int h); [in C]
cd.GetImageRGB(imagergb: *imagergb_tag*; x, y: *number*) [in Lua]

Returns the red, green and blue components of each pixel in a server image. The RGB components are provided in three matrices stored as byte arrays. The **(i,j)** component of these matrices is at the address **(j*w+i)**. As occurs with all primitives from the Canvas Draw library, the pixel **(0,0)** is at the bottom left corner, and the pixel **(w-1,h-1)** is that the upper right corner of the image rectangle.

void cdPutImageRectRGB(int iw, int ih, unsigned char *r, unsigned char *g, unsigned char *b, int x, int y, int w, int h, int xmin, int xmax, int ymin, int ymax); [in C]
void wdPutImageRectRGB(int iw, int ih, unsigned char *r, unsigned char *g, unsigned char *b, double x, double y, double w, double h, int xmin, int xmax, int ymin, int ymax); (WC) [in C]
cd.PutImageRectRGB(imagergb: *imagergb_tag*; x, y, w, h, xmin, xmax, ymin, ymax: *number*) [in Lua]
cd.wPutImageRectRGB(imagergb: *imagergb_tag*; x, y, w, h, xmin, xmax, ymin, ymax: *number*) (WC) [in Lua]

Puts, in a specified area of the canvas, an image with its red, green and blue components defined in the three matrices stored in byte arrays. The **(i,j)** component of these matrices is at the address **(j*iw+i)**. The pixel **(0,0)** is at the bottom left corner, and the pixel **(iw-1,ih-1)** is that the upper right corner of the image rectangle.

Parameters **w** and **h** refer to the target rectangle of the canvas, so that it is possible to reduce or expand the image drawn. If **w** and **h** are 0, the size of the image is assumed (**iw** and **ih**).

It also allows specifying a rectangle inside the image to be drawn, if **xmin**, **xmax**, **ymin** and **ymax** are 0 then the whole image is assumed.

If the driver has bpp <=8 or only 256 colors or less, then the image is converted to 256 optimal colors using the function **cdRGB2Map** and is drawn using **cdPutImageRectMap**.

void cdPutImageRectRGBA(int iw, int ih, unsigned char *r, unsigned char *g, unsigned char *b, unsigned char *a, int x, int y, int w, int h, int xmin, int xmax, int ymin, int ymax); [in C]
void wdPutImageRectRGBA(int iw, int ih, unsigned char *r, unsigned char *g, unsigned char *b, unsigned char *a, double x, double y, double w, double h, int xmin, int xmax, int ymin, int ymax); (WC) [in C]
cd.PutImageRectRGBA(imagergba: *imagergba_tag*; x, y, w, h, xmin, xmax, ymin, ymax: *number*) [in Lua]
cd.wPutImageRectRGBA(imagergba: *imagergba_tag*; x, y, w, h, xmin, xmax, ymin, ymax: *number*) (WC) [in Lua]

The same as function **cdPutImageRectRGB**, except for the fact that it is possible to specify an alpha channel. The resulting color is the image color weighted by the alpha value, using the formula **result= (source * alpha + destiny * (255 - alpha))/255**. This means that, if alpha is 0, the resulting color is the target color (completely transparent), and, if alpha is 255, the resulting color is the original image color (completely opaque).

If this function is not defined for a given driver or if alpha is NULL, then the function **cdPutImageRGB** is used, as long as it is defined.

void cdPutImageRectMap(int iw, int ih, unsigned char *index, long int *colors, int x, int y, int w, int h, int xmin, int xmax, int ymin, int ymax); [in C]
void wdPutImageRectMap(int iw, int ih, unsigned char *index, long int *colors, double x, double y, double w, double h, int xmin, int xmax, int ymin, int ymax); (WC) [in C]
cd.PutImageRectMap(imagemap: *imagemap_tag*; palette: *palette_tag*; x, y, w, h, xmin, xmax, ymin, ymax: *number*) [in Lua]
cd.wPutImageRectMap(imagemap: *imagemap_tag*; palette: *palette_tag*; x, y, w, h, xmin, xmax, ymin, ymax: *number*) (WC) [in Lua]

The same as function **cdPutImageRectRGB**, except for the fact that the colors are provided by means of an index matrix (map). The color corresponding to a given index is given in **colors[index]**. The map is also a matrix stored as a byte vector. If the color vector is null, then a vector with 256 gray tones is assumed.

void cdRGB2Map(int width, int height, unsigned char *red, unsigned char *green, unsigned char *blue, unsigned char *map, int pal_size, long *color); [in C]
cd.RGB2Map(imagergb: *imagergb_tag,* imagemap: *imagemap_tag,* palette: *palette_tag*) [in Lua]

Converts an RGB image into an image with 256 indexed colors. The resulting image must have the same size (width x length) as the RGB image. It is necessary to allocate memory for the arrays **map** and **colors**. This is the same algorithm used in the IM library - in fact, the same code.

Extras

The following functions are used only for encapsulating the several types of client images from the library in a single structure, simplifying their treatment.

For such, a public structure was created, called **cdBitmap**, which will store the image. From this structure, the following fields are officially defined:

```
cdBitmap:
  int w      /* image width */
  int h      /* image heigth */
  int type   /* image type: CD_RGBA, CD_RGB or CD_MAP */
```

cdBitmap* cdCreateBitmap(int w, int h, int type); [in C]
cd.CreateBitmap(w, h, type: *number*) -> (image: *bitmap_tag*) [in Lua]

Creates an image with width **w**, and height **h** and of type **type**. The type can be CD_RGBA, CD_RGB or CD_MAP. However, CD_MAP only means that the image will have 256 colors if **type** is greater than 0. It is assumed that the image will be MAP with the same number of colors in the palette as **type**. Internally, the color palette is always allocated with 256 entries, which may or may not be totally fulfilled. In this case, the value of **type** can be changed as wished. It also encapsulates **cdCreateImage**.

cdBitmap* cdInitBitmap(int w, int h, int type, ...); [in C]
[There is no equivalent in Lua]

Similar to **cdCreateBitmap**, but it accepts the data area already allocated by the user. The parameters vary according to the image type.

```
CD_RGBA - (unsigned char* red, unsigned char* green, unsigned char* blue, unsigne
CD_RGB - (unsigned char* red, unsigned char* green, unsigned char* blue)
CD_MAP - (unsigned char* index, lont int* colors)
```

void cdKillBitmap(cdBitmap* image); [in C]
cd.KillBitmap(image: *bitmap_tag*) [in Lua]

Liberates the memory allocated for the image. It is not necessary to have an active canvas to call this function. It also encapsulates **cdKillImage**.

unsigned char* cdBitmapGetData(cdBitmap* image, int dataptr); [in C]
cd.BitmapGetData(image: *bitmap_tag*; dataptr: *number*) -> (data: *channel_tag*) [in Lua]

Returns a pointer to the image's data area according to **dataptr**. The following values are defined for **dataptr**:

```
CD_IRED - red component of an RGB image. channel_tag in Lua.
CD_IGREEN - green component of an RGB image. channel_tag in Lua.
CD_IBLUE - blue component of an RGB image. channel_tag in Lua.
CD_IALPHA - alpha component of an RGBA image. channel_tag in Lua.
CD_INDEX - indices of a MAP image. channel_tag in Lua.
CD_COLORS - color table of a MAP image. In this case, a type conversion must be r
```

void cdBitmapSetRect(cdBitmap* image, int xmin, int xmax, int ymin, int ymax); [in C]
cd.BitmapSetRect(image: *bitmap_tag*; xmin, xmax, ymin, ymax: *number*) [in Lua]

Allows specifying a region of interest inside the image to be used by the function **cdPutBitmap**. If no region was defined, the whole image is used, that is, (0, w-1, 0, h-1).

void cdPutBitmap(cdBitmap* image, int x, int y, int w, int h); [in C]
void wdPutBitmap(cdBitmap* image, double x, double y, double w, double h); (WC) [in C]
cd.PutBitmap(image: *bitmap_tag*; x, y, w, h: *number*) [in Lua]
cd.wPutBitmap(image: *bitmap_tag*; x, y, w, h: *number*) (WC) [in Lua]

Draws the image in the position (x,y), changing the scale. It encapsulates **cdPutImageRectRGB**, **cdPutImageRectRGBA** and **cdPutImageRectMap**. The region of the image drawn depends on the rectangle defined by **cdBitmapSetRect**. If no rectangle was defined, then the whole image is used.

The parameters **w** and **h** allow scaling the image, increasing or decreasing its dimensions when drawn. If **w** and/or **h** are 0, then no scale change is assumed.

void cdGetBitmap(cdBitmap* image, int x, int y); [in C]
cd.GetBitmap(image: *bitmap_tag*; x, y: *number*) [in Lua]

Encapsulates **cdGetImageRGB**. Nothing happens if the image is MAP.

void cdBitmapRGB2Map(cdBitmap* image_rgb, cdBitmap* image_map); [in C]
cd.BitmapRGB2Map(image_rgb: *bitmap_tag*, image_map: *bitmap_tag*) [in Lua]

Encapsulates **cdRGB2Map**. The images must be of types RGB(A) and MAP, respectively.

Extras in Lua

cd.CreateImageRGB(width, height: number) -> (imagergb: imagergb_tag)

Creates an RGB image in Lua.

cd.KillImageRGB(imagergb: imagergb_tag)

Destroys the created RGB image and liberates allocated memory.

cd.CreateImageRGBA(width, height: number) -> (imagergba: imagergba_tag)

Creates an RGBA image in Lua.

cd.KillImageRGBA(imagergba: imagergba_tag)

Destroys the created RGBA image and liberates allocated memory.

cd.CreateImageMap(width, height: number) -> (imagemap: imagemap_tag)

Creates a Map image in Lua.

cd.KillImageMap(imagemap: imagemap_tag)

Destroys the created Map image and liberates allocated memory.

Data Access

Data access in Lua is done directly using the operator "`[y*width + x]`" in image channels. Each channel works as a value table which should be consulted or modified in the following way:

```
imagergb = cd.CreateImageRGB(100, 200)
...
imagergb.r[y*100 + x] = 255
imagergb.g[y*100 + x] = 128
imagergb.b[y*100 + x] = 0
...
green = imagergb.g[y*100 + x] -- it will return 128
```

Notice also that it is always important to define the index in the channels, because, for instance, the type of `imagergb.r` (without index), `channel_tag`, is internal to the implementation of CDLua and it is useless for the end user. The order of the tables *is* important, so that `imagergb[n].r` has no meaning to CDLua and the expression will cause a fatal error. Finally, the user could expect the value of `imagergb[n]` to be of type `color_tag`. Unfortunately, this is not the case, and such expression will cause the same fatal error.

MAP images have only one channels, so the channel is not specified: `imagemap[y*100+x]`. The pallete is kept separeted from the image map.

Known channel names are:

```
r - red channel of RGB or RGBA images.
g - gree channel of RGB or RGBA images.
```

```
b - blue channel of RGB or RGBA images.
a - alpha channel of RGBA images.
m - indices channel of MAP images (valid only for cdBitmap objects).
p - colors table of MAP images (valid only for cdBitmap objects). It is a palette
```

Server Images

It is a high performance image compatible with a specific canvas. It is faster than user image functions, but less flexible. It is commonly used for off-screen drawing in Window Systems.

You can make gets and puts on several canvases but they must be created using the same driver. It is possible to specify a part of the image to be drawn, but it is not possible to zoom.

It is called "server" images because the data is stored in a system private format, that the application (or the client) does not have access.

To create a server image there must be an active canvas of a driver with server image support.

---

void* cdCreateImage(int w, int h); [in C]
cd.CreateImage(w, h: *number*) -> (image: *image_tag*) [in Lua]

Creates a compatible image with size = **w x h** pixels. A compatible image has the same color representation (number of bits per pixel) of the active canvas. Once the server image is created it is independent of the active canvas. The server image can only be used with an other canvas of the same type as the canvas that was active when the image was created. The default background is the same as the canvas, **CD_WHITE**.

void cdKillImage(void *image); [in C]
cd.KillImage(image: *image_tag*) [in Lua]

Liberates memory allocated for the image. It is not necessary to have an active canvas to call this function.

void cdGetImage(void *image, int x, int y); [in C]
cd.GetImage(image: *image_tag*; x, y: *number*) [in Lua]

Copies a rectangular region from the current rectangular context to the memory **(image)**. **(x,y)** is the coordinate of the bottom left corner of the rectangular region. The width and length of the rectangular region are defined in the image structure (when the image is created).

void cdPutImageRect(void *image, int x, int y, int xmin, int xmax, int ymin, int ymax); [in C]
void wdPutImageRect(void* image, double x, double y, int xmin, int xmax, int ymin, int ymax); (WC) [in C]
cd.PutImageRect(image: *image_tag*; x, y, xmin, xmax, ymin, ymax: *number*) [in Lua]
cd.wPutImageRect(image: *image_tag*; x, y, xmin, xmax, ymin, ymax: *number*) (WC) [in Lua]

Copies an image in a rectangular region of the canvas with the bottom left corner in **(x,y)**. Allows specifying a rectangle inside the image to be drawn, if **xmin, xmax, ymin** and **ymax** are 0, then the whole image is assumed.

void cdScrollArea(int xmin, int xmax, int ymin, int ymax, int dx, int dy); [in C]
cd.ScrollArea(xmin, xmax, ymin, ymax, dx, dy: *number*) [in Lua]

Copies the rectangle defined by the coordinates **(xmin,ymin)** and **(xmax,ymax)** to the rectangle defined by **(xmin+dx,ymin+dy)** and **(xmax+dx,ymax+dy)**. It has the same effect as **cdGetImage** followed by **cdPutImage**, but it should be faster and does not require the explicit creation of an image to be executed. Note that the region belonging to the first rectangle, but not to the second, remains unchanged (the function does not clean this region).

System

char* cdVersion(void); [in C]
cd.Version() -> (version: *string*) [in Lua]

Returns the current version number of the library. The string with the version number has a format "`major.minor.build`". For instance, the string "2.1.3" has number 2 as the main (major) version number, 1 as the secondary (minor) version number, and 3 as the build number. The major version number represents a change in the structure or behavior of functions; the minor version number represents one or more new drivers and functions added to the library; and the build version number represents one or more corrected bugs.

Metafile Interpretation

int cdPlay(cdContext* ctx, int xmin, int xmax, int ymin, int ymax, void *data); [in C]
cd.Play(ctx, xmin, xmax, ymin, ymax: *number*, data: *string*) -> (status: *number*) [in Lua]

Interprets the graphical contents (primitives and attributes) in a given driver and calls equivalent functions of the CD library using the active canvas. The primitives are drawn inside the region defined by the given coordinates. If `xmin`, `xmax`, `ymin` and `ymax` are all "0", the primitives will be drawn with their coordinates having the original values in the file. Only some drivers implement this function.

Drivers available:

- **CD_CLIPBOARD** = Clipboard, `data` is ignored.
- **CD_WMF** = Windows Metafile, `data` is a **char*** for the string "*filename*". Works only in the MS Windows system.
- **CD_EMF** = Windows Enhanced Metafile, `data` is a **char*** for the string "*filename*". Works only in the MS Windows system.
- **CD_CGM** = Computer Graphics Metafile ISO, `data` is a **char*** for the string "*filename*".
- **CD_METAFILE** = Metafile Canvas Draw, `data` is a **char*** for the string "*filename*".

int cdRegisterCallback(cdContext *ctx, int cb, int(*func)(cdContext *driver, ...)); [in C]
cd.RegisterCallback(ctx, cb: *number*, func: *function*) -> (status: *number*) [in Lua]

Used to customize the behavior of the **cdPlay** function. If you register a known callback function, it will be called during the processing loop of `cdPlay`.

The callback should return `CD_CONTINUE`, if it returns `CD_ABORT`, the `cdPlay` function is aborted. The callback identifiers of a given driver must be in the header file relative to that driver, with prefix "`CD_XXYYYCB`", where `XX` identifies that driver and `YYY` identifies the callback name.

There is a default callback common to all implementations of `cdPlay`, **CD_SIZECB**. Its definition is "`int cdResizeCB(cdContext *driver, int width, int height, double mm_width, double mm_height)`", and it returns the size of the image in the file before any function in the CD library is called, so that you can call the `cdPlay` function without an active canvas and create the canvas inside the callback. It works as a **cdGetCanvasSize** function.

Color Coding

The library's color system is RGB. In order to simplify some functions, a compact representation was created for the 3 values. To make a conversion from this representation to the 3 separate values and vice-versa, use functions `cdDecodeColor` and `cdEncodeColor`.

When the canvas used does not support more than 8 bpp of color resolution, you can use function `cdPalette` to give the driver an idea of which colors to prioritize. `cdPalette`'s behavior is driver dependent.

There are some predefined colors:

```
CD_RED          = (255,  0,  0)
CD_DARK_RED     = (128,  0,  0)
CD_GREEN        = (0  ,255,  0)
CD_DARK_GREEN   = (  0,128,  0)
CD_BLUE         = (  0,  0,255)
CD_DARK_BLUE    = (  0,  0,128)
CD_YELLOW       = (255,255,  0)
CD_DARK_YELLOW  = (128,128,  0)
CD_MAGENTA      = (255,  0,255)
CD_DARK_MAGENTA = (128,  0,128)
CD_CYAN         = (  0,255,255)
CD_DARK_CYAN    = (  0,128,128)
CD_WHITE        = (255,255,255)
CD_BLACK        = (  0,  0 , 0)
CD_DARK_GRAY    = (128,128,128)
CD_GRAY         = (192,192,192)
```

---

long int cdEncodeColor(unsigned char red, unsigned char green, unsigned char blue) [in C]
cd.EncodeColor(r, g, b: *number*) -> (old_color: *color_tag*) [in Lua]

Returns a codified triple (*r*,*g*,*b*) in a long integer such as **0x00RRGGBB**, where **RR** are the red components, **GG** are the green ones and **BB** are the blue ones. The code is used in the CD library to define colors. It can be used without an active canvas.

void cdDecodeColor(long int color, unsigned char *red, unsigned char *green, unsigned char *blue) [in C]
cd.DecodeColor(color: *color_tag*) -> (r, g, b: *number*) [in Lua]

Returns the red, green and blue components of a color in the CD library. Can be used without an active canvas.

long int cdEncodeAlpha(long int color, unsigned char alpha) [in C]
cd.EncodeAlpha(color: *color_tag*, alpha: *number*) -> (color: *color_tag*) [in Lua]

Returns the given color coded with the alpha information. ATENTION: In the moment this is usefull only in Win32 with GDI+ active. Se in Windows Using GDI+ Base Driver. The internal representation of the component is inverted, because the default value must be 0 and opaque for backward compatibility, so you should use **cdDecodeAlpha** to retrieve the alpha component.

unsigned char cdDecodeAlpha(long int color) [in C]
cd.DecodeAlpha(color: *color_tag*) -> (a: *number*) [in Lua]

Returns the alpha component of a color in the CD library. Can be used without an active canvas. 0 is transparent, 255 is opaque.

unsigned char cdRed(long int color); [in C]
cd.Red(color: *color_tag*) -> (r: *number*) [in Lua]

Macro that returns the red component of a color in the CD library. Can be used without an active canvas.

unsigned char cdGreen(long int color); [in C]
cd.Green(color: *color_tag*) -> (g: *number*) [in Lua]

Macro that returns the green component of a color in the CD library. Can be used without an active canvas.

unsigned char cdBlue(long int color); [in C]
cd.Blue(color: *color_tag*) -> (b: *number*) [in Lua]

Macro that returns the blue component of a color in the CD library. Can be used without an active canvas.

---

int cdGetColorPlanes(void); [in C]
cd.GetColorPlanes() -> (bpp: *number*) [in Lua]

Returns a given number, for instance *p*, which defines the number of colors supported by the current device as 2^p, representing the number of bits by pixel.

void cdPalette(int n, long int *color, int mode); [in C]
cd.Palette(palette: *palette_tag*; mode: *number*) [in Lua]

In systems limited to 256 palette colors, this function aims at adding **n** colors to the system's palette. In such systems, the colors demanded forward or backward which are not in the palette are approximated to the closest available color. The type can be **CD_FORCE** or **CD_POLITE**. **CD_FORCE** ignores the system colors and interface elements, since the menus and dialogues may be in illegible colors, but there will be more colors available. **CD_POLITE** is the recommended type. It must always be used before drawing. It cannot be queried.

cd.CreatePalette(size: number) -> (palette: palette_tag) [in Lua Only]

Creates a palette. The object returned can be used as a table to set palette entries (palette[0], palette[1],..), each entry is a color object.

cd.KillPalette(palette: palette_tag) [in Lua Only]

Destroys the created palette and liberates allocated memory.

Drivers

Driver is the implementation of functions of a canvas for a specific canvas type. In other words it represents the context in which the canvas is situated. For example, a Window System that has windows on which you can draw.

It can be portable, platform independent, or it can has a different implementation in each platform. In this case its functions may have different behaviors, but the library is implemented in such a way that these differences are minimized.

CD_IUP - IUP Driver (cdiup.h)

This driver provides access to an interface element of a IUP canvas. IUP is a portable user-interface library used to create portable user-interface applications.

Use

The canvas is created by means of a call to the function **cdCreateCanvas**(CD_IUP, Data), after which other CD functions can be called as usual. This function creates a CD canvas based on the existing IUP canvas. The parameter Data is a pointer to a handle of the IUP canvas (Ihandle*). For use with CDLUA, a canvas created with IUPLUA must necessarily be passed as parameter.

Any amount of such canvases may exist simultaneously, but they should not use the same IUP canvas. It is important to note that a call to function **cdKillCanvas** is required to **close** the file properly.

The CD canvas is automatically stored in the IUP canvas as the "**_CD_CANVAS**" attribute.

This driver can totally be replaced by the Native Window driver using IUP's attribute. The canvas' creation changes to:

```
myCdCanvas = cdCreateCanvas(CD_NATIVEWINDOW, IupGetAttribute(myIupCanvas, "CONID"
IupSetAttribute(myIupCanvas, "_CD_CANVAS", myCdCanvas);
```

To use this driver, it must be linked with the "**cdiup**" library (`cdiup.lib` in Windows, `cdiuplib.a` in UNIX).

In Lua, it is necessary to call function **cdluaiup_open()** after a call to function **cdlua_open()**, apart from linkediting with the "**cdluaiup**" library.

To use this driver in Windows using GDI+ is necessary to call the **cdInitGdiPlusIUP()** function once. And to link with the "**cdiupgdiplus**" library.

Behavior of Functions

This driver is greatly platform-dependent, but little dependent on the IUP library. For further detail, see the **Behavior of Functions** in each platform: Microsoft Windows (GDI), Windows Using GDI+, X-Windows (XLIB). However, it should be noted that some functions behave differently from the basic functions of each platform.

Control

- **cdActivate**: updates the canvas size; the IUP canvas might have been resized.

Exclusive Attributes

- "**WINDOWRGN**": set the shape of a window to the current complex clipping region (set only). If data is NULL the region is reset.

CD_NATIVEWINDOW - Native Window Driver (cdnative.h)

This driver provides access to an existing Native Window, a basic element of the user-interface system. It also provides access to other native handles like HDC handles in Windows.

Use

The canvas is created by means of a call to the function **cdCreateCanvas**(`CD_NATIVEWINDOW, Data`), after which other functions in the CD library can be called as usual. This function **creates** a CD canvas based on an existing system canvas. The parameter `Data` is a pointer to a handle of the canvas. It is system-dependent, having a different meaning in each platform:

> **Microsoft Windows**: can be the handle of the Windows window (`HWND`), or the handle of a previously created Device Context (`HDC`), or can be a string in the format "`hdc width height`" or, in C, "`%p %d %d`". To get the entire screen use a NULL data. In Lua it must be a userdata with the HWND or HDC pointers.
> **X-Windows**: It is a string in the format "`display window`" or, in C, "`%p %lu`" (uses the default screen).

The given parameters must exists until **cdKillCanvas** is called. The `HDC` is released only if created inside **cdCreateCanvas** from an `HWND` or when data is NULL.

Any amount of such canvases may exist simultaneously, but they should not use the same window, except if you are using a GDI canvas and a GDI+ canvas at the same time.

The string has the same format as the IUP attribute **CONID**, so it is possible to create a canvas using this driver, but based on a IUP canvas. Therefore, it is no longer necessary to use the IUP driver. For instance:

```
cdCreateCanvas(CD_NATIVEWINDOW, IupGetAttribute(myIupCanvas,
"CONID")).
IupSetAttribute(myIupCanvas, "_CD_CANVAS", myCdCanvas);
```

In CDLUA, the creation parameter must be a string in X-Windows and a userdata in Microsoft Windows.

Exclusive Functions

```
void cdGetScreenSize(int *width, int *height, double *width_mm, double
*height_mm); [in C]
cdGetScreenSize() -> (width, heigth, mm_width, mm_height: number) [in Lua]
```

Equivalent to function **cdGetCanvasSize**, but returns the values relative to the main screen of the window system. It is not necessary to have an active canvas to call this function.

```
int cdGetScreenColorPlanes(void); [in C]
cdGetScreenColorPlanes() -> (bpp: number) [in Lua]
```

Equivalent to function **cdGetColorPlanes**, but returns the value relative to the main screen of the window system. It is not necessary to have an active canvas to call this function.

Behavior of Functions

This driver is greatly platform-dependent. For further detail, see the **Behavior of Functions** in each platform: Microsoft Windows (GDI), Windows Using GDI+, X-Windows (XLIB). However, it should be noted that some functions behave differently from the basic functions of each platform.

Control

- **cdActivate**: updates the canvas size; the window might have been resized. If the canvas was created using a HDC, the size will not be updated.

Exclusive Attributes

- "**WINDOWRGN**": set the shape of a window to the current complex clipping region (set only). If data is NULL the region is reset.

CD_CLIPBOARD - Clipboard Driver (cdclipbd.h)

This driver allows the access to a Clipboard area. It is greatly dependent on the system. In Win32, it creates an Enhanced Metafile, a **Bitmap** or a CD Metafile; in X-Windows it creates only a CD Metafile.

Use

The canvas is created by means of a call to function **cdCreateCanvas**(CD_CLIPBOARD, Data), after which other functions in the CD library can be called as usual. The Data parameter string is platform-

dependent and varies according to the metafile created. See each metafile's documentation, but remember to exclude parameter `"filename"`.

In the Windows environment, if the string `"-b"` is present, it means that a **Bitmap** must be created instead of a metafile, and, if the string `"-m"` is specified, a **CD Metafile** will be created. For a **Bitmap** the remaining string must contains the bitmap size and optionally its resolution: *"-b widthxheight [resolution]"* or in C "***%dx%d %g"***, the resolution default is the screen resolution.

In the X-Windows environment, the Display `("%p")` where the data will be stored must be passed as a parameter before the **CD Metafile** parameters. This environment's driver is used only for applications that use CD to communicate with each other, because a CD Metafile is created.

Any amount of such canvases may exist simultaneously. It is important to note that a call to function **`cdKillCanvas`** is required to properly copy the data to the Clipboard.

You can interpret the data from the Clipboard using function **`cdPlay`**. In the X-Windows environment, the parameter `"data"` for the **`cdPlay`** function is the pointer to the Display where the metafile will be obtained.

Behavior of Functions

This driver is greatly platform-dependent. For further detail, see the **Behavior of Functions** in each platform: Microsoft Windows (GDI), X-Windows (XLIB). However, it should be noted that some functions behave differently from the basic functions of each platform.

CD_PRINTER - Printer Driver (cdprint.h)

This driver provides access to a System Default Printer.

Currently, it works only in Microsoft Windows platforms, but it is possible to use it in other platforms without the risk of compilation error. If you attempt to create a canvas in another platform, the function **`cdCreateCanvas`** will return NULL.

Use

The canvas is created by calling function **`cdCreateCanvas`**`(CD_PRINTER, Data)`, after which other CD functions can be called as usual. The `Data` string has the following format:

"*name* [-d]" *or in C style "****%s -d****"*

`Name` is an optional document name that will appear in the printer queue. Optionally, `-d` displays the System Printer dialogue box before starting to print, allowing you to configure the printer's parameters. When using this parameter and the return canvas is NULL, one must assume that the print was canceled by the user.

Any amount of such canvases may exist simultaneously. It is important to note that a call to function **`cdKillCanvas`** is required to properly send the data to the printer.

Pages - Use **`cdFlush`** to change to a new page. You can draw first on page 1, then on page 2 and so forth.

Behavior of Functions

This driver is greatly platform-dependent. For further detail, see the **Behavior of Functions** in each platform: Microsoft Windows (GDI), Windows Using GDI+, X-Windows (XLIB). However, it should be noted that some functions behave differently from the basic functions of each platform.

A printer created in Win32s has the same limitations as the WMF driver. In Windows 95 or NT, it has the same limitations as the EMF driver.

Control

- **cdFlush**: changes to a new page, preserving the previous one. In the Win32 base driver, after the first page, function **cdText** draws the text below its correct position - we do not know why this happens.

Attributes

- **cdHatch**: opaque in Win32 base driver (GDI).

CD_IMAGERGB - RGB Client Image Driver (cdirgb.h)

This driver allows access to a Client Image, an imaged based in RGB colors with 24 bits per pixel (8 per channel). It is used to implement high-quality offscreen drawings, but is slower than the Server Image version. In fact, it is a rasterizer, that is, it converts vector primitives into a raster representation. All primitives are implemented by the library and are not system-dependent (the primitives of the Server Image version are system-dependent).

Use

The canvas is created by means of a call to the function **cdCreateCanvas**(CD_IMAGERGB, Data), after which other functions in the CD library can be called as usual. The function creates an RGB image, and then a CD canvas. The Data parameter string has the following format:

"widthxheight [r g b] -r[resolution]" or in C "***%dx%d %p %p %p -r%g***"

It must include the canvas' dimensions. Width and height are provided in pixels (note the lowercase "x" between them). As an option, you can specify the buffers to be used by the driver, so that you can draw over an existing image. The resolution can be defined with parameter -r; its default value is "3.8 pixels/mm" (96.52 DPI).

Any amount of such canvases may exist simultaneously. It is important to note that a call to function **cdKillCanvas** is required to **close** the file properly.

In CDLUA, the canvas can be created in two ways: with an already defined image or without it. With an image, an RGB image must be passed as parameter, created by functions **cdCreateImageRGB** or **cdCreateBitmap** in Lua. The resolution must be passed as in the string above, in an extra parameter after the image. Without an image, the parameter passed must be a string in the format above, excluding the *r*, *g* and *b* pointers.

Exclusive Functions in this Driver

unsigned char * cdRedImage(cdCanvas* canvas); [in C]

Returns the red image buffer. The buffer's format is compatible with the Client Image specification.

unsigned char * cdGreenImage(cdCanvas* canvas); [in C]

Returns the green image buffer. The buffer's format is compatible with the Client Image specification.

unsigned char * cdBlueImage(cdCanvas* canvas); [in C]

Returns the blue image buffer. The buffer's format is compatible with the Client Image specification.

cdImageRGB(canvas: canvas_tag) -> (image: imagergb_tag) [in Lua]

Returns the canvas' internal RGB image.

Behavior of Functions

All primitives are from the Simulation driver, see the Simulation driver's documentation for further information.

Control

- **cdFlush**: does nothing.
- **cdPlay**: does nothing, returns CD_ERROR.

Coordinate System and Clipping

- **cdUpdateYAxis**: does nothing. The axis orientation is the same as the CD library's.
- Polygon clipping is not performed for Server and Client Images.

Colors

- **cdGetColorPlanes**: returns 24.
- **cdPalette**: does nothing.

Exclusive Attributes

- "**REDIMAGE**", "**GREENIMAGE**", "**BLUEIMAGE**": return the respective pointers of the canvas RGB image (get only). Alternative for the functions **cdRedImage**, **cdGreenImage**, **cdBlueImage**. Not accessible in Lua.

CD_IMAGE - Server Image Driver (cdimage.h)

This driver provides access to a Server Image, a memory-based high-performance image that corresponds to the attributes of the system's devices. It is used for offscreen drawings.

Use

The canvas is created by means of a call to function **cdCreateCanvas**(CD_IMAGE, Data), after which other functions in the CD library can be called as usual. The function creates a CD canvas based on an existing Server Image. The Data parameter must be a pointer to an image created with function **cdCreateImage**.

Any amount of such canvases may exist simultaneously. It is important to note that a call to function **cdKillCanvas** is required to properly **end** the driver. You can call function **cdKillImage** only after calling **cdKillCanvas**.

For use with CDLUA, the Server Image passed as parameter must have been created with function **cdCreateImage** in Lua.

Behavior of Functions

This driver is greatly platform-dependent. For further detail, see the **Behavior of Functions** in each platform: Microsoft Windows (GDI), Windows Using GDI+, X-Windows (XLIB). However, it should be noted that some functions behave differently from the basic functions of each platform.

CD_DBUFFER - Double Buffer Driver (cddbuf.h)

Implements the concept of offscreen drawing.

Use

The canvas is created by means of a call to function **`cdCreateCanvas`**`(CD_DBUFFER, Data)`, after which other functions in the CD library can be called as usual. This function creates a CD canvas based on an existing window canvas (Native Windows or IUP). The parameter `Data` is a pointer to the already created canvas.

Any amount of such canvases may exist simultaneously. It is important to note that a call to function **`cdKillCanvas`** is required to properly **end** the driver. Call function **`cdKillCanvas`** in this driver before calling **`cdKillCanvas`** in the window driver.

The drawing functions will function normally, as if they were drawing in the server image driver. When function **`cdFlush`** is executed, the image is drawn in the window canvas passed as parameter in the canvas creation.

When the window's size changes, the server image is automatically recreated in the same size as the canvas. This is done in the function **`cdActivate`**.

Behavior of Functions

This driver is greatly platform-dependent. For further detail, see the **Behavior of Functions** in each platform: Microsoft Windows (GDI), Windows Using GDI+, X-Windows (XLIB). However, it should be noted that some functions behave differently from the basic functions of each platform.

CD_PDF - PDF Driver (cdpdf.h)

This drivers allows generating a PDF file. This format developed for representing documents in a manner that is independent of the original application software, hardware, and operating system used to create those documents. The format's copyrights are property of Adobe Systems.

This driver is very similar to the PS driver but it uses the PDFlib library to generate the PDF (http://www.pdflib.com/). There are two PDFlib licenses available, one commercial and one free with a flexible license, see PDFlib Lite License. The CD_PDF driver works with both versions, but by default uses the PDF Lite version code. The configuration of the PDF Lite code excludes other image file formats.

Use

The file is created and opened by calling function **`cdCreateCanvas`**`(CD_PDF, Data)`, in which `Data` contains the filename and canvas dimensions. This function opens the file and writes its header.

Then, other functions in the CD library can be called as usual. The `Data` parameter string has the following format:

"*filename -p[paper] -w[width] -h[height] -s[resolution] [-o]*"
or in C
"%s -p%d -w%g -h%g -s%d -o"

The filename must be inside double quotes (") if it has spaces. Any amount of such canvases may exist simultaneously. It is important to note that a call to function **`cdKillCanvas`** is required to **close** the file properly.

Paper Size - The default paper size is A4. It is possible to change it by using one of the predefined sizes - **`CD_A0`**, **`CD_A1`**, **`CD_A2`**, **`CD_A3`**, **`CD_A4`**, **`CD_A5`**, **`CD_LETTER`** and **`CD_LEGAL`** - with parameter "`-p`". It is also possible to define a paper in a particular size by using parameters "`-w`" e "`-h`". Values are provided in millimeters.

Default Paper Sizes

	Width (mm)	Length (mm)
A0	841	1187
A1	594	841
A2	420	594
A3	297	420
A4	210	297
A5	148	210
Letter	216	279
Legal	216	356

Resolution - Resolution is used to convert values from millimeters to pixels (the same as points, but the number of points is per inch - DPI). Use parameter "`-s`" to configure the resolution. The default value is 300 DPI.

Orientation - The page can be oriented as portrait or landscape. The default value is portrait, but when the parameter "-o" is used, the horizontal and vertical values are switched.

Behavior of Functions

Control

- **`cdPlay`**: does nothing, returns `CD_ERROR`.
- **`cdFlush`**: changes to a new page, preserving the previous one.
- **`cdClear`**: does nothing.

Coordinate System & Clipping

- **`cdUpdateYAxis`**: does nothing.
- **Complex Regions**: not supported.

Attributes

- **`cdBackground`** does nothing, returns `CD_WHITE`.
- **`cdBackOpacity`**: does nothing, returns `CD_TRANSPARENT`.
- **`cdWriteMode`**: does nothing, returns `CD_REPLACE`.

- **cdNativeFont**: the font string describer has the following format:
"fontname, size [style] [-k] [-u]", where fontname is the name of the font in Windows (notice the comma after the font name), size is given in points or in pixels, style is the same as cdFont, -u means underline and -k means strikethrough, or "-d" to show the font-selection dialogue box. However, this function also accepts the font string used by the WINFONT attribute of the IUP library.
- **cdHatch**: is always opaque.
- **cdStipple**: is always opaque.
- **cdFont**: see the font mapping table for the equivalence used to map CD fonts into PS fonts:

Font Mapping

CD Fonts	PS Fonts
CD_SYSTEM	Courier
CD_COURIER	Courier
CD_TIMES_ROMAN	Times-Roman
CD_HELVETICA	Helvetica

Colors

- **cdGetColorPlanes**: returns 24.
- **cdPalette**: does nothing.

Client Images

- **cdGetImageRGB**: does nothing.
- **cdPutImageMap**: stores an RGB image.

Primitives

- **cdMark**: is simulated.
- **cdPixel**: does not exist in PDF, is simulated using a circle with radius=1.

Server Images

- All functions do nothing.

Exclusive Attributes

- "**POLYHOLE**": defines the index of a vertex where there is a hole in a closed polygon. It will affect the next cdEnd. Can be called several times between **cdBegin** and **cdEnd** to define holes. The value passed must be a string containing an integer ("%d"). If the value of the attribute passed is NULL, all holes will no longer be considered. When consulted returns the current number of holes ("%d"). It can have a maximum of 500 holes.

CD_PS - PostScript Driver (cdps.h)

This drivers allows generating a PostScript file. This format was created to be a high-quality graphics language for printers and is currently supported by several printers. If your printer supports PostScript, you can send the file generated by the driver directly to the printer port. Usually, the filename has an extension .PS or .EPS. The driver generates level-2 PostScript, therefore some PostScript viewers might present errors. The format's copyrights are property of Adobe Systems.

Use

The file is created and opened by calling function **cdCreateCanvas**(CD_PS, Data), in which Data contains the filename and canvas dimensions. This function opens the file and writes its header. Then, other functions in the CD library can be called as usual. The Data parameter string has the following format:

"*filename -p[paper] -w[width] -h[height] -l[left] -r[right] -b[bottom] -t[top] -s[resolution] [-e] [-g] [-o] [-1] d[margin]*"
or in C
"%s -p%d -w%g -h%g -l%g -r%g -b%g -t%g -s%d -e -o -1 -g -d%g"

The filename must be inside double quotes (") if it has spaces. Any amount of such canvases may exist simultaneously. It is important to note that a call to function **cdKillCanvas** is required to **close** the file properly.

Paper Size - The default paper size is A4. It is possible to change it by using one of the predefined sizes - **CD_A0**, **CD_A1**, **CD_A2**, **CD_A3**, **CD_A4**, **CD_A5**, **CD_LETTER** and **CD_LEGAL** - with parameter "-p". It is also possible to define a paper in a particular size by using parameters "-w" e "-h". Values are provided in millimeters.

Default Paper Sizes

	Width (mm)	Length (mm)
A0	841	1187
A1	594	841
A2	420	594
A3	297	420
A4	210	297
A5	148	210
Letter	216	279
Legal	216	356

Margins - The margins are controlled by parameters "-l" "-r" "-t" and "-b" (*left, right, top, bottom*). Values are provided in millimeters. Default margins are 25.4 mm to all parameters. You can draw only inside the margins.

Resolution - Resolution is used to convert values from millimeters to pixels (the same as points, but the number of points is per inch - DPI). Use parameter "-s" to configure the resolution. The default value is 300 DPI.

Orientation - The page can be oriented as portrait or landscape. The default value is portrait, but when the parameter "-o" is used, the horizontal and vertical values are switched.

EPS - The PostScript file can be in an **Encapsulated PostScript** format. For such, simply specify the parameter "-e". It is useful for other applications to import the PostScript file. You can define the margins of the bounding box by means of parameter "-d", in millimeters.

Debug - Parameter "-g" adds a series of comments to the PS file, making the beginning and end of a command from the CD library explicit. It is useful only for those who understand PostScript and wish to identify a problem. It considerably increases the file size.

Level 1 - Parameter "-1" forces the driver to generate a level-1 PostScript. In this case, pattern, stipple and hatch are not supported.

Pages - Use function cdFlush to change to a new page. The previous page will not be changed.

Behavior of Functions

Control

- `cdPlay`: does nothing, returns `CD_ERROR`.
- `cdFlush`: changes to a new page, preserving the previous one. Does nothing in EPS mode.
- `cdClear`: does nothing.

Coordinate System & Clipping

- `cdGetCanvasSize`: returns the page's size within the margins (drawing area).
- `cdUpdateYAxis`: does nothing.
- **Complex Regions**: not supported.

Attributes

- `cdBackground` does nothing, returns `CD_WHITE`.
- `cdBackOpacity`: does nothing, returns `CD_TRANSPARENT`.
- `cdWriteMode`: does nothing, returns `CD_REPLACE`.
- `cdFontDim`: is simulated.
- `cdNativeFont`: selects a PostScript font, given its name and size ("`fontname size`" or, in C style, "`%s %d`").
- `cdTextSize`: is simulated.
- `cdHatch`: is always opaque (to be implemented).
- `cdStipple`: is always opaque (to be implemented).
- `cdTextAlignment`: `Baseline` is the same as `South`.
- `cdFont`: see the font mapping table for the equivalence used to map CD fonts into PS fonts:

Font Mapping

CD Fonts	PS Fonts
CD_SYSTEM	Courier
CD_COURIER	Courier
CD_TIMES_ROMAN	Times
CD_HELVETICA	Helvetica

Colors

- `cdGetColorPlanes`: returns 24.
- `cdPalette`: does nothing.

Client Images

- `cdGetImageRGB`: does nothing.
- `cdPutImageMap`: stores an RGB image in the file (to be implemented).
- `cdPutImageRGBA`: alpha is ignored (to be implemented).

Primitives

- `cdMark`: is simulated.
- `cdPixel`: does not exist in PS, is simulated using a circle with radius=1.

Server Images

- All functions do nothing.

WC

- Implemented directly in the driver.

Exclusive Attributes

- "**POLYHOLE**": defines the index of a vertex where there is a hole in a closed polygon. It will affect the next cdEnd. Can be called several times between **cdBegin** and **cdEnd** to define holes. The value passed must be a string containing an integer ("%d"). If the value of the attribute passed is NULL, all holes will no longer be considered. When consulted returns the current number of holes ("%d"). It can have a maximum of 500 holes.
- "**CMD**": saves a string directly to the file. Allows adding PostScript commands to the file generated by the CD library. (set only)
- "**ROTATE**": allows the usage of 1 angle and 1 coordinate (x, y), that define a global rotation transformation centered in the specified coordinate. Use 1 real and 2 integer values inside a string ("%g %d %d" = angle x y).

CD_METAFILE - CD Metafile Driver (cdmf.h)

This driver allows the generation of a CD Metafile, a very simple format that includes calls to functions of the CD library and provides persistence to its primitives.

Use

The file is created by calling function **cdCreateCanvas**(CD_METAFILE, Data). The Data parameter is a string that must contain the filename and the canvas dimensions, in the following format:

"*filename* [widthxheight resolution]" or in *C use "**%s %gx%g %g**"*

Only the parameter filename is required. The filename must be inside double quotes (") if it has spaces. Width and height are provided in millimeters (note the lowercase "x" between them), and their default value in pixels is INT_MAX for both dimensions. Resolution is the number of pixels per millimeter; its default value is "3.8". Width, height and resolution are real values.

Any amount of such canvases may exist simultaneously. It is important to note that a call to function **cdKillCanvas** is required to **close** the file properly.

Images - Be careful when saving images in the file, because it uses a text format to store all numbers and texts of primitives, including images, which significantly increases its size.

Extension - Although this is not required, we recommend the extension used for the file to be ".MF".

Behavior of Functions

Coordinate System and Clipping

- **cdGetCanvasSize**: returns the size used in the call to function cdCreateCanvas.
- **cdUpdateYAxis**: does nothing.
- **Complex Regions**: not supported.

Attributes

- **cdFontDim**: uses a size estimator, returning approximate values.
- **cdTextSize**: uses a size estimator, returning approximate values.

Colors

- **cdGetColorPlanes**: always returns 24.

Client Images

- **cdGetImageRGB**: does nothing.

Server Images

- All functions do nothing.

WC

- Implemented directly in the driver.

CD_CGM - *Computer Graphics Metafile Driver* (cdcgm.h)

This driver allows generating a Computer Graphics Metafile, which is an ANSI standard for the persistent storage of graphics primitives. The file usually has an extension .CGM.

Use

The file file is created by means of a call to the function **cdCreateCanvas**(CD_CGM, Data), which **opens** the file and writes its header. Then, other functions in the CD library can be called as usual. The Data parameter string has the following format:

*"filename [widthxheight] [resolution] [-t] -p[precision]" or in C style "**%s %gx%g %g %s**"*

Only the parameter filename is required. The filename must be inside double quotes (") if it has spaces. Width and height are provided in millimeters (note the lowercase "x" between them). When the canvas' size is not specified, the VDC Extension saved to the file is the image's bounding rectangle. The resolution is the number of pixels per millimeter; its default value is "3.8". Width, height and resolution are real values. Parameter -t modifies the codification. Width, height and resolution are used only by **cdGetCanvasSize** and in pixel-millimeter conversion. Parameter -p specifies the precision of integers, which can be 16 (default) or 32.

Any amount of such canvases may exist simultaneously. It is important to note that a call to function **cdKillCanvas** is required to **close** the file properly.

Coding - The CGM format supports binary and text coding. If you are not sure what to do, use binary coding, which is the default. Should you prefer text coding, add a "-t" string to the Data parameter.

Precision of Coordinates - The primitives can use coordinates in real numbers. However, for compatibility reasons, we use coordinates in integers.

Behavior of Functions

Control

- **cdClear**: does nothing.
- **cdFlush**: creates a new image, preserving the previous one. The CGM format supports multiple images in a file.
- **cdPlay**: works with files created with text or binary coding. There are several callbacks for this driver. If one of the callbacks returns a value different from zero, cdPlay's processing is interrupted. The driver implements the

callback **CD_SIZECB** and other callbacks associated to CGM:
CD_COUNTERCB - `int(*cdcgmcountercb)(cdContext *driver, double percent)` - Executed for each header of CGM commands; returns the percentage (0-100%) of headers read.
CD_SCLMDECB - `int(*cdcgmsclmdecb)(cdContext *driver, short scl_mde, short *drw_mode, double *factor)` - Executed for the command CGM SCALE MODE. Returns the current CGM scale mode and allows the callback to modify the scale mode used by the `cdPlay` function `(ABSTRACT=0, METRIC=1)`. Should you choose the METRIC or ABSTRACT scale mode but the original scale mode is METRIC, you must provide the conversion factor in mm per pixel.
CD_VDCEXTCB - `int(*cdcgmvdcextcb)(cdContext *driver, short type, void *xmn, void *ymn, void *xmx, void *ymx)` - Executed for the CGM command CGM VDC EXTENT, returns the VDC SPACE.
CD_BEGPICTCB - `int(*cdcgmbegpictcb)(cdContext *driver, char *pict)` - Executed for the command BEGIN PICTURE, returns the string that describes the image.
CD_BEGPICTBCB - `int(*cdcgmbegpictbcb)(cdContext *driver)` - Executed for the command BEGIN PICTURE BODY.
CD_CGMBEGMTFCB - `int (*cdcgmbegmtfcb)(cdContext *driver, int *xmin, int *ymin, int *xmax, int *ymax)` - Executed for the command BEGIN METAFILE, provides the drawing limits of the image in the file.

Coordinate System and Clipping

- **cdUpdateYAxis**: does nothing. The axis orientation is the same as the CD library.
- **Complex Regions**: not supported.

Primitives

- **cdBegin**: if parameter **CD_CLIP** or **CD_BEZIER** are specified, does nothing.
- **cdMark**: `CD_DIAMOND`, `CD_HOLLOW_DIAMOND` and `CD_HOLLOW_BOX` are equivalent to `CD_BOX`, and `CD_HOLLOW_CIRCLE` is equivalent to `CD_CIRCLE`.
- **cdPixel**: does not exist in CGM, is simulated using a mark with size 1.
- **cdChord**: does nothing.

Attributes

- **cdWriteMode**: does nothing, returns `CD_REPLACE`.
- **cdFontDim**: is simulated.
- **cdNativeFont**: does nothing.
- **cdFillMode**: does nothing.
- **cdLineCap**: does nothing.
- **cdLineJoin**: does nothing.
- **cdTextSize**: is simulated.
- **cdTextOrientation**: does nothing.
- **cdFont**: see the font mapping table for the equivalence used to map CD fonts into CGM fonts:

Font Mapping

CD Fonts	CGM Fonts
CD_SYSTEM	SYSTEM
CD_COURIER	COURIER
CD_TIMES_ROMAN	TIMES_ROMAN
CD_HELVETICA	HELVETICA

Colors

- **cdGetColorPlanes**: returns 24.
- **cdPalette**: does nothing.

Client Images

- **cdGetImageRGB**: does nothing.
- **cdPutImageRGBA**: alpha is ignored.

Server Images

- All functions do nothing.

CD_DGN - MicroStation Design File Driver (cddgn.h)

This driver allows generating a MicroStation design file. The file name usually has an extension .DGN. The driver supports only MicroStation version 4.0 or later. The format's copyrights are property of Bentley Systems.

Use

The file is created and opened by calling function **cdCreateCanvas**(CD_DGN, Data), in which Data contains the filename and canvas dimensions. This function opens the file and writes its header. Then, other functions in the CD library can be called as usual. The Data parameter string has the following format:

"filename [widthxheight] [resolution] [-f] [-sseedfile]" or in C *"**%s %gx%g %g %s**"*

Only the parameter filename is required. The filename must be inside double quotes (") if it has spaces. Width and height are provided in millimeters (note the lowercase "x" between them), and their default value in pixels is INT_MAX for both dimensions. Resolution is the number of pixels per millimeter; its default value is "3.8". Width, height and resolution are real values. Parameter -f modifies the polygon filling's behavior. Just as in MicroStation, you can specify a seed file using parameter -s. Width, height and resolution are used only by **cdGetCanvasSize** and in pixel-millimeter conversion.

Any amount of such canvases may exist simultaneously. It is important to note that a call to function **cdKillCanvas** is required to close the file properly.

Images and Colors - The DGN format does not support server images and works with an indexed-color format. Color quality is limited to 256 colors, and the format uses a uniform palette to convert RGB colors into palette indices. If you configure a palette, the color conversion process will become slower.

Filling - Up to version 5.0, MicroStation presents some limitations for polygon filling. You can disable filling by means of string "-f" in the Data parameter. Filled polygons can only have around 10,000 vertices; if the value is larger, the polygon style changes to closed lines.

Seed - In the seed file, several DGN parameters can be defined to be used in the drawing. The library offers a default seed file, called "SEED2D.DGN". The file's location depends on the environment variable **CDDIR**.

Behavior of Functions

Control

- **cdClear**: does nothing.
- **cdPlay**: does nothing, returns CD_ERROR.

Coordinate System and Clipping

- **cdClip**: does nothing (no clipping function is supported), returns CD_CLIPOFF.
- **cdClipArea**: does nothing.
- **cdGetClipArea**: does nothing, returns CD_CLIPOFF.
- **cdUpdateYAxis**: does nothing. The axis orientation is the same as the CD library.
- **Complex Regions**: not supported.

Primitives

- **cdBegin**: if parameter **CD_CLIP** or **CD_BEZIER** are specified, does nothing.
- **cdMark**: is simulated.
- **cdChord**: does nothing.

Attributes

- **cdBackOpacity**: does nothing, returns CD_OPAQUE.
- **cdWriteMode**: does nothing, returns CD_REPLACE.
- **cdInteriorStyle**: does nothing.
- **cdFillMode**: does nothing.
- **cdLineCap**: does nothing.
- **cdLineJoin**: does nothing.
- **cdHatch**: does nothing.
- **cdStipple**: does nothing.
- **cdPattern**: does nothing.
- **cdTextSize**: returns a bounding box which is usually larger than the text (the computation is based on the widest character).
- **cdTextAlignment**: uses **cdTextSize**, therefore is not precise.
- **cdNativeFont**: selects the font number defined in MicroStation ("%d" in C style).
- **cdFont**: See the font mapping table for the equivalence used to map CD fonts into MicroStation fonts:

Font Mapping

CD Fonts	MicroStation Fonts
CD_SYSTEM / CD_HELVETICA	#0 - Standard
CD_COURIER	#43 - Low_res_filled
CD_TIMES_ROMAN	#2 - Fancy

Colors

- **cdGetColorPlanes**: returns 8 (MicroStation uses a palette with 256 values).
- **cdBackground**: always returns CD_WHITE.

Client Images

- **cdGetImageRGB**: does nothing.
- **cdPutImageRGB**: considering that the format supports only 256 colors, image quality is quite poor.
- **cdPutImageRGBA**: alpha is ignored.
- **cdPutImageMap**: considering that the format supports only 256 colors, image quality is quite poor.

Server Images

- All functions do nothing.

CD_DXF - AutoCAD Image Exchange File Driver (cddxf.h)

This driver allows generating an AutoCAD image exchange file. The file name usually has an extension .DXF. This driver supports only AutoCAD version 10.0 or later. The format's copyrights are

property of Autodesk.

Use

The file is created and opened by calling function **cdCreateCanvas**(CD_DXF, Data), in which Data contains the file name and canvas dimensions. This function opens the file and writes its header. Then, other functions in the CD library can be called as usual. The Data parameter string has the following format:

"filename [widthxheight] [resolution]" or in C *"**%s %gx%g %g**"*

Only the parameter filename is required. The filename must be inside double quotes (") if it has spaces. Width and height are provided in millimeters (note the lowercase "x" between them), and their default value in pixels is INT_MAX for both dimensions. Resolution is the number of pixels per millimeter; its default value is "3.8". Width, height and resolution are given in real values and can be used only by **cdGetCanvasSize** and in pixel-millimeter conversion.

Any amount of such canvases may exist simultaneously. It is important to note that a call to function **cdKillCanvas** is required to close the DXF file properly.

Images - The DXF format does not support client or server images and works with an indexed-color format (color quality is limited to 256 fixed colors).

Precision of Coordinates - The primitives use coordinates in real numbers.

Layers - The format can work with several layers. It is necessary to draw the primitives of layer '0' first, then layer '1' and so on. Use functions **cdFlush** to change the current layer.

Behavior of Functions

Control

- **cdFlush**: changes the current layer (the initial layer is '0', followed by '1' and so on).
- **cdClear**: does nothing.
- **cdPlay**: does nothing, returns CD_ERROR.

Coordinate System and Clipping

- **cdClip**: does nothing (no clipping function is supported), returns CD_CLIPOFF.
- **cdClipArea**: does nothing.
- **cdGetClipArea**: does nothing, returns CD_CLIPOFF.
- **cdUpdateYAxis**: does nothing. Axis orientation is the same as in the CD library.
- **Complex Regions**: not supported.

Primitives

- **cdBox**: draws only the box's borders (no filling function is supported). Behaves like **cdRect**.
- **cdSector**: draws a "hollow" sector, that is, only its borders.
- **cdMark**: besides employing simulation, filled marks are drawn in the same way as their corresponding empty marks, that is, only the borders.
- **cdBegin**: **CD_FILL** is mapped to **CD_CLOSED_LINES**. if parameter **CD_CLIP** or **CD_BEZIER** are specified, does nothing.
- **cdChord**: does nothing.

Attributes

- **cdBackOpacity**: does nothing, returns CD_TRANSPARENT.
- **cdWriteMode**: does nothing, returns CD_REPLACE.
- **cdMarkType**: since no filling function is supported, configuring mark type to solid type is the same as configuring it to its corresponding empty type (that is, CD_BOX corresponds to CD_HOLLOW_BOX, and so on).
- **cdInteriorStyle**: does nothing (filling is not supported), returns 0.
- **cdHatch**: does nothing.
- **cdFillMode**: does nothing.
- **cdLineCap**: does nothing.
- **cdLineJoin**: does nothing.
- **cdStipple**: does nothing.
- **cdPattern**: does nothing.
- **cdTextSize**: returns a bounding box usually larger than the text (the computation is based on the widest character).
- **cdNativeFont**: does nothing.
- **cdTextOrientation**: does nothing.
- **cdFont**: italic styles correspond to the basic styles with an inclination of 15^{o}. See the font mapping table for the equivalence used to map fonts of the CD library into AutoCAD fonts.

Font Mapping

CD Fonts	AutoCAD Fonts
CD_SYSTEM	STANDARD (sem arquivo)
CD_COURIER / CD_PLAIN	ROMAN (romanc.shx)
CD_COURIER / CD_PLAIN	ROMAN_BOLD (romant.shx)
CD_TIMES_ROMAN / CD_PLAIN	ROMANTIC (rom_____.pfb)
CD_TIMES_ROMAN / CD_BOLD	ROMANTIC_BOLD (romb_____.pfb)
CD_HELVETICA / CD_PLAIN	SANSSERIF (sas_____.pfb)
CD_HELVETICA / CD_BOLD	SANSSERIF_BOLD (sasb____.pfb)

Colors

- **cdForeground**: indexes `long int *color` in the fixed palette (AutoCAD uses a 256-color palette - for further detail, see AutoCAD's Reference Manual).
- **cdBackground**: does nothing, returns CD_WHITE.
- **cdGetColorPlanes**: returns 8.
- **cdPalette**: does nothing (the palette is fixed).

Client Images

- All functions do nothing.

Server Images

- All functions do nothing.

CD_EMF - Enhanced Metafile Driver (cdemf.h)

This driver allows generating a Microsoft Windows Enhanced Metafile, the format used by 32-bit Windows systems to store graphics primitives. Usually, the filename has an extension "*.emf".

The driver works only in the Microsoft Windows platform, but you can use it in other platforms without the

risk of compilation error. If you attempt to create a canvas in another platform, function **cdCreateCanvas** will return NULL.

Use

The canvas is created by means of a call to function **cdCreateCanvas**(CD_EMF, Data), after which other CD functions can be called as usual. Parameter Data has the following format:

"filename widthxheight" or in C *"**%s %dx%d**"*

It must include the filename and the canvas' dimensions. The filename must be inside double quotes (") if it has spaces. Width and height are provided in pixels (note the lowercase "x" between them). Resolution (the number of pixels per millimeter) is always the screen resolution.

Any amount of such canvases may exist simultaneously. Function **cdCreateCanvas** **opens** the file, and a call to function **cdKillCanvas** is required to **close** the file properly.

Behavior of Functions

This driver is greatly platform-dependent. For further detail, see the **Behavior of Functions** of the Microsoft Windows (GDI) or Windows Using GDI+ platform base drivers. It has been noticed that EMF, when saved in the Windows 95 environment, is not totally compatible with EMF saved in the Windows NT environment.

Control Functions

- **cdPlay**: different from the basic driver, is implemented. Not implemented using GDI+.
- **cdClear**: different from the basic driver, does nothing.

Client Images

- **cdGetImageRGB**: does nothing.
- **cdPutImageRGBA**: the alpha component is ignored. Using GDI+ works normally.

Server Images

- All functions do nothing.

CD_WMF - Windows Metafile Driver (cdwmf.h)

This driver allows creating a Microsoft Windows Metafile, the format used by 16-bit Windows systems to store graphics primitives. Usually, the filename has an extension "*.wmf".

The driver works only in the Microsoft Windows platform, but you can use it in other platforms without the risk of compilation error. If you attempt to create a canvas in another platform, function **cdCreateCanvas** will return NULL.

Use

The canvas is created by means of a call to the function **cdCreateCanvas**(CD_WMF, Data), after which other functions in the CD library can be called as usual. The Data parameter string has the following format:

"filename widthxheight [resolution]*"* or in C ***"%s %dx%d %g"***

The file's name and dimensions are required. `Width` and `height` are provided in pixels (note the lowercase "x" between them). `Resolution` is the number of pixels per millimeter; its default value is the screen resolution.

Any amount of such canvases may exist simultaneously. Function **`cdCreateCanvas`** creates a memory-based metafile, and a call to function **`cdKillCanvas`** is required to **close** the file properly.

In fact the driver uses a slightly different format, called Aldus Placeable Metafile (APM). It attaches a small header to the beginning of the file, allowing other applications to import better the metafile contents.

It is recomended to use EMFs instead of WMF whenever is possible. This driver is NOT available for the GDI+ base driver.

Behavior of Functions

This driver is greatly platform-dependent. For further detail, see the **Behavior of Functions** of the Microsoft Windows (GDI) platform. However, it should be noted that some functions behave differently from the basic functions of each platform.

Control

- **`cdPlay`**: different from the basic driver, is implemented.
- **`cdClear`**: different from the basic driver, does nothing.

Coordinate System and Clipping

- **`cdClip`**: does nothing, returns `CD_CLIPOFF`.
- **`cdClipArea`**: does nothing
- **`cdGetClipArea`**: does nothing, returns `CD_CLIPOFF`.

Attributes

- **`cdStipple`**: is always opaque and smaller than 8x8 pixels.
- **`cdPattern`**: does nothing.
- **`cdLineWidth`**: is always 1.
- **`cdTextAlignment`**: `CD_CENTER/CD_WEST/CD_EAST` is saved as `CD_BASE_CENTER/CD_BASE_LEFT/CD_BASE_RIGHT`, but the position error is compensated.
- **`cdTextOrientation`**: does nothing

Client Images

- **`cdGetImageRGB`**: does nothing.
- **`cdPutImageRGBA`**: the alpha component is ignored.

Server Images

- All functions do nothing.

Simulation Base Driver

The Simulation driver was created to simulate functions that were not supported by some CD drivers. It works jointly with the other driver (known as "client"), using its pixel, line and text functions to simulate arcs, sectors, polygons, boxes, clipping and fillings with styles.

Important: All simulation primitives are based in the client's Pixel, Image and/or Line functions.

Use

The Simulation driver is used in several parts of the CD library. For example, most drivers do not contain the mark primitive - in this case, the mark's simulation is automatically used.

In many drivers, the behavior of a given primitive may not be the expected. Usually this is documented in the manual. Should you wish to activate the simulation of a primitive, simply call function **cdSimulate** with the code of the primitive to be simulated.

Behavior of Functions

Coordinate System and Clipping

- **Complex Regions**: not supported.

Attributes

- **cdNativeFont**: selects a True Type font file for the FreeType library to render the text. The string format includes the name and size ("`filename, size`" or, in C style, "`%s %d`"). Notice that TTF fonts have different file for different font styles, like bold and italic. Font files can be in the current directory, in the directory pointed by the CDDIR environment variable, in Windows in the system defined Font directory, or using the full path of the file. `size` is given in points or in pixels. The FreeType library was configured to support only TrueType fonts, but it can be configured to support Type 1 fonts also. You may remove the FreeType code from the CD library and use any other distribution.
- **cdFillMode**: only `CD_EVENODD` is supported.
- **cdLineCap**: only `CD_CAPFLAT` is supported.
- **cdLineJoin**: only `CD_MITER` is supported.
- **cdFont**: see the font mapping table for the equivalence used to map CD fonts into TrueType font files:

Font Mapping

CD Fonts	TTF
CD_SYSTEM	cour.ttf
CD_COURIER	cour.ttf
CD_TIMES_ROMAN	times.ttf
CD_HELVETICA	arial.ttf

The file name suffix "bd", "i" and "bi" are used for bold, italic and bold-italic fonts.

Primitives

- **cdPixel**: always uses the client's pixel function. When clipping simulation is active, it executes area and polygon clipping.
- **cdLine**: draws lines pixel per pixel. The line clipping simulation, when possible, divides the line in several portions and draws each portion with the line function, which can be either simulated or not.
- **cdRect**: the simulation is made by means of the line function.
- **cdMark**: the simulation is made by means of the line function or the polygon function.
- **cdArc**: simulates using the client's `cdPixel`. If clipping simulation is active, then the primitive will necessarily be simulated.

- **cdSector**: simulates using the line function or `cdPixel`, depending on the interior style. If clipping simulation is active, then the primitive will necessarily be simulated.
- **cdChord**: Not implemented yet.
- **cdBox**: simulates using the line function or `cdPixel`, depending on the interior style. If clipping simulation is active and it is polygon clipping, then the primitive will necessarily be simulated.
- **cdBegin**, **cdVertex** and **cdEnd**: simulate using the line function or `cdPixel`, depending on the interior style. If clipping simulation is active, then the primitive will necessarily be simulated. `Open` and `Closed` are simulated by calling the line function.
- **cdText**: text simulation is made using TrueType font files in a transparent way for the user. Oriented text is not supported. Clipping simulation is made by eliminating characters which are partially or completely out of the drawing region.

Microsoft Windows Base Driver

This driver represents a base driver for all system-dependent drivers implemented in the Microsoft Windows system. The implementation uses Win32 API graphics functions, the GDI. The driver works better in Windows NT, but it may also work in Windows 9x/Me.

Behavior of Functions

Control

- **cdFlush**: does nothing.
- **cdPlay**: does nothing, returns `CD_ERROR`.

Coordinate System and Clipping

- **cdUpdateYAxis**: the orientation of axis Y is the opposite to its orientation in the CD library.

Primitives

- **cdMark**: is simulated.
- **cdText**: when Write Mode is **XOR** or **NOT_XOR**, the XOR effect is simulated using bitmaps.
- **cdLine**: needs to draw an extra pixel in the final position.

Attributes

- **cdWriteMode**: for the client and server image functions, the mode **NOT_XOR** works as **XOR**.
- **cdStipple**: is always opaque. If not in Windows NT and if `width` or `height` are greater than 8, the stipple is simulated using non-regular Windows clipping regions and bitmaps. The simulation is made when filled boxes, sectors and polygons are drawn.
- **cdPattern**: If not in Windows NT and if `width` or `height` are greater than 8, the pattern is simulated using non-regular Windows clipping regions and bitmaps. The simulation is made when filled boxes, sectors and polygons are drawn.
- **cdTextAlignment**: the vertical alignment of CD_CENTER, CD_EAST, CD_WEST is manually calculated.
- **cdLineWidth**: If not in Windows NT line width is always 1. If line width is 1, then a cosmetic pen is used for fast drawing.
- **cdLineStyle**: If line width is 1, the style is a little different from when line width is not 1, because a cosmetic pen is used for width=1.
- **cdNativeFont**: the font string describer has the following format: "*fontname, size [style] [-k] [-u]*", where `fontname` is the name of the font in Windows (notice the comma after the font name), `size` is given in points or in pixels, `style` is the same as **cdFont**, `-u` means underline and `-k` means strikethrough, or *"-d"* to show the font-selection dialogue box. However, this function also accepts the font string used by the `WINFONT` attribute of the IUP library.
- **cdFont**: see the font mapping table for the equivalence used to map CD fonts into Windows fonts:

Font Mapping

CD Fonts	Windows Fonts
CD_SYSTEM	System
CD_COURIER	Courier New
CD_TIMES_ROMAN	Times New Roman
CD_HELVETICA	Arial

Client Images

- **cdPutImageRGBA**: Try to use the new GDI function AlphaBlend, if not available captures an image from the canvas to blend it manually.

Colors

- **cdPalette**: is useful only if the device has 256 colors. If it has less than 256 colors, ignore this function, for it will not make much difference. If two different canvases have their palettes modified, the last one to be modified will have the best quality; the other one will not have good quality and the colors might have a completely different appearance.

Exclusive Attributes

- "**HDC**": returns the HDC of the Win32 canvas. It can only be retrieved (get only). In Lua is returned as a user data.
- "**SIMXORTEXT**": controls the internal XOR simulation for text. Assumes values "1" (active) and "0" (inactive). Default value: "1". When a text is drawn with write mode XOR it must be simulated. Using this attribute you can disable the simulation, but XOR will not work for cdText.
- "**SIMPATTERN8X8**": controls the internal pattern and stipple simulation. Assumes values "1" (active) and "0" (inactive). Default value: "1". When patterns and stipples are used If not in Windows NT and they have size different of 8x8, then they must be simulated. Using this attribute you can disable the simulation, but patterns and stipple are restricted to 8x8 size in non Windows NT systems.
- "**PENFILLPOLY**": controls the polygon filling outline. Assumes values "1" (active) and "0" (inactive). Default value: "1". When a filled polygon is drawn, a line in the same color is used to draw the border which is not included in the filling. Deactivating this attribute solves the problem of polygons with holes, in which there is a line connecting the external polygon to the internal polygon.
- "**IMAGEFORMAT**": defines the number of bits per pixel used to create server images. It uses 1 integer that can have the values: "32" or "24" (%d). Use NULL to remove the attribute. It is used only in the **cdCreateImage**. When not defined, the server images use the same format of the canvas.
- "**IMAGEALPHA**": allows the usage of an alpha channel for server images if also IMAGEFORMAT=32. The attribute format is a pointer to the transparency values in a sequence of chars in the same format of alpha for client images. The attribute is used only in the **cdCreateImage**, but the pointer must exists while the image exists. The alpha values are transfered to the image only in **cdPutImageRect**, so they can be freely changed any time. It will use the **AlphaBlend** GDI function. The data is not duplicated, only the pointer is stored. The size of the data must be the same size of the image. Use NULL to remove the attribute. Not accessible in Lua. Unsupported in non Windows NT systems.
- "**IMAGEMASK**": defines a binary transparency mask for server images. The format is the same of a stipple, can contain only 0s and 1s. Use 2 integers, width and height, and a char pointer to the mask values inside a string ("%d %d %p"). Use NULL to remove the attribute. It can not be retrieved (set only). Not accessible in Lua. It will use the **MaskBlt** GDI function. Unsupported in non Windows NT systems.
- "**IMAGEPOINTS**": define 3 coordinates of a paralelogram that will be used to warp server images. Use 6 integer values inside a string ("%d %d %d %d %d %d" = x1 y1 x2 y2 x3 y3). Use NULL to remove the attribute. The respective specified points are the upper-left corner, the upper-right corner and the lower left corner. The drawing

is also affected by the "IMAGEMASK" attribute. It will use the **PlgBlt** GDI function. Unsupported in non Windows NT systems.

- "**ROTATE**": allows the usage of 1 angle and 1 coordinate (x, y), that define a global rotation transformation centered in the specified coordinate. Use 1 real and 2 integer values inside a string ("%g %d %d" = angle x y). Unsupported in non Windows NT systems.

Microsoft Windows Base Driver Using GDI+

This driver represents a base driver for all system-dependent drivers implemented in the Microsoft Windows system, but uses a new API called GDI+. The drivers **Clipboard, Native Window**, **IUP**, **Image**, **Printer**, **EMF** and **Double Buffer** were implemented. The driver **WMF**, and the function **cdPlay** of the **Clipboard** and **EMF** drivers were not implemented using GDI+.

The main motivation for the use of GDI+ was transparency for all the primitives. Beyond that we got other features like anti-aliasing, gradient filling, bezier lines, filled cardinal splines and world coordinate directly implemented by the driver.

This driver still does not completely replace the GDI Windows base driver, because GDI+ does not have support for XOR. Also the applications need to adapt the rendering of text that is slightly different from GDI. It is know that GDI+ can be slower than GDI in some cases and faster in other cases, Microsoft does not make this clear.

So we let the programmer to choose what to use. We created the function **cdUseContextPlus** that allows to activate or to deactivate the use of GDI+ for the available drivers. This function affects only the **cdCreateCanvas** function call, once created the canvas will be always a GDI+ canvas. In fact the function affects primary the definitions **CD_IUP**, **CD_NATIVEWINDOW**, etc, because they are function calls and not static defines.

Using GDI+ it is allowed to create more that one canvas at the same time for the same Window. And they can co-exist with a standard GDI canvas.

To enable the use of GDI+ based drivers you must call the initialization function **cdInitGdiPlus()** once and link to the libraries "**cdgdiplus.lib**" and "**gdiplus.lib**". Also the file "**gdiplus.dll**" must be available in your system. These files already came with Visual C++ 7 and Windows XP. For other compilers or systems you will need to copy the ".lib" file for you libraries area, and you will need to copy the DLL for the Windows\System (Win98/Me) or Windows\System32 (Win2000/NT4-SP6) folder.

In CDLUA it is not necessary any additional initialization.

Exclusive Functions

```
int cdUseContextPlus(int use); [in C]
cdUseContextPlus(use: number) -> (old_use: number) [in Lua]
```

Activates or deactivates the use of an external context for the next calls of the **cdCreateCanvas** function. This function is declared in the "**cdgdiplus.h**" header, because now it is useful only for GDI+. But it is implemented in the standard library.

```
void cdInitGdiPlus(void); [in C]
```

Initializes the GDI+ driver to be used as an external context replacing the traditional GDI drivers. This function is declared in the "**cdgdiplus.h**" header.

Behavior of Functions

Control

- **cdPlay**: does nothing, returns CD_ERROR.

Coordinate System and Clipping

- **cdUpdateYAxis**: the orientation of axis Y is the opposite to its orientation in the CD library.

Primitives

- **cdPixel**: uses GDI. Excepting when the canvas is an image so it is done using GDI+.
- **cdMark**: is simulated.
- **cdSector**: it also draws an arc in the same position to complete the size of the sector.
- **cdText**: opaque text is simulated using a rectangle in the back.
- **cdBegin**: Beyond the standard modes it accepts the additional modes: **CD_FILLSPLINE** and **CD_FILLGRADIENT**. The C definitions of these modes are available in the **cdgdiplus.h** header.

  CD_SPLINE defines the points of a curve constructed by a cardinal spline. Uses the current line style.

  CD_FILLSPLINE defines the points of a filled curve constructed by a cardinal spline. Uses the current interior style.

  CD_FILLGRADIENT defines the points of a filled polygon. It is filled with a gradient from colors in each vertex to a color in its center. The colors are defined by the "**GRADIENTCOLOR**" attribute, that must be set before each **cdVertex** call and before **cdEnd** for the center color. This will not affect the current interior style.

Attributes

- **cdBackOpacity**: only changes the transparency of the background color to 0 (transparent) or 255 (opaque).
- **cdHatch**: diagonal styles are drawn with anti-aliasing.
- **cdWriteMode**: does nothing. There is no support for XOR or NOT_XOR.
- **cdPattern**: each pixel can contain transparency information.
- **cdLineStyle**: uses a custom GDI+ style when line width is 1. In World Coordinates the line style has its scaled changed.
- **cdFontDim**: the maximum width is estimated from the character "W".
- **cdNativeFont**: the font string describer has the following format: "*fontname, size [style] [-k] [-u]*", where fontname is the name of the font in Windows (notice the comma after the font name), size is given in points or in pixels, style is the same as **cdFont**, -u means underline and -k means strikethrough, or *"-d"* to show the font-selection dialogue box. However, this function also accepts the font string used by the WINFONT attribute of the IUP library.
- **cdTextAlignment**: is simulated. Although GDI+ has text alignment, the results for the CD were poor.
- **cdFont**: see the font mapping table for the equivalence used to map CD fonts into Windows fonts (note that in GDI+ there is no direct mapping for the system font):

Font Mapping

CD Fonts	Windows Fonts
CD_SYSTEM	GenericSansSerif (usually MS Sans Serif)
CD_COURIER	Courier New
CD_TIMES_ROMAN	Times New Roman
CD_HELVETICA	Arial

Colors

- **cdPalette**: works only when the canvas is a server image.
- **cdForeground** e **cdBackground**: accepts the transparency information encoded in the color.

Client Images

- **cdGetImageRGB**: uses GDI. Excepting when the canvas is an image so it is done using GDI+.

Server Images

- **cdGetImage**: uses GDI. Excepting when the canvas is an image so it is done using GDI+.
- **cdScrollArea**: uses GDI. Excepting when the canvas is an image so it is done using GDI+.

Exclusive Attributes

- "**HDC**": returns the HDC of the Win32 canvas. It can only be retrieved (get only). In Lua is returned as a user data. It is not NULL only in some Native Windows canvas and in the printer canvas.

- "**ANTIALIAS**": controls the use of anti-aliasing by the text, image zoom and line drawing primitives. Assumes values "1" (active) and "0" (inactive). Default value: "1".

- "**GRADIENTCOLOR**": necessary for the creation of the gradient fill defined by a polygon (see details in the function **cdBegin** above). Defines the color of each vertex and the center (%d %d %d" = r g b). It can not be retrieved (set only).

- "**IMAGETRANSP**": defines an interval of colors to be considered transparent in client and server images (except for RGBA images). It uses two colors to define the interval ("%d %d %d %d %d %d" = r1 g1 b1 r2 g3 b3). Use NULL to remove the attribute.

- "**IMAGEFORMAT**": defines the number of bits per pixel used to create server images. It uses 1 integer that can have the values: "32" or "24" (%d). Use NULL to remove the attribute. It is used only in the **cdCreateImage**. When not defined, the server images use the same format of the canvas.

- "**IMAGEALPHA**": allows the usage of an alpha channel for server images if also IMAGEFORMAT=32. The attribute format is a pointer to the transparency values in a sequence of chars in the same format of alpha for client images. The attribute is used only in the **cdCreateImage**, but the pointer must exists while the image exists. The alpha values are transfered to the image only in **cdPutImageRect**, so they can be freely changed any time. The data is not duplicated, only the pointer is stored. The size of the data must be the same size of the image. Use NULL to remove the attribute. Not accessible in Lua.

- "**IMAGEPOINTS**": define 3 coordinates of a paralelogram that will be used to warp server and client images in the subsequent calls of **PutImage** functions. Use 6 integer values inside a string ("%d %d %d %d %d %d" = x1 y1 x2 y2 x3 y3). Use NULL to remove the attribute. The destination rectangle of the **PutImage** functions will be ignored. The respective specified points are the upper-left corner, the upper-right corner and the lower left corner. In GDI+ this attribute is more complete than in GDI, because affects also client images.

- "**ROTATE**": allows the usage of 1 angle and 1 coordinate (x, y), that define a global rotation transformation centered in the specified coordinate. Use 1 real and 2 integer values inside a string ("%g %d %d" = angle x y).

- **"LINEGRADIENT":** defines a filled interior style that uses a line gradient between two colors. It uses 2 points ("%d %d %d %d" = x1 y1 x2 y2), one for the starting point using (using the foreground color), and another one for the end point (using the background color).

- **"LINECAP":** defines addicional line cap styles. It can have the following values: "Triangle", "NoAnchor", "SquareAnchor", "RoundAnchor", "DiamondAnchor", or "ArrowAnchor". It can not be retrieved (set only).

X-Windows Base Driver

This driver represents a basic driver for all system-dependent drivers implemented in the X-Windows system. The implementation uses the XLIB API functions. It was developed using X11R4, but works in more recent versions, such as X11R6.

Note: The coordinates internally implemented by the video driver use 16-bit integers. Therefore, if a coordinate with less than -32k or more than 32k is defined, it will be interpreted incorrectly.

Behavior of Functions

Control

- **cdPlay**: does nothing, returns CD_ERROR.

Coordinate System and Clipping

- **cdUpdateYAxis**: the orientation of axis Y is the opposite to its orientation in the CD library.

Primitives

- **cdBegin**: Filled polygons have an error of one pixel to the right and below. **CD_BEZIER** is simulated with lines.
- **cdMark**: is simulated.
- **cdBox**: in Linux with ATI board, is being drawn with one extra pixel to the right and below.

Attributes

- **cdLineWidth**: if width is 1, the driver will use 0 for a better performance.
- **cdLineStyle**: thick lines have style only in the line's direction. For example, you will see small rectangles in a thick dotted line.
- **cdNativeFont**: uses an X-Windows font string format. You can use program **xfontsel** to select a font and obtain the string.
- **cdFont**: see the font mapping table for the equivalence used to map CD fonts into X-Windows fonts:

Font Mapping

CD Fonts	X-Windows Fonts
CD_SYSTEM	Fixed
CD_COURIER	Courier
CD_TIMES_ROMAN	Times
CD_HELVETICA	Helvetica

Colors

- **cdPalette**: When the number of bits per pixel is smaller than or equal to 8, the driver will use the system palette to solve colors passed as parameters to the canvas. The driver allocates colors as they are requested - if a color cannot be allocated, the closest color is used in the palette. For such, the driver sees all available colors, in the current application and others. If one of the applications is terminated, a color in the palette may become invalid and will only be updated by the driver when it is requested again. For this reason, a call to **cdForeground** or **cdBackground** or **cdPalette** is recommended before drawing.
  When CD_FORCE is used, the driver forces color allocation in the X server. This may imply changing colors in other applications when a cursor moves in and out of the canvas. However, if the number of requested colors is smaller than the maximum number of possible colors in the palette, then the first colors in the default system palette will be preserved, minimizing this problem.
  When CD_POLITE is used, all colors allocated by the driver are liberated, and the requested colors are allocated. This is useful for the application to prioritize the colors that will be allocated, causing other colors to be mapped to their closest colors.
  Note that canvases in the same application interfere with one another, but when a canvas is terminated it liberates all allocated colors.

Client Images

- **cdGetImageRGB**: can be very slow due to the heavy conversions performed to translate data in system format

into RGB vectors.

Exclusive Attributes

- "**GC**": returns the X11 graphics context (get only). In Lua is returned as a user data.