CD
Canvas Draw, A 2D Graphics Library
Version 5.4

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 window or a more abstract surface, such as Image, Clipboard, Metafile, PS, and so on.

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

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. Its drivers are implemented in several platforms, some use portable code, others use native graphics libraries, such as 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.

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.

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@tecgraf.puc-rio.br**. 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 the **CD** support features at **SourceForge**: http://sourceforge.net/projects/canvasdraw/. It provides us Mailing List, CVS Repository and Downloads.

The discussion list is available at: http://lists.sourceforge.net/lists/listinfo/canvasdraw-users.
Source code, pre-compiled binaries and documentation can be downloaded at: http://sourceforge.net/project/showfiles.php?group_id=241317.
The CVS can be browsed at: http://canvasdraw.cvs.sourceforge.net/canvasdraw/.

If you want us to develop a specific feature for the library, Tecgraf is available for partnerships and cooperation. Please contact **tcg@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.

Library Authors:

- Marcelo Gattass
- Luiz Henrique de Figueiredo
- Luiz Fernando Martha
- Antonio Scuri

Thanks to the people that worked and contributed to the library:

- Alexandre Ferreira
- André Derraik
- Camilo Freire
- Carlos Augusto Mendes
- Carlos Cassino
- Carlos Henrique Levy

- Carolina Alfaro
- Danilo Tuler
- Diego Fernandes Nehab
- Erick de Moura Ferreira
- Marcelo Cohen
- Milton Jonathan
- Pedro Miller
- Rafael Rieder
- Renato Borges
- Vinicius da Silva Almendra

We also thank the developers of the FreeType, libJPEG and Mesa libraries, for making the source code available, which helped us improve our implementation of the Simulation driver and of the X-Windows driver. Thanks to Alan Richardson for the XVertex rotines. Thanks to Lode Vandevenne for the LodePNG rotines used in the SVG 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.

Mesa X-Windows utilities source code copyright Brian Paul. libJPEG quantization source code copyright Thomas G. Lane. XVertex rotines source code copyright Alan Richardson.

Thanks for the SourceForge for hosting the support features. Thanks for the LuaForge team for previously hosting the support features for many years.

CD is registered at the National Institute of Intellectual Property in Brazil (INPI) under the number 07571-1, and so it is protected against illegal use. See the Tecgraf Library License for further usage information and Copyright.

Documentation

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

The full documentation can be downloaded from the Download Files. The documentation is also available in Adobe Acrobat and Windows HTML Help formats.

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

Tecgraf Library License

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

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 paperwork, no royalties, no GNU-like "copyleft" restrictions, either. Just download and use it. 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, PUC-Rio keeps 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 and a link to our site in a web page for your product.

The libraries are designed, implemented and maintained 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.

Download

The download site for pre-compiled binaries, documentation and sources is at **SourceForge**:

http://sourceforge.net/projects/canvasdraw/files/

Use this link for the latest version: http://sourceforge.net/projects/canvasdraw/files/5.4.1/

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/

Tecgraf/PUC-Rio Library Download Tips

All the libraries were build using **Tecmake**. Please use it if you intend to recompile the sources. **Tecmake** can be found at http://www.tecgraf.puc-rio.br/tecmake.

The **IM** files can be downloaded at http://sourceforge.net/projects/imtoolkit/files/.
The **CD** files can be downloaded at http://sourceforge.net/projects/canvasdraw/files/.
The **IUP** files can be downloaded at http://sourceforge.net/projects/iup/files/.

The **Lua** files can be downloaded at http://sourceforge.net/projects/luabinaries/files/.

Build Configuration

Libraries and executables were built using speed optimization. In UNIX the dynamic libraries were NOT built with the -fpic parameter. In MacOS X the dynamic libraries are in bundle format. The source code along with the "config.mak" files for **Tecmake** are also available.

The DLLs were built using the **cdecl** calling convention. This should be a problem for Visual Basic users.

In Visual C++ 6 and 7 we use the single thread C Run Time Library for static libraries and the multi thread C RTL for DLLs. Because this were the default in Visual Studio for new projects. Since Visual C++ 8, both use the multithread C RTL.

Packaging

The package files available for download are named according to the platform where they were build.

In UNIX all strings are based in the result of the command "uname -a". The package name is a concatenation of the platform **uname**, the system **major** version number and the system **minor** version number. Some times a suffix must be added to complement the name. The compiler used is always gcc. Binaries for 64-bits receive the suffix: "_64". In Linux when there are different versions of gcc for the same uname, the platform name is created adding the major version number of the compiler added as a suffix: "g3" for gcc 3 and "g4" for gcc 4.

In Windows the platform name is the **compiler** and its **major** version number.

All library packages **(*_lib*)** contains pre-compiled binaries for the specified platform and includes. Packages with **"_bin"** suffix contains executables only.

The package name is a general reference for the platform. If you have the same platform it will work fine, but it may also work in similar platforms.

Here are some examples of packages:

iup2_4_Linux26_lib.tar.gz = IUP 2.4 32-bits Libraries and Includes for Linux with Kernel version 2.6 built with gcc 3.
iup2_4_Linux26g4_64_bin.tar.gz = IUP 2.4 64-bits Executables for Linux with Kernel version 2.6 built with gcc 4.
iup2_4_Win32_vc8_lib.tar.gz = IUP 2.4 32-bits Static Libraries and Includes for Windows to use with Visual C++ 8 (2005).
iup2_4_Win32_dll9_lib.tar.gz = IUP 2.4 32-bits Dynamic Libraries (DLLs), import libraries and Includes for Windows to use with Visual C++ 9 (2008).
iup2_4_Docs_html.tar.gz = IUP 2.4 documentation files in HTML format (the web site files can be browsed locally).
iup2_4_Win32_bin.tar.gz = IUP 2.4 32-bits Executables for Windows.

The documentation files are in HTML format. They do not include the CHM and PDF versions. These two files are provided as a separate download, but they all have the same documentation.

Installation

For any platform we recommend you to create a folder to contain the third party libraries you download. Then just unpack the packages you download in that folder. The packages already contains a directory structure that separates each library or toolkit. For example:

```
\mylibs\
        iup\
            bin\
            html\
            include\
            lib\Linux26
            lib\Linux26g4_64
            lib\vc8
            src
        cd\
        im\
        lua5.1\
```

This structure will also made the process of building from sources more simple, since the projects and makefiles will assume this structure .

Usage

For makefiles use:

```
1) "-I/mylibs/iup/include" to find include files
2) "-L/mylibs/iup/lib/Linux26" to find library files
3) "-liup" to specify the library files
```

For IDEs the configuration involves the same 3 steps above, but each IDE has a different dialog. The IUP toolkit has a Guide for some IDEs:

Borland C++ BuilderX - http://www.tecgraf.puc-rio.br/iup/en/ide_guide/cppbx.html
Code Blocks - http://www.tecgraf.puc-rio.br/iup/en/ide_guide/codeblocks.html
Dev-C++ - http://www.tecgraf.puc-rio.br/iup/en/ide_guide/dev-cpp.html
Eclipse for C++ - http://www.tecgraf.puc-rio.br/iup/en/ide_guide/eclipse.html
Microsoft Visual C++ (Visual Studio 2003) - http://www.tecgraf.puc-rio.br/iup/en/ide_guide/msvc.html
Microsoft Visual C++ (Visual Studio 2005) - http://www.tecgraf.puc-rio.br/iup/en/ide_guide/msvc8.html
Open Watcom - http://www.tecgraf.puc-rio.br/iup/en/ide_guide/owc.html

Available Platforms

The following platforms can be available:

Package Name	Description
AIX43	IBM AIX 4.3 (ppc) / gcc 2.95 / Motif 2.1
IRIX65	SGI IRIX 6.5 (mips) / gcc 3.0 / Motif 2.1
IRIX6465	SGI IRIX 6.5 (mips) / gcc 3.3 / Motif 1.2
Linux24	Red Hat 7.3 (x86) / Kernel 2.4 / gcc 2.95 / Open Motif 2.1 / GTK 2.0
Linux24g3	CentOS 3.9 (x86) / Kernel 2.4 / gcc 3.2 / Open Motif 2.2 [3] / GTK 2.2
Linux24g3_64	Red Hat E.L. WS 3 (x64) / Kernel 2.4 / gcc 3.2 / Open Motif 2.2 [3] / GTK 2.2

Linux26	CentOS 4.6 (x86) / Kernel 2.6 / gcc 3.4 / Open Motif 2.2 [3] / GTK 2.4
Linux26_64	CentOS 4.6 (x64) / Kernel 2.6 / gcc 3.4 / Open Motif 2.2 [3] / GTK 2.4
Linux26g4	CentOS 5.3 (x86) / Kernel 2.6 / gcc 4.1 / Open Motif 2.2 [3] / GTK 2.12
Linux26g4_64	CentOS 5.2 (x64) / Kernel 2.6 / gcc 4.1 / OpenMotif 2.2 [3] / GTK 2.10
Linux26g4ppc	Ubuntu 7.10 (ppc) / Kernel 2.6 / gcc 4.1 / Open Motif 2.2 [3] / GTK 2.?
Linux26_ia64	Red Hat E.L. AS 4 (ia64) / Kernel 2.6 / gcc 3.4 / Open Motif 2.2 [3] / GTK 2.4
SunOS57	Sun Solaris 7 (sparc) / gcc 2.95 / Motif 2.1
SunOS58	Sun Solaris 8 (sparc) / gcc 3.4 / Motif 2.1
SunOS510	Sun Solaris 10 (sparc) / gcc 3.4 / Motif 2.1
SunOS510x86	Sun Solaris 10 (x86) / gcc 3.3 / Motif 2.1
FreeBSD54	Free BSD 5.4 (x86) / gcc 3.4
MacOS104	Mac OS X 10.4 (ppc) [Tiger] / Darwin Kernel 8 / gcc 4.0
MacOS104x86	Mac OS X 10.4 (x86) [Tiger] / Darwin Kernel 8 / gcc 4.0
MacOS105x86	Mac OS X 10.5 (x86) [Leopard] / Darwin Kernel 9 / gcc 4.0
MacOS106	Mac OS X 10.6 (x64) [Snow Leopard] / Darwin Kernel 10 / gcc 4.2
Win32_vc6	Static library built with Microsoft Visual C++ 6 (static RTL/single thread)
Win32_vc7	Static library built with Microsoft Visual C++ 7.1 (.NET 2003) (static RTL/single thread) Also compatible with Microsoft Visual C++ Toolkit 2003
Win32_vc8	Static library built with Microsoft Visual C++ 8.0 (2005) (static RTL/multithread) Also compatible with Microsoft Visual C++ 2005 Express Edition
Win32_vc9	Static library built with Microsoft Visual C++ 9.0 (2008) (static RTL/multithread) Also compatible with Microsoft Visual C++ 2008 Express Edition
Win32_vc10	Static library built with Microsoft Visual C++ 10.0 (2010) (static RTL/multithread) Also compatible with Microsoft Visual C++ 2010 Express Edition - http://www.microsoft.com/express/vc/ Â¹
Win32_dll6	DLL and import library built with vc6, creates dependency with MSVCRT.DLL
Win32_dll7	DLL and import library built with vc7, creates dependency with MSVCR71.DLL
Win32_dll8	DLL and import library built with vc8, creates dependency with MSVCR80.DLL
Win32_dll9	DLL and import library built with vc9, creates dependency with MSVCR90.DLL
Win32_dll10	DLL and import library built with vc10, creates dependency with MSVCR100.DLL
Win64_vc8	Same as **Win32_vc8** but for 64-bits systems using the x64 standard.
Win64_vc9	Same as **Win32_vc9** but for 64-bits systems using the x64 standard.
Win64_vc10	Same as **Win32_vc10** but for 64-bits systems using the x64 standard.
Win64_dll8	Same as **Win32_dll8** but for 64-bits systems using the x64 standard.
Win64_dll9	Same as **Win32_dll9** but for 64-bits systems using the x64 standard.
Win64_dll10	Same as **Win32_dll10** but for 64-bits systems using the x64 standard.
Win32_gcc3	Static library built with Cygwin gcc 3.4 (Depends on Cygwin DLL 1.5)
Win32_gcc4	Static library built with Cygwin gcc 4.3 (Depends on Cygwin DLL 1.7) - http://www.cygwin.com/ Â¹
Win32_cygw15	Same as **Win32_gcc3**, but using the Cygwin Posix system and also with a DLL and import library
Win32_cygw17	Same as **Win32_gcc4**, but using the Cygwin Posix system and also with a DLL and import library
Win32_dllg4	DLL and import library built with Cygwin gcc 4.3 (See **Win32_gcc4**)
Win32_mingw3	Static library built with MingW gcc 3.4
Win32_mingw4	Static library built with MingW gcc 4.3 - http://www.mingw.org/ Â¹ Also compatible with Dev-C++ - http://www.bloodshed.net/devcpp.html and with Code Blocks - http://www.codeblocks.org/ Â¹
Win32_dllw4	DLL and import library built with MingW gcc 4.3 (See **Win32_mingw4**)
Win32_owc1	Static library built with Open Watcom 1.5 - http://www.openwatcom.org/
Win32_bc55	Static library built with Borland C++ 5.5 Compiler - https://downloads.embarcadero.com/free/c_builder Â¹
Win32_bc56	Static library built with Borland C++ BuilderX 1.0 / Borland C++ 5.6 Compiler Â¹,Â² (the C++ BuilderX IDE can also be configured to use mingw3 or gcc3 versions.)
Win32_bc6	Static library built with Embarcadero C++ Builder 2010 / Embarcadero C++ 6 Compiler - https://downloads.embarcadero.com/free/c_builder (trial)
Win32_bin	Executables only for Windows NT/2000/XP/Vista/7 (can be generated by any of the above compilers)
Win64_bin	Same as **Win32_bin** but for 64-bits systems using the x64 standard
Win32_cygw15_bin	Executables only for Windows NT/2000/XP, but using the Cygwin Posix system (See **Win32_cygw15**)
Win32_cygw17_bin	Executables only for Windows NT/2000/XP, but using the Cygwin Posix system (See **Win32_cygw17**)

Â¹ - Notice that all the Windows compilers with links here are free to download and use.
Â² - Recently Borland removed the C++ Builder X from download. But if you bought a book that has the CD of the compiler, then it is still free to use.
[3] - OpenMotif 2.2 is classified as 'experimental' by the Open Group.

CVS

The CVS repository is at **SourceForge**. It can also be interactively browsed at:

http://canvasdraw.cvs.sourceforge.net/canvasdraw/

To checkout use the module name "cd" and the CVSROOT:

```
:pserver:anonymous@canvasdraw.cvs.sourceforge.net:/cvsroot/canvasdraw
```

History of Changes

Version 5.4.1 (09/Nov/2010)

- IMPORTANT: since the previous version CDLua dynamic libraries in Linux are dependent on GDK, because they depend on the main library, that now contains the GDK base drivers. i.e. the X-Win base drivers can NOT be used anymore in dynamic libraries when using Lua in Linux.
- Changed: the Cairo context plus driver in Lua is now the default library when using "cdluacontextplus" in Linux.
- Fixed: CD_GL static library for Visual C++ compilers were incorrectly using FTGL as dynamic library.
- Fixed: CD_QUERY in **cdCanvasNativeFont**.
- Fixed: canvas:Transform when nil is used to reset the transformation.
- Fixed: **cdCanvasClear** not considering the transparency of the background color in the GDI+ base driver and in the CD_GL driver.
- Fixed: background transparency was not being considered when backopacity was set to OPAQUE after the background color was set in the GDI+ base driver.
- Fixed: polygon filling in CD_IMAGERGB driver when there are many horizontal or vertical lines in a sequence on the same polygon.
- Fixed: locale in SVG and PS for floating point numbers, it must use dots "." for decimal separators.
- Fixed: Cairo context plus base driver when used with GDK.
- Fixed: **PutImageRGB**, **PutImageRGBA**, **PutImageMap** and **Pattern** when in 64bits using the Cairo context plus base driver.
- Fixed: resources release in Win32 double buffer driver.

Version 5.4 (24/June/2010)

- New: context plus driver Cairo.
- New: OpenGL driver CD_GL.
- New: "CMD", "OPACITY" and "HATCHBOXSIZE" attributes in the SVG driver.
- New: CD_PATH **cdCanvasBegin** mode to create a path composed of several primitives that can be line draw, filled or used as clipping. New function **cdCanvasPathSet** to configure the action between sequences of **cdCanvasVertex**.
- New: screenshots page in the documentation.
- Changed: CD_DXF now supports solid filled primitives for polygons and rectangles.
- Changed: GDI+ base driver now supports floating point primitives. ATTENTION: check for alignment and size problems in the application. Please report if anything changed.
- Changed: removed compatibility with require"cdlua51", now LuaBinaries must be used or LUA_CPATH must be set.
- Changed: added compatibility with Lua 5.2.
- Changed: IMPORTANT - the main library in Linux and BSD is now the GDK base driver. The X11 base driver library was renamed to **cdx11** on those systems.
- Fixed: PDF driver documentation, **CanvasBackOpacity** and **CanvasBackground** are supported.
- Fixed: SVG driver support for **BackOpacity**=OPAQUE in Hatch. Transform and Clipping conflict. **InteriorStyle** initialization. Pattern orientation was upside down.
- Fixed: alpha transparency in **CanvasArc** and Bezier polygon in the IMAGERGB driver.
- Fixed: missed implementation for CD_DEBUG, CD_PICTURE and CD_DBUFFERRGB in Lua.
- Fixed: Chord in PDF driver.
- Fixed: custom line style sizes in CD_PDF and CD_PS drivers to match behavior of other drivers.
- Fixed: cd.**GetTextBounds** and cd.**wGetTextBounds** to return the correct values and in a table in Lua.

Version 5.3 (26/Jan/2010)

- New: driver SVG.
- New: base driver GDK.
- New: function **CanvasYAxisMode** to control the Y axis orientation.
- New: functions **wdCanvasSetTransform**, **wdCanvasGetTransform**, **wdCanvasTranslate** and **wdCanvasScale**, to better control the WD transformation.
- New: "PDFLIBVERSION", "SUBJECT","TITLE","CREATOR","AUTHOR","KEYWORDS" attributes for the PDF driver.
- New: "FREETYPEVERSION" attribute for the Simulation base driver.
- New: "PRINTERNAME" attribute for the CD_PRINTER base driver.
- Changed: Freetype updated to version 2.3.11.
- Changed: PDFLib Lite updated to version 7.0.4p4.
- Changed: return value to boolean of **CanvasIsPointInRegion** and **cd.UseContextPlus** in Lua.
- Changed: added missing support for alpha channel in image:**cdCreateCanvas**, **cd:imImageCreate**, image**:cdCreateBitmap** and image**:cdInitBitmap** in cdluaim binding.
- Changed: ANTIALIAS attribute is now respected also by **CanvasText** in the IMAGERGB driver.
- Changed: WriteMode=XOR is ignored if alpha transparency is used in the IMAGERGB driver.
- Fixed: indexing of **cdImageRGB**, **cdImageRGBA** and **cdBitmap** objects in Lua.
- Fixed: **CanvasText** for WD when using text with multiple lines.
- Fixed: compositing in the IMAGERGB driver when canvas has a semi-transparent alpha channel and a color with semi transparent alpha are used.
- Fixed: polygon filling in the IMAGERGB driver when the segments contain horizontal lines.
- Fixed: line style background not transparent in the IMAGERGB driver.
- Fixed: **CanvasClear** method was affected by **CanvasWriteMode** in the X11 base driver.
- Fixed: invalid memory access at **CanvasFont** in the PICTURE driver.
- Fixed: improved support for line endings at **cdPlay** in the METAFILE driver.
- Fixed: text alignment in the Windows base driver when WriteMode=XOR.

Version 5.2 (26/Jun/2009)

- New: functions **CanvasGetVectorTextBox, CanvasGetVectorFontSize** and **CanvasVectorFontSize**.
- Changed: the functions **CanvasVectorText**, **CanvasText** and **CanvasGetTextSize** now supports line breaks using '\n'. **CanvasMultiLineVectorText** is now deprecated but kept for compatibility. IMPORTANT - VectorTextSize now uses the full line size.
- Changed: CD_DBUFFER now activates the buffered canvas before creating the image canvas. If size is 0 then uses 1 and avoid to fail during creation.
- Changed: removed "lua5.1.so" dependency in UNIX.
- Fixed: **CanvasText** for CD_IMAGERGB driver when the foreground color has alpha.

Version 5.1.1 (15/Dec/2008)

- Fixed: CDLua binding functions canvas:foreground, canvas:setforeground, canvas:background and canvas:setbackground. Color was being checked from the wrong position in the stack. They were reporting and invalid color when being used.
- Fixed: CDLua binding functions canvas:foreground and canvas:background when using cd.QUERY are value.

Version 5.1 (14/Oct/2008)

- New: CD_DEBUG driver.
- New: the "imlua_cd" library moved from IM to CD under the name "cdluaim". Only the initialization function name is changed.

- New: cdluacontextplus library so the "ContextPlus" base drivers (GDI+ and XRender) can be dinamically loaded using require.
- Changed: **IMPORTANT** - the "cdiup" and "cdluaiup" libraries moved from CD to IUP under the name "iupcd" and "iupluacd". But headers and documentation remains on the CD package. Function names were NOT changed. This change eliminates a cross-dependency that IUP and CD had, now only IUP depends on CD.
- Changed: **IMPORTANT** - renamed "cdgliplus" and "cdxrender" libraries to "cdcontextplus".
- Changed: **IMPORTANT** - removed the FreeType library files from the main library. They now are available as an additional library that can be replaced by other FreeType distributions. You should now link with the "freetype6" library in Windows and with the "freetype" library in UNIX (if not using GTK, freetype is already included in GTK). This change will avoid conflicts when using CD and GTK.
- Changed: **IMPORTANT** - removed the PDFLib library files from the cdpdf library. They now are available as an additional library that can be replaced by other PDFLib distributions. You should now link with the "pdflib" library.
- Changed: **IMPORTANT** - the support services (Downloads, Mailing List and CVS) moved from LuaForge to SourceForge.
- Changed: Makefiles for UNIX now uses a compact version of Tecmake that does not need any installation, just type "make".
- Changed: All dll8 and dll9 DLLs now have a Manifest file that specifies the correct MSVCR*.DLL.
- Changed: improved CDLua parameter checking and error report.
- Changed: improved compatibility for font names in X and Win32.
- Changed: Copyright notice modified to reflect the registration at INPI (National Institute of Intellectual Property in Brazil). License continues under the same terms.
- Changed: improved pattern and stipple resize in **wdCanvasPattern** and **wdCanvasStipple**.
- Changed: pattern creation in Win32 to a more faster method.
- Changed: optimized font search in X-Windows base driver for size variations.
- Changed: the number of bits per pixel returned by the CD_PRINTER driver when the printer is a PDF Writer was 1, now we add a workaround to return 24.
- Changed: improved the color convertion when drawing a RGB image in a CD_PRINTER canvas with 1 bpp (usually a laser printer).
- Changed: (UNDONE from 5.0) in Lua canvases are NOT garbage collected anymore. Since there can be different Lua canvases pointing to the same canvas.
- Changed: font map in simulation driver is not case sensitive anymore.
- Changed: premake files are used now only internally and were removed from the distribution.
- Fixed: added missing CD_NO_OLD_INTERFACE definition on Linux makefiles.
- Fixed: attributes not being preserved after changing clipping or adding a new page in CD_PDF.
- Fixed: polygon clipping in CD_IMAGERGB driver when polygon is larger than the canvas.
- Fixed: **cdCanvasVertex** when adding two reference points with the same coordinates in a bezier.
- Fixed: client image zoom in CD_IMAGERGB driver.
- Fixed: text draw position and gettextsize in Xrender base driver.
- Fixed: double buffer driver invalid memory access when using the Xrender base driver.

Version 5.0 (26/Nov/2007)

- New: attributes "OPACITY", "PATTERN" and "PDF" in the CD_PDF driver.
- New: XRender base driver.
- Changed: PDF Lite library updated to version "7.0.2".
- Changed: FreeType library updated to version "2.3.5".
- Changed: now using "(char*)CD_QUERY" as the parameter in **cdCanvasNativeFont**, it returns the current selected font in the common format definition.
- Changed: avoid setting X-Windows color background when calling **cdCanvasClear** for NativeWindow driver. Now all X-Windows drivers will use only XFillRectangle.
- Changed: in Lua canvases are now garbage collected.
- Changed: metatable names in Lua are now the same as the C struct names.
- Fixed: function cdlua_checkcanvas that affects the creation of the cd.DBUFFER canvas. Thanks to Martin Saerbeck.
- Fixed: vertical text alignment in PDF and PS drivers.
- Fixed: ascent and descent font dimensions in PDF driver.
- Fixed: check for mark size and font size when given size is 0.

Version 5.0 RC2 (09/Apr/2007)

- New: function **cdCanvasInvertYAxis** that will invert the given y coordinate even if the canvas is not internally inverted.
- Changed: PDF Lite library updated to version "7.0.0p3".
- Changed: FreeType library updated to version "2.2.1".
- Changed: In the new API **cdCanvasFont** you can specify partial parameters using NULL, -1 and 0 for typeface, style and size. When these parameters are specified the current font parameter is used. For example, **cdCanvasFont(NULL, -1, 10)** will only change the font size.

Version 5.0 RC1 (08/Mar/2007)

- New: attribute HATCHBOXSIZE in CD_PDF driver, to control the hatch spacing.
- New: attribute ADDFONTMAP in simulation base driver to accept a map betweeen a font name and a font file name.
- New: Pango Font Description string is now accepted in **NativeFont** and replace the previous CD format is most drivers.
  INCOMPATIBILITY - If style is not used, most drivers had a format compatible with the new format. But please check your **NativeFont** usage. The IUP format is still supported.
- New: API using canvas as a parameter. Old API still exists. Library is backward compatible with previous versions, but the documentation shows only the new names. The new functions add a "Canvas" to the function prefix, for ex: the **cdLine** equivalent is **cdCanvasLine**. For these functions **cdActivate** is not required. But **cdCanvasActivate** exists for special cases where the canvas must be updated if an external factor was changed, like a window resize. To facilitate the migration to the new API use the definition CD_NO_OLD_INTERFACE to exclude the old API definitions and check if you are using only the new functions.
- New: support for primitives using "double" floating point precision and not related to WC functions.
- New: "cd_canvas.hpp" header file which defines a C++ class cdCanvasC that wraps the cdCanvas structure API.
- New: ROTATE attribute in CD_PDF driver.
- New: binding Lua of the CD_PDF driver.
- New: support for alpha channel in CD_IMAGERGB driver. Also support for alpha in color coding in the CD_IMAGERGB driver primitives.
- New: attribute ANTIALIAS in the CD_IMAGERGB driver. Text is always antialiased as before.
- New: implemented **Chord** primitive in simulation base driver.
- New: implemented CD_WINDING fill mode in the simulation base driver.
- New: implemented complex clipping regions in CD_IMAGERGB driver. Fixed polygon clipping and other clipping errors in the CD_IMAGERGB driver.
- New: driver CD_DBUFFERRGB that uses the CD_IMAGERGB driver for double buffer, and can be a double buffer for any other driver (CD_DBUFFER works only for Window based drivers).
- New: CD_PICTURE driver to store primitives and attributes in memory that can be played and resized in any other driver.
- New: functions to set color foreground and background without query support (cdCanvasSetForeground and cdCanvasSetBackground). CD_QUERY conflicts with color RGBA=(255,255,255,255) (full transparent white).
- New: support for generic canvas transformations using **Transform**, **TransformTranslate**, **TransformRotate** and **TransformScale** functions.
- New: attribute "GDI+" for all GDI+ based drivers that returns "1". So it can be detected if the driver uses the GDI+ base driver.
- Changed: **INCOMPATIBILITY** - removed clipping simulation from the simulation base driver. It is not possible anymore to simulate clipping, only primitives can be simulated.
- Changed: canvas internal pointer allocation so it can be checked for valid canvas in all external API function calls.
- Changed: **NativeFont**("-d") to set also the foreground color from the color in the dialog, and initialize the font in the dialog with the current selected font.
- Changed: In the new API **cdCanvasFont** changed the typeface parameter type from a small set of integer values to a more flexible string.
- Changed: all accented characters are now available in the default vector text font.
- Changed: all functions in the API now use "const" when applicable.

- Changed: server image defintion from "void*" to "cdImage*". This will affect C++ applications that must update their code.
- Changed: removed **cdGetClipPoly** and **wdGetClipPoly** functions.
- Changed: **UpdateYAxis** now also returns the changed value.
- Changed: **INCOMPATIBILITY - cdCallback** definition used in **RegisterCallback**, called from **Play**. Replaced the "cdContext*" by a "cdCanvas*". If you do not use the pointer it can be simply ignored.
- Changed: WC functions now are only client functions of the CD API.
- Changed: removed old support for Windows 9x.
- Changed: removed the **cdInitGdiPlusIUP** function and the "**cdiupgdiplus**" library. They are not necessary anymore. Althougth the CD_IUP driver still works with GDI+ support.
- Changed: improved speed and precision of the bezier polygon of the simulation base driver.
- Changed: renamed distribution folder name from "cd/data" to "cd/etc".
- Changed: CD Lua for Lua 3 library name changed to include "3" as a suffix.
- Fixed: conversion from ANSI to ASCII in vector text fonts.
- Fixed: Sector primitive in simulation base driver.
- Fixed: deactivation of internal canvas in Double Buffer driver over a Native Windows driver for Win32.
- Fixed: EPS compatibility in PostScript driver.
- Fixed: the default values in **cdCreateCanvas** for CD_DGN, CD_DXF and CD_CGM.
- Fixed: **Play** for CD_EMF when data contains poly-polygons or poly-polylines.
- Fixed: **LineWidth** in WC when updating the size in pixels.
- Fixed: **TextSize** and **FontDim** in driver DXF.
- Fixed: **Font** in the X-Windows base driver, size parameter was incorrectly passed to the X-Windows. **WARNING**: the result font will have a size different than previous CD versions in X-Windows.
- Fixed: **Flush** in CD_DBUFFER driver, it was affected by the write mode state of the buffered canvas.
- Fixed: WC tranformation update when the Window is invalid. Thanks to Marian Trifon.
- Fixed: polygon filling in simulation base driver.
- Fixed: invalid resample in **PutImageRect*** in GDI+ base driver cause a band with a mix of the background color appear on right and bottom when image is zoomed in (larger than original size).

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.
- Fixed: Invalid cdKillImage in X-Windows when active canvas is not the canvas where the image was created.
- Fixed: Text clipping for `CD_IMAGERGB` driver.
- Fixed: fixed size in milimeter of **`cdGetScreenSize`** in Win32.
- Fixed: fixed size of the EMF picture.
- Fixed: fixed the parse of filenames with spaces for all file based drivers. The filename must be inside double quotes (") if it has spaces.
- Fixed: **`cdSetAttribute`** in Lua now can set nil values.
- Fixed: fixed **`cdSector`** when interior style is `CD_HOLLOW`, to include two lines connecting to the center point.
- Fixed: In GDI+, the NATIVEWINDOW driver ignored other data pointer configurations in **`cdCreateCanvas`**.
- Fixed: In GDI+, **`cdStipple`** was not updated when the foreground or background colors where changed.
- Fixed: In GDI+, **`cdSector`** and **`cdArc`** have incorrect angles.
- Fixed: "simple.c" and "simple.zip" were outdated. Now new makefiles were added.
- Fixed: in Windows base driver small incompatibility in cdNativeFont with the IUP FONT attribute.
- Changed: Optimization flags now are ON when building the library in all platforms.
- Changed: 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.
- Changed: Better organization of the documentation.
- Changed: In Windows the NATIVEWINDOW driver now accepts a NULL pointer to draw in the entire screen.
- Changed: Optimized **`cdPutImageRGBARect`** in Windows base driver.
- Changed: Now by default CD will not print X-Windows messages. To enable you must set the CD_XERROR environment variable.
- Changed: The default fill rule for polygons in CD_PS is now the Even-Odd rule. Matching the other drivers.
- Changed: Line Styles in GDI+ was corrected again to match GDI line styles when line width is not 1.
- Changed: The native WC support in GDI+ was removed because of alignment and size problems, simulation will be used.
- Changed: the EMF drivers now ignore the resolution parameter. For EMFs, the resolution is always the screen resolution.
- Changed: the value of following attributes were changed to strings `"IMAGEMASK"`, `"IMAGEPOINTS"`, `"ROTATE"`, `"GRADIENTCOLOR"`, `"IMAGETRANSP"` and `"IMAGEFORMAT"`.
- Changed: 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.
- Changed: **`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.
- Changed: **`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: "USERLINESTYLE" attribute for the base GDI and X11 drivers.

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

Version 4.3.2 (14/Apr/2004)

- Fixed: 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)

- Fixed: 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.
- Changed: the Lua binding is now distributed in the same package. There is only one version number.
- Fixed: the PS header had same flaws, the character ":" was missing in some DCS attributes.
- Fixed: screen resolution was wrong in the Win32 driver, this afects the size of the canvas in milimeters.
- Fixed: in the Win32 driver the creation of a polygon for clipping does not activate the clipping.
- Fixed: in the Win32 driver the function **`cdNativeFont`** using "-d" parameter need some ajusts. Also the returned string does not contains all the used parameters.
- Fixed: in the Win32 driver the function **`cdPutImageRectRGBA`** had a positioning error.

Version 4.2 (20/July/2001)

- Changed: in driver Win32, **`cdNativeFont`** accepts parameter `"-d"` on the font name to show the font-selection dialog.
- Changed: the whole code can now be compiled as C++.
- Changed: functions **`wdPattern`** and **`wdStipple`** were changed to make pattern deformation more uniform.
- Fixed: in the Clipboard driver on Win32, when parameter `"-b"` was used the image was not correctly copied.
- Fixed: 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.
- Fixed: **`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)

- Changed: the driver Native Windows in Win32 now also accepts an already created HDC handle as a parameter.
- Changed: in the **`cdPutImageMap`*** functions, in case the color vector is null, a vector with 256 gray shades in assumed.
- Fixed: **`cdRegisterAttribute`** was not verifying whether the attribute had already been registered.
- Fixed: function **`cdArc`** in the simulation driver (includes **ImageRGB**) was returning without drawing anything in an incorrect test.
- Fixed: function **`cdTextBox`** was returning incorrect values when the text had an orientation different from the default one in some alignment instances.
- Fixed: in function **`cdRGB2Map`** there was a memory invasion.
- Fixed: the vector text simulation was not freeing the memory used for fonts loaded from files.
- Fixed: in the Doubled Buffer driver in X-Windows there was an invalid memory liberation.
- Fixed: 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`**.
- Fixed: when the PS driver generated EPS, it did not correctly add the description of the bounding box (a line break was missing).
- Fixed: 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.
- Fixed: in the X-Win driver, an error in the X-Vertex library caused the texts in 90/270 degrees to be drawn incorrectly.
- Fixed: in the X-Win driver, the **`cdPutImageMap`** functions were generating a memory invasion when the X was in 16 bits.
- Fixed: 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`", "`PENFILLPOLY`"). 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.
- Fixed: 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.
- Fixed: function **`cdActivate`** in Lua was not returning the correct values.
- Fixed: when the image was partially out of the window, above or below, functions **`cdPutImageMap`** and **RGB** were drawing a wrong portion of the image.
- Fixed: 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.
- Fixed: when the active canvas was already `NULL`, the activation of a `NULL` canvas was generating a memory invasion.
- Fixed: 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.
- Fixed: in the IMAGERGB driver, functions **`cdRedImage, cdBlueImage`** and **`cdGreenImage`** were returning invalid pointers.
- Fixed: 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.

- Fixed: the approximate computation of the text size was not accepting sizes in pixels.
- Fixed: the creation of the IMAGERGB driver in Lua was incorrect when the resolution parameter (which is optional) was not specified.
- Fixed: functions **cdGetClipPoly** and **wdGetClipPoly** in Lua were causing memory invasion.
- Changed: 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.
- Changed: in driver X-Windows, the text drawing attribute was implemented with any orientation.
- Changed: 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".
- Changed: function **cdVectorTextTransform** now returns the previsous transformation matrix.

Version 4.0.1 (05/Mar/99)

- Fixed: in the Windows driver, the polygon simulation with pattern was corrected to polygons with repeated points.
- Fixed: in the Windows driver, function **cdNativeFont** was corrected for IUP fonts. It was affecting the Matrix's visualization.
- Fixed: function **cdNativeFont** was wrongly testing its input parameter and always returning.
- Fixed: 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**.
- Fixed: 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.
- Changed: 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.

- Changed: 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.
- Changed: when a canvas which is already active is activated again, an internal driver function is now called, notifying an update instead of an activation.
- Changed: 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.
- Changed: all functions in the control module now have `ASSERT` directives. Thus, using the library with depuration information, one can better detect simple errors.
- Changed: 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).
- Changed: the IMAGERGB driver is now implemented using the simulation functions.
- Changed: the **cdMark** function is back to the function table, so that the drivers in which the primitive can be implemented can profit from it.
- Changed: 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.
- Changed: in the X-Windows driver, the XV code was used to optimize functions **cdPutImageRectRGB** and **cdPutImageRectMap**.
- Changed: the Lua binding was updated. Now the user guide contains Lua function together with C functions.
- Changed: 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.
- Changed: 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.
- Changed: the documentation of the old DOS driver was removed from the user guide.
- Changed: the default resolution for drivers DGN, DXF, METAFILE, CGM and ImageRGB is no longer 1 but 3.8 points per mm (96 DPI).
- Changed: 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.
- Changed: in functions **cdFontDim** and **cdTextSize**, if the driver does not support this kind of query, the values are estimated.
- Changed: function **cdHatch** now returns the previous value.
- Changed: 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.
- Changed: the **cdPlay** function now accepts all window parameters null. In this case, the primitives in the file are interpreted without scaling.
- Changed: **cdFont**now accepts font sizes in pixels when negative values are used as a parameter.

- Changed: 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**.
- Changed: 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.

- Fixed: the documentation of function **cdFont** was confusing, causing errors in the conversion from pixels to points.
- Fixed: function **wdFont** was making a wrong conversion of the font size parameter from millimeters to points.
- Fixed: functions **wdVectorText** and **wdMultiLineVectorText** were generating an extra polygon when the text contained blank spaces in certain positions.
- Fixed: 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.
- Fixed: 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.
- Fixed: 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.
- Fixed: 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.
- Fixed: driver WMF now has text alignment.
- Fixed: in the PRINTER (Windows) driver, function **cdFlush** was not preserving current attributes.
- Fixed: 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)

- Fixed: / 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.
- Changed: the **cdMark** function now uses the **cdPixel** function when drawing a mark of 1 pixel size.
- Changed: / 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.
- Changed: / Win32: the polygon used for clipping is now optimized to not include 3 points that are in the same horizontal or vertical line.
- Fixed: / WD: the **wdVectorText** function was not drawing correctly when extended characters (>128) were used.
- Fixed: / 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.
- Fixed: / X: several resource and memory leaks were corrected.
- Fixed: / IMAGERGB: functions **cdRedImage**, **cdGreenImage** and **cdBlueImage** were not returning the correct pointer.
- Fixed: / 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.
- Changed: / 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.
- Fixed: / X / NativeWindow: the canvas was not created if the screen was 0.
- Fixed: / Win32 / NativeWindow: now the driver considers the existence of non owner-draw device contexts.
- Fixed: / Win32: function **cdClipArea** was not including xmax and xmin in the clipping area.
- Changed: the **cdCallback** typedef was defined, being useful for type casting when calling the **cdRegisterCallback** function.
- Fixed: / Win32: a compatibility problem with the **cdNativeFont** string and the WINFONT IUP attribute was corrected.
- Changed: / Win32: the **cdPutImageRGB** and **cdPutImageMap** functions use a cache memory for best performance.
- Fixed: 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)

- Changed: / X: memory use of the **cdPutImageRGB**, **cdPutImageRGBA** and **cdPutImageMap** functions was optimized, as well as the performance of the **cdPutImageMap** function.
- Changed: / X and Win32: when the canvas has bpp <= 8, function **cdPutImageRGB** converts the image into Map before displaying it.
- Changed: / 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.
- Fixed: / PS: parameters "-w" and "-h" were incorrectly interpreted.
- Fixed: / X: the internal function names were generating an error in the VMS plataform.
- Fixed: / X: the **cdKillCanvas** function was freeing some pointers of the active canvas.
- Changed: / Win32: the **cdVertex** function now ignores duplicate points.
- Changed: / Win32: the **cdNativeFont** function also accepts the font string of the `WINFONT IUP` attribute.
- Fixed: / 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)

- Changed: / X: the **cdFont** function now has a better heuristic to find a closer font if the requested font does not match an available one.
- Changed: / 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.
- Fixed: / X and Win32: the **cdPutImageRect** function was placing the bitmap in a wrong position.
- Fixed: / Win32: the **cdCreateImage** function did not return `NULL` when the creating failed.
- Changed: / Win32: the **cdPutImageRGB**, **cdPutImageRGBA** and **cdPutImageMap** functions were largely optimized when the picture displayed is larger than the screen.
- Changed: / 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.
- Changed: / PS and CGM: now the **cdFontDim** and **cdTextSize** functions return approximate dimensions, instead of nothing.
- Fixed: / PS: the default font was not being set when the canvas was created.
- Fixed: / PS: text alignment was incorrect in the vertical direction.
- Fixed: / SIM: the clipping algorithm of the **cdPixel** function of the Simulation driver was corrected.
- Fixed: / 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

CD

- libEMF in UNIX.

MAC

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

X-WIN

- Xp: X Printer Extension driver
- XShm: Double Buffering and MIT-Shared Memory extensions for server images ?
- XIE: X Imaging Extensions ?
- Shape Extension and XShapeCombineMask to implement "WINDOWRGN" attribute (non rectangular windows from regions)

OpenGL

- Use textures to improve image drawing with transformation and patter+stipple support.
- Use tesselation to support fillmode and non convex polygons.

Simulation

- Implement line styles, line cap and line join for line with > 1.

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 PS Version 3 or *mask images*. Same for **cdPutImageRGBA**.

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

DXF

- 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**.
- Correct function **cdKillCanvas**, which generates "*assertion failed*" when the library is used with debug information and the Seed file is not included.

Screenshots

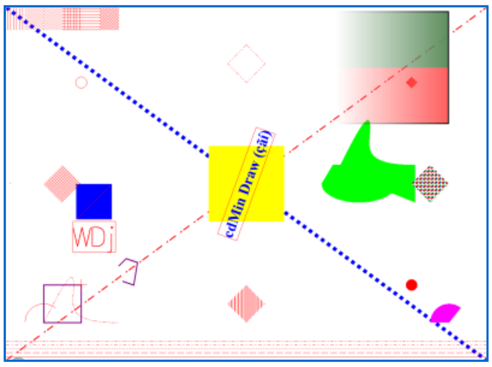
All the screenshots here were generated with the same C source code. The same example is available in Lua source code. The code contains only the primitives and attributes, the canvas initialization is simply:

```
cdCanvas* canvas = cdCreateCanvas(ctx, data);

SimpleDraw(canvas);

/* Destroys the canvas and releases internal memory,
   important for file based drivers to close the file. */
cdKillCanvas(canvas);
```

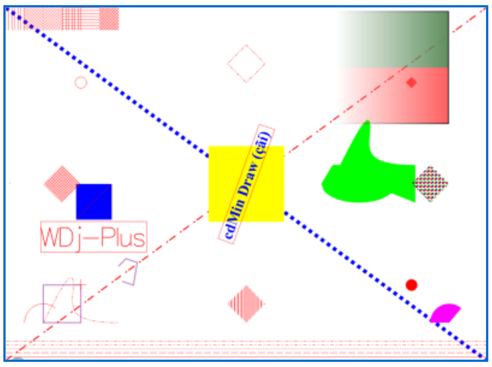
CD_NATIVEWINDOW or CD_IUP

```
Ihandle* cnv = IupCanvas(NULL);
cdInitContextPlus();
...
if (contextplus) cdUseContextPlus(1);
canvas = cdCreateCanvas(CD_IUP, cnv);
if (contextplus) cdUseContextPlus(0);
```

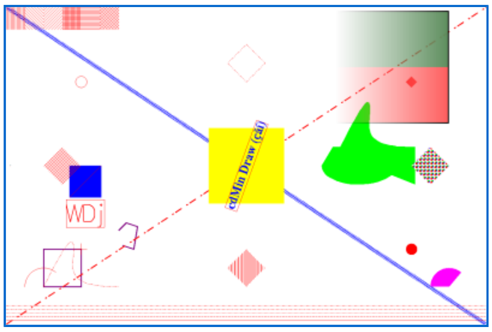
Win32



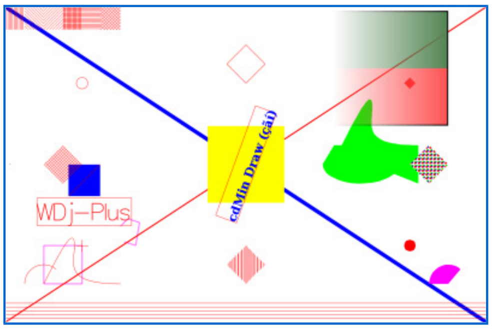
GDI+



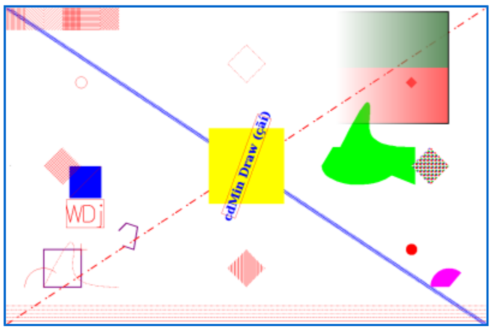
X11



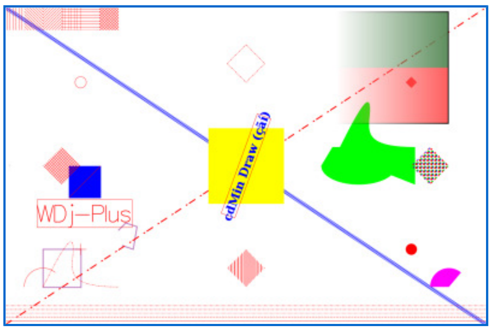
XRender



GDK

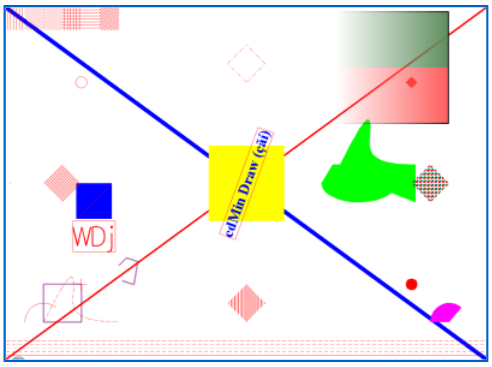


Cairo



CD_IMAGERGB

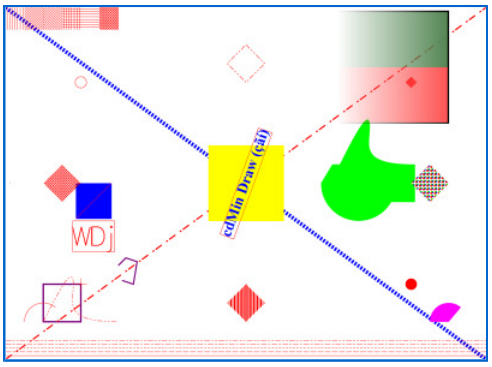
```
canvas = cdCreateCanvas(CD_IMAGERGB, "1280x938");
```



CD_PDF

```
canvas = cdCreateCanvas(CD_PDF, "cd_pdf.pdf -w270.933 -h198.543 -s120");
```

As Seen in Acrobat Reader

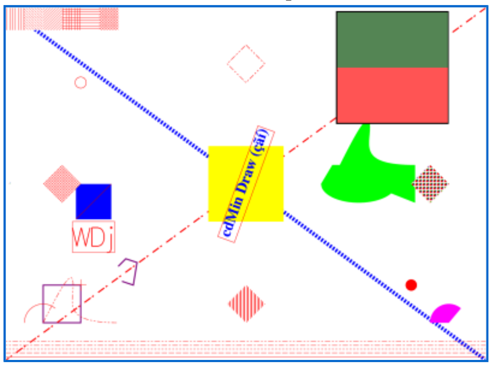


cd_pdf.pdf

CD_PS

```
canvas = cdCreateCanvas(CD_PS, "cd_ps.ps -l0 -r0 -t0 -b0 -w270.933 -h198.543 -s120");
canvas = cdCreateCanvas(CD_PS, "cd_ps.eps -e -w270.933 -h198.543 -s120");
```

As Seen in Ghostscript/GSView

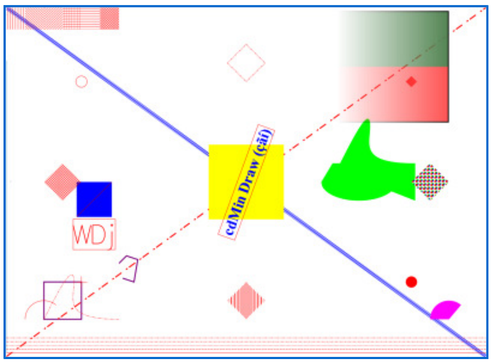


cd_ps.ps
cd_ps.eps

CD_SVG

```
canvas = cdCreateCanvas(CD_SVG, "cd_svg.svg 270.933x198.543 4.72441");
```

As Seen in Firefox



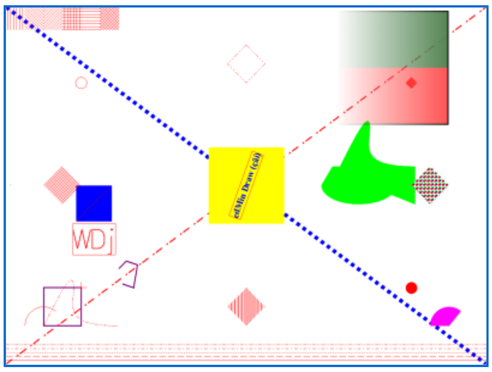
cd_svg.svg

CD_EMF

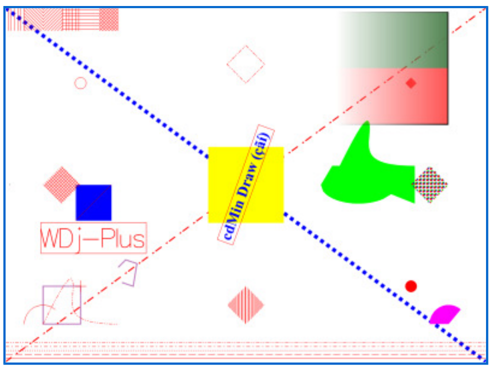
```
canvas = cdCreateCanvas(CD_EMF, "cd_emf.emf 1280x938");
```

As Seen in Powerpoint

Win32 **GDI+**



cd_emf_win32.emf



cd_emf_gdiplus.emf

Other Metafiles

CD_METAFILE - cd_metafile.mf

```
canvas = cdCreateCanvas(CD_METAFILE, "cd_wmf.mf 270.933x198.543 4.72441");
```

CD_DEBUG - cd_debug.txt

```
canvas = cdCreateCanvas(CD_DEBUG, "cd_debug.txt 270.933x198.543 4.72441");
```

CD_WMF - cd_wmf.wmf

```
canvas = cdCreateCanvas(CD_WMF, "cd_wmf.wmf 1280x938");
```

CD_CGM - cd_cgm.cgm

```
canvas = cdCreateCanvas(CD_CGM, "cd_cgm.cgm 270.933x198.543 4.72441");
```

CD_CGM - cd_cgm_t.cgm (text mode)

```
canvas = cdCreateCanvas(CD_CGM, "cd_cgm_t.cgm -t 270.933x198.543 4.72441");
```

CD_DGN - cd_dgn.dgn

```
canvas = cdCreateCanvas(CD_DGN, "cd_dgn.dgn 270.933x198.543 4.72441");
```

CD_DXF - cd_dxf.dxf

```
canvas = cdCreateCanvas(CD_DXF, "cd_dxf.dxf 270.933x198.543 4.72441");
```

Screenshots C Source Code

Get the source code here: simple.zip.

```
void SimpleDraw(cdCanvas* canvas)
{
  int w, h;
  int *dashes;
  int irect[8];
  double drect[8];

  /* Get size in pixels to be used for computing coordinates. */
  cdCanvasGetSize(canvas, &w, &h, NULL, NULL);
  /* Clear the background to be white */
  cdCanvasBackground(canvas, CD_WHITE);
  cdCanvasClear(canvas);
```

```
/* Draw a reactangle and a polyline at the bottom-left area,
   using a thick line with transparency.
   Notice that transparency is only supported in a few drivers,
   and line join is not supported in the IMAGERGB driver. */
cdCanvasLineWidth(canvas, 3);
cdCanvasLineStyle(canvas, CD_CONTINUOUS);
cdCanvasForeground(canvas, cdEncodeAlpha(CD_DARK_MAGENTA, 128));
cdCanvasRect(canvas, 100, 200, 100, 200);
cdCanvasBegin(canvas, CD_OPEN_LINES);
cdCanvasVertex(canvas, 300, 250);
cdCanvasVertex(canvas, 320, 270);
cdCanvasVertex(canvas, 350, 260);
cdCanvasVertex(canvas, 340, 200);
cdCanvasVertex(canvas, 310, 210);
cdCanvasEnd(canvas);
/* Draw the red diagonal line with a custom line style.
   Notice that line styles are not supported in the IMAGERGB driver. */
cdCanvasForeground(canvas, CD_RED);
cdCanvasLineWidth(canvas, 3);
dashes = {20, 15, 5, 5};
cdCanvasLineStyleDashes(canvas, dashes, 4);
cdCanvasLineStyle(canvas, CD_CUSTOM);
cdCanvasLine(canvas, 0, 0, w-1, h-1);

/* Draw the blue diagonal line with a pre-defined line style.
   Notice that the pre-defined line style is dependent on the driver. */
cdCanvasForeground(canvas, CD_BLUE);
cdCanvasLineWidth(canvas, 10);
cdCanvasLineStyle(canvas, CD_DOTTED);
cdCanvasLine(canvas, 0, h-1, w-1, 0);

/* Reset line style and width */
cdCanvasLineStyle(canvas, CD_CONTINUOUS);
cdCanvasLineWidth(canvas, 1);
/* Draw an arc at bottom-left, and a sector at bottom-right.
   Notice that counter-clockwise orientation of both. */
cdCanvasInteriorStyle(canvas, CD_SOLID);
cdCanvasForeground(canvas, CD_MAGENTA);
cdCanvasSector(canvas, w-100, 100, 100, 100, 50, 180);
cdCanvasForeground(canvas, CD_RED);
cdCanvasArc(canvas, 100, 100, 100, 100, 50, 180);

/* Draw a solid filled rectangle at center. */
cdCanvasForeground(canvas, CD_YELLOW);
cdCanvasBox(canvas, w/2 - 100, w/2 + 100, h/2 - 100, h/2 + 100);
```

```
/* Prepare font for text. */
cdCanvasTextAlignment(canvas, CD_CENTER);
cdCanvasTextOrientation(canvas, 70);
cdCanvasFont(canvas, "Times", CD_BOLD, 24);

/* Draw text at center, with orientation,
   and draw its bounding box.
   Notice that in some drivers the bounding box is not precise. */
cdCanvasGetTextBounds(canvas, w/2, h/2, "cdMin Draw (çãí)", irect);
cdCanvasForeground(canvas, CD_RED);
cdCanvasBegin(canvas, CD_CLOSED_LINES);
cdCanvasVertex(canvas, irect[0], irect[1]);
cdCanvasVertex(canvas, irect[2], irect[3]);
cdCanvasVertex(canvas, irect[4], irect[5]);
cdCanvasVertex(canvas, irect[6], irect[7]);
cdCanvasEnd(canvas);
cdCanvasForeground(canvas, CD_BLUE);
cdCanvasText(canvas, w/2, h/2, "cdMin Draw (çãí)");

/* Prepare World Coordinates */
wdCanvasViewport(canvas, 0,w-1,0,h-1);
if (w>h)
    wdCanvasWindow(canvas, 0,(double)w/(double)h,0,1);
else
    wdCanvasWindow(canvas, 0,1,0,(double)h/(double)w);

/* Draw a filled blue rectangle in WC */
wdCanvasBox(canvas, 0.20, 0.30, 0.40, 0.50);
cdCanvasForeground(canvas, CD_RED);

/* Draw the diagonal of that rectangle in WC */
wdCanvasLine(canvas, 0.20, 0.40, 0.30, 0.50);

/* Prepare Vector Text in WC. */
wdCanvasVectorCharSize(canvas, 0.07);

/* Draw vector text, and draw its bounding box.
   We also use this text to show when we are using a contextplus driver. */
cdCanvasForeground(canvas, CD_RED);
if (contextplus)
    wdCanvasGetVectorTextBounds(canvas, "WDj-Plus", 0.25, 0.35, drect);
else
    wdCanvasGetVectorTextBounds(canvas, "WDj", 0.25, 0.35, drect);
cdCanvasBegin(canvas, CD_CLOSED_LINES);
```

```
wdCanvasVertex(canvas, drect[0], drect[1]);
wdCanvasVertex(canvas, drect[2], drect[3]);
wdCanvasVertex(canvas, drect[4], drect[5]);
wdCanvasVertex(canvas, drect[6], drect[7]);
cdCanvasEnd(canvas);
cdCanvasLineWidth(canvas, 2);
cdCanvasLineStyle(canvas, CD_CONTINUOUS);
if (contextplus)
  wdCanvasVectorText(canvas, 0.25, 0.35, "WDj-Plus");
else
  wdCanvasVectorText(canvas, 0.25, 0.35, "WDj");

/* Reset line width */
cdCanvasLineWidth(canvas, 1);

/* Draw a filled path at center-right (looks like a weird fish).
   Notice that in PDF the arc is necessarily a circle arc, and not an ellipse. */
cdCanvasForeground(canvas, CD_GREEN);
cdCanvasBegin(canvas, CD_PATH);
cdCanvasPathSet(canvas, CD_PATH_MOVETO);
cdCanvasVertex(canvas, w/2 + 200, h/2);
cdCanvasPathSet(canvas, CD_PATH_LINETO);
cdCanvasVertex(canvas, w/2 + 230, h/2 + 50);
cdCanvasPathSet(canvas, CD_PATH_LINETO);
cdCanvasVertex(canvas, w/2 + 250, h/2 + 50);
cdCanvasPathSet(canvas, CD_PATH_CURVETO);
cdCanvasVertex(canvas, w/2+150+150, h/2+200-50); /* control point for start */
cdCanvasVertex(canvas, w/2+150+180, h/2+250-50); /* control point for end */
cdCanvasVertex(canvas, w/2+150+180, h/2+200-50); /* end point */
cdCanvasPathSet(canvas, CD_PATH_CURVETO);
cdCanvasVertex(canvas, w/2+150+180, h/2+150-50);
cdCanvasVertex(canvas, w/2+150+150, h/2+100-50);
cdCanvasVertex(canvas, w/2+150+300, h/2+100-50);
cdCanvasPathSet(canvas, CD_PATH_LINETO);
cdCanvasVertex(canvas, w/2+150+300, h/2-50);
cdCanvasPathSet(canvas, CD_PATH_ARC);
cdCanvasVertex(canvas, w/2+300, h/2); /* center */
cdCanvasVertex(canvas, 200, 100); /* width, height */
cdCanvasVertex(canvas, -30*1000, -170*1000); /* start angle, end angle (degrees / 1000) */
cdCanvasPathSet(canvas, CD_PATH_FILL);
cdCanvasEnd(canvas);

/* Draw 3 pixels at center left. */
cdCanvasPixel(canvas, 10, h/2+0, CD_RED);
cdCanvasPixel(canvas, 11, h/2+1, CD_GREEN);
cdCanvasPixel(canvas, 12, h/2+2, CD_BLUE);

/* Draw 4 mark types, distributed near each corner. */
cdCanvasForeground(canvas, CD_RED);
cdCanvasMarkSize(canvas, 30);
cdCanvasMarkType(canvas, CD_PLUS);
cdCanvasMark(canvas, 200, 200);
cdCanvasMarkType(canvas, CD_CIRCLE);
cdCanvasMark(canvas, w - 200, 200);
cdCanvasMarkType(canvas, CD_HOLLOW_CIRCLE);
cdCanvasMark(canvas, 200, h - 200);
cdCanvasMarkType(canvas, CD_DIAMOND);
cdCanvasMark(canvas, w - 200, h - 200);

/* Draw all the line style possibilities at bottom.
   Notice that they have some small differences between drivers. */
cdCanvasLineWidth(canvas, 1);
cdCanvasLineStyle(canvas, CD_CONTINUOUS);
cdCanvasLine(canvas, 0, 10, w, 10);
cdCanvasLineStyle(canvas, CD_DASHED);
cdCanvasLine(canvas, 0, 20, w, 20);
cdCanvasLineStyle(canvas, CD_DOTTED);
cdCanvasLine(canvas, 0, 30, w, 30);
cdCanvasLineStyle(canvas, CD_DASH_DOT);
cdCanvasLine(canvas, 0, 40, w, 40);
cdCanvasLineStyle(canvas, CD_DASH_DOT_DOT);
cdCanvasLine(canvas, 0, 50, w, 50);

/* Draw all the hatch style possibilities in the top-left corner.
   Notice that they have some small differences between drivers. */
cdCanvasHatch(canvas, CD_VERTICAL);
cdCanvasBox(canvas, 0, 50, h - 60, h);
cdCanvasHatch(canvas, CD_FDIAGONAL);
cdCanvasBox(canvas, 50, 100, h - 60, h);
cdCanvasHatch(canvas, CD_BDIAGONAL);
cdCanvasBox(canvas, 100, 150, h - 60, h);
cdCanvasHatch(canvas, CD_CROSS);
cdCanvasBox(canvas, 150, 200, h - 60, h);
cdCanvasHatch(canvas, CD_HORIZONTAL);
cdCanvasBox(canvas, 200, 250, h - 60, h);
cdCanvasHatch(canvas, CD_DIAGCROSS);
cdCanvasBox(canvas, 250, 300, h - 60, h);

/* Draw 4 regions, in diamond shape,
   at top, bottom, left, right,
  using different interior styles. */

/* At top, not filled polygon, notice that the last line style is used. */
cdCanvasBegin(canvas, CD_CLOSED_LINES);
cdCanvasVertex(canvas, w/2, h - 100);
cdCanvasVertex(canvas, w/2 + 50, h - 150);
cdCanvasVertex(canvas, w/2, h - 200);
cdCanvasVertex(canvas, w/2 - 50, h - 150);
cdCanvasEnd(canvas);

/* At left, hatch filled polygon */
cdCanvasHatch(canvas, CD_DIAGCROSS);
cdCanvasBegin(canvas, CD_FILL);
cdCanvasVertex(canvas, 100, h/2);
cdCanvasVertex(canvas, 150, h/2 + 50);
cdCanvasVertex(canvas, 200, h/2);
cdCanvasVertex(canvas, 150, h/2 - 50);
cdCanvasEnd(canvas);

/* At right, pattern filled polygon */
cdCanvasPattern(canvas, STYLE_SIZE, STYLE_SIZE, pattern);
cdCanvasBegin(canvas, CD_FILL);
cdCanvasVertex(canvas, w - 100, h/2);
cdCanvasVertex(canvas, w - 150, h/2 + 50);
cdCanvasVertex(canvas, w - 200, h/2);
cdCanvasVertex(canvas, w - 150, h/2 - 50);
cdCanvasEnd(canvas);
/* At bottom, stipple filled polygon */
cdCanvasStipple(canvas, STYLE_SIZE, STYLE_SIZE, stipple);
cdCanvasBegin(canvas, CD_FILL);
cdCanvasVertex(canvas, w/2, 100);
cdCanvasVertex(canvas, w/2 + 50, 150);
cdCanvasVertex(canvas, w/2, 200);
cdCanvasVertex(canvas, w/2 - 50, 150);
cdCanvasEnd(canvas);

/* Draw two beziers at bottom-left */
cdCanvasBegin(canvas, CD_BEZIER);
cdCanvasVertex(canvas, 100, 100);
cdCanvasVertex(canvas, 150, 200);
cdCanvasVertex(canvas, 180, 250);
cdCanvasVertex(canvas, 180, 200);
cdCanvasVertex(canvas, 180, 150);
cdCanvasVertex(canvas, 150, 100);
cdCanvasVertex(canvas, 300, 100);
```

```
    cdCanvasEnd(canvas);

    /* Draw the image on the top-right corner but increasing its actual size,
       and uses the complete image */
    cdCanvasPutImageRectRGBA(canvas, IMAGE_SIZE, IMAGE_SIZE, red, green, blue, alpha,
                            w - 400, h - 310, 3*IMAGE_SIZE, 3*IMAGE_SIZE, 0, 0, 0, 0);

    /* Adds a new page, or
       flushes the file, or
       flushes the screen, or
       swap the double buffer. */
    cdCanvasFlush(canvas);
}
```

Screenshots Lua Source Code

Get the source code here: simpledraw.lua.

```
function SimpleDraw()
    -- Get size in pixels to be used for computing coordinates.
    w, h = canvas:GetSize()
    -- Clear the background to be white
    canvas:Background(cd.WHITE)
    canvas:Clear()

    -- Draw a reactangle and a polyline at the bottom-left area,
    -- using a thick line with transparency.
    -- Notice that transparency is only supported in a few drivers,
    -- and line join is not supported in the IMAGERGB driver.
    canvas:LineWidth(3)
    canvas:LineStyle(cd.CONTINUOUS)
    canvas:Foreground(cd.EncodeAlpha(cd.DARK_MAGENTA, 128))
    canvas:Rect(100, 200, 100, 200)
    canvas:Begin(cd.OPEN_LINES)
    canvas:Vertex(300, 250)
    canvas:Vertex(320, 270)
    canvas:Vertex(350, 260)
    canvas:Vertex(340, 200)
    canvas:Vertex(310, 210)
    canvas:End()
    -- Draw the red diagonal line with a custom line style.
    -- Notice that line styles are not supported in the IMAGERGB driver.
    canvas:Foreground(cd.RED)
    canvas:LineWidth(3)
    dashes = {20, 15, 5, 5}
    canvas:LineStyleDashes(dashes, 4)
    canvas:LineStyle(cd.CUSTOM)
```

```
canvas:Line(0, 0, w-1, h-1)

-- Draw the blue diagonal line with a pre-defined line style.
-- Notice that the pre-defined line style is dependent on the driver.
canvas:Foreground(cd.BLUE)
canvas:LineWidth(10)
canvas:LineStyle(cd.DOTTED)
canvas:Line(0, h-1, w-1, 0)

-- Reset line style and width
canvas:LineStyle(cd.CONTINUOUS)
canvas:LineWidth(1)
-- Draw an arc at bottom-left, and a sector at bottom-right.
-- Notice that counter-clockwise orientation of both.
canvas:InteriorStyle(cd.SOLID)
canvas:Foreground(cd.MAGENTA)
canvas:Sector(w-100, 100, 100, 100, 50, 180)
canvas:Foreground(cd.RED)
canvas:Arc(100, 100, 100, 100, 50, 180)

-- Draw a solid filled rectangle at center.
canvas:Foreground(cd.YELLOW)
canvas:Box(w/2 - 100, w/2 + 100, h/2 - 100, h/2 + 100)

-- Prepare font for text.
canvas:TextAlignment(cd.CENTER)
canvas:TextOrientation(70)
canvas:Font("Times", cd.BOLD, 24)

-- Draw text at center, with orientation,
-- and draw its bounding box.
-- Notice that in some drivers the bounding box is not precise.
irect = canvas:GetTextBounds(w/2, h/2, "cdMin Draw (çãí)")
canvas:Foreground(cd.RED)
canvas:Begin(cd.CLOSED_LINES)
canvas:Vertex(irect[1], irect[2])
canvas:Vertex(irect[3], irect[4])
canvas:Vertex(irect[5], irect[6])
canvas:Vertex(irect[7], irect[8])
canvas:End()
canvas:Foreground(cd.BLUE)
canvas:Text(w/2, h/2, "cdMin Draw (çãí)")

-- Prepare World Coordinates
canvas:wViewport(0,w-1,0,h-1)
```

```
if (w>h) then
   canvas:wWindow(0,w/h,0,1)
else
   canvas:wWindow(0,1,0,h/w)
end

-- Draw a filled blue rectangle in WC
canvas:wBox(0.20, 0.30, 0.40, 0.50)
canvas:Foreground(cd.RED)

-- Draw the diagonal of that rectangle in WC
canvas:wLine(0.20, 0.40, 0.30, 0.50)

-- Prepare Vector Text in WC.
canvas:wVectorCharSize(0.07)

-- Draw vector text, and draw its bounding box.
-- We also use this text to show when we are using a contextplus driver.
canvas:Foreground(cd.RED)
if (contextplus) then
   drect = canvas:wGetVectorTextBounds("WDj-Plus", 0.25, 0.35)
else
   drect = canvas:wGetVectorTextBounds("WDj", 0.25, 0.35)
end
canvas:Begin(cd.CLOSED_LINES)
canvas:wVertex(drect[1], drect[2])
canvas:wVertex(drect[3], drect[4])
canvas:wVertex(drect[5], drect[6])
canvas:wVertex(drect[7], drect[8])
canvas:End()
canvas:LineWidth(2)
canvas:LineStyle(cd.CONTINUOUS)
if (contextplus) then
   canvas:wVectorText(0.25, 0.35, "WDj-Plus")
else
   canvas:wVectorText(0.25, 0.35, "WDj")
end

-- Reset line width
canvas:LineWidth(1)

-- Draw a filled path at center-right (looks like a weird fish).
-- Notice that in PDF the arc is necessarily a circle arc, and not an ellipse.
canvas:Foreground(cd.GREEN)
canvas:Begin(cd.PATH)
```

```
canvas:PathSet(cd.PATH_MOVETO)
canvas:Vertex(w/2 + 200, h/2)
canvas:PathSet(cd.PATH_LINETO)
canvas:Vertex(w/2 + 230, h/2 + 50)
canvas:PathSet(cd.PATH_LINETO)
canvas:Vertex(w/2 + 250, h/2 + 50)
canvas:PathSet(cd.PATH_CURVETO)
canvas:Vertex(w/2+150+150, h/2+200-50) -- control point for start
canvas:Vertex(w/2+150+180, h/2+250-50) -- control point for end
canvas:Vertex(w/2+150+180, h/2+200-50) -- end point
canvas:PathSet(cd.PATH_CURVETO)
canvas:Vertex(w/2+150+180, h/2+150-50)
canvas:Vertex(w/2+150+150, h/2+100-50)
canvas:Vertex(w/2+150+300, h/2+100-50)
canvas:PathSet(cd.PATH_LINETO)
canvas:Vertex(w/2+150+300, h/2-50)
canvas:PathSet(cd.PATH_ARC)
canvas:Vertex(w/2+300, h/2) -- center
canvas:Vertex(200, 100) -- width, height
canvas:Vertex(-30*1000, -170*1000) -- start angle, end angle (degrees / 1000)
canvas:PathSet(cd.PATH_FILL)
canvas:End()

-- Draw 3 pixels at center left.
canvas:Pixel(10, h/2+0, cd.RED)
canvas:Pixel(11, h/2+1, cd.GREEN)
canvas:Pixel(12, h/2+2, cd.BLUE)

-- Draw 4 mark types, distributed near each corner.
canvas:Foreground(cd.RED)
canvas:MarkSize(30)
canvas:MarkType(cd.PLUS)
canvas:Mark(200, 200)
canvas:MarkType(cd.CIRCLE)
canvas:Mark(w - 200, 200)
canvas:MarkType(cd.HOLLOW_CIRCLE)
canvas:Mark(200, h - 200)
canvas:MarkType(cd.DIAMOND)
canvas:Mark(w - 200, h - 200)

-- Draw all the line style possibilities at bottom.
-- Notice that they have some small differences between drivers.
canvas:LineWidth(1)
canvas:LineStyle(cd.CONTINUOUS)
canvas:Line(0, 10, w, 10)
```

```
canvas:LineStyle(cd.DASHED)

canvas:Line(0, 20, w, 20)

canvas:LineStyle(cd.DOTTED)

canvas:Line(0, 30, w, 30)

canvas:LineStyle(cd.DASH_DOT)

canvas:Line(0, 40, w, 40)

canvas:LineStyle(cd.DASH_DOT_DOT)

canvas:Line(0, 50, w, 50)


-- Draw all the hatch style possibilities in the top-left corner.

-- Notice that they have some small differences between drivers.

canvas:Hatch(cd.VERTICAL)

canvas:Box(0, 50, h - 60, h)

canvas:Hatch(cd.FDIAGONAL)

canvas:Box(50, 100, h - 60, h)

canvas:Hatch(cd.BDIAGONAL)

canvas:Box(100, 150, h - 60, h)

canvas:Hatch(cd.CROSS)

canvas:Box(150, 200, h - 60, h)

canvas:Hatch(cd.HORIZONTAL)

canvas:Box(200, 250, h - 60, h)

canvas:Hatch(cd.DIAGCROSS)

canvas:Box(250, 300, h - 60, h)


-- Draw 4 regions, in diamond shape,

-- at top, bottom, left, right,

-- using different interior styles.


-- At top, not filled polygon, notice that the last line style is used.

canvas:Begin(cd.CLOSED_LINES)

canvas:Vertex(w/2, h - 100)

canvas:Vertex(w/2 + 50, h - 150)

canvas:Vertex(w/2, h - 200)

canvas:Vertex(w/2 - 50, h - 150)

canvas:End()


-- At left, hatch filled polygon

canvas:Hatch(cd.DIAGCROSS)

canvas:Begin(cd.FILL)

canvas:Vertex(100, h/2)

canvas:Vertex(150, h/2 + 50)

canvas:Vertex(200, h/2)

canvas:Vertex(150, h/2 - 50)

canvas:End()
```

```
    -- At right, pattern filled polygon

    canvas:Pattern(pattern)

    canvas:Begin(cd.FILL)

    canvas:Vertex(w - 100, h/2)

    canvas:Vertex(w - 150, h/2 + 50)

    canvas:Vertex(w - 200, h/2)

    canvas:Vertex(w - 150, h/2 - 50)

    canvas:End()

    -- At bottom, stipple filled polygon

    canvas:Stipple(stipple)

    canvas:Begin(cd.FILL)

    canvas:Vertex(w/2, 100)

    canvas:Vertex(w/2 + 50, 150)

    canvas:Vertex(w/2, 200)

    canvas:Vertex(w/2 - 50, 150)

    canvas:End()


    -- Draw two beziers at bottom-left

    canvas:Begin(cd.BEZIER)

    canvas:Vertex(100, 100)

    canvas:Vertex(150, 200)

    canvas:Vertex(180, 250)

    canvas:Vertex(180, 200)

    canvas:Vertex(180, 150)

    canvas:Vertex(150, 100)

    canvas:Vertex(300, 100)

    canvas:End()


    -- Draw the image on the top-right corner but increasing its actual size,

    -- and uses the complete image

    canvas:PutImageRectRGBA(imagergba, w - 400, h - 310, 3*IMAGE_SIZE, 3*IMAGE_SIZE, 0, 0, 0, 0)


    -- Adds a new page, or

    -- flushes the file, or

    -- flushes the screen, or

    -- swap the double buffer.

    canvas:Flush()

end
```

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:

- **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.

- **GGI** - 2D graphics library aimed only at display. http://www.ggi-project.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, usually render in bitmaps on native systems. Display hardware acceleration is used almost only to display the bitmaps. Although it can reach high quality rendering. http://cairographics.org/.
- **AGG** - The AGG Project (Anti-Grain Geometry). High Fidelity 2D Graphics A High Quality Rendering Engine for C++. Renders to a bitmap then transfer it to the native system, just like Cairo. GNU GPL license. http://www.antigrain.com/

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, like Qt and wxWidgets.

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** and **freetype6.lib/libfreetype.a/libfreetype.so** libraries to the linker options. If you use an IUP Canvas then you must also link with the **iupcd.lib/libiupcd.a/libiupcd.so** library available in the IUP distribution.

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

In the Downloads you will be able to find pre-compiled binaries for many platforms, all those binaries were built using Tecmake. Tecmake is a command line multi compiler build tool based on GNU make, available at http://www.tecgraf.puc-rio.br/tecmake. Tecmake is used by all the Tecgraf libraries and many applications.

In UNIX, you do not need to install Tecmake, a version for Posix systems is already included in the source code package. Just type "make" in the command line on the "src" folder and all libraries will be build. Set the TECTOOLS_HOME environment variable to the folder were the IM and Lua libraries are installed, by default it will assume "TECTOOLS_HOME=../..".

In Linux, check the "Building Lua, IM, CD and IUP in Linux" guide.

In Windows, the easiest way to build everything 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. There are files named *make_uname.bat* that build the libraries using **Tecmake**. To build for Windows using Visual C 9.0 (2008) for example, just execute *"make_uname vc9"* , or for the DLLs type *"make_uname dll9"*. The Visual Studio workspaces with the respective projects available in the source package is for debugging purposes only.

If you don't want to install Tecmake you can use the version for Windows systems that is included in the source code package. BUt is is more complicated that its Posix counterpart. First the GNU make must be installed and in the PATH before other makes. In the command line you must specify the compiler path and the TEC_UNAME variable. For example:

```
make -f tecmakewin.mak TEC_UNAME=mingw4 MINGW4=c:/mingw
make -f tecmakewin.mak TEC_UNAME=gcc4 GCC4=c:/cygwin          (Cygwin building for Win32)
```

Make sure you have all the dependencies for the library you want installed, see the documentation bellow.

If you are going to build all the libraries, the makefiles and projects expect the following directory tree:

```
\mylibs\
        cd\
        im\
        lua5.1\
```

Libraries Dependencies

```
cd -> FreeType (included as separate library)
   cdwin* -> gdi32 user32 comdlg32 (system - Windows)
   cdx11* -> X11 (system - UNIX)
   cdgdk+cdcairo* -> gdk-win32-2.0 pangowin32-1.0(system - Windows)
```

```
          -> gdk-x11-2.0 pangox-1.0 (system - UNIX)
          -> cairo gdk_pixbuf-2.0 pango-1.0 gobject-2.0 gmodule-2.0 glib-2.0 (system - Windows/UNIX)
cdgdiplus* -> cd
          -> gdiplus (system - Windows)
cdxrender* -> cd
          -> Xrender Xft (system - UNIX)
cdcairo  -> pangocairo-1.0 cairo (system - Windows/UNIX)
cdpdf    -> pdflib (included as separate library)
cdgl     -> opengl32 (system - Windows)
         -> GL (system - UNIX)
         -> ftgl (included as separate library)
cdlua51  -> cd
         -> lua5.1
cdluacontextplus -> cdlua51
                 -> cdcontextplus
cdluaim51 -> cdlua51
          -> imlua51
cdluapdf51 -> cdlua51
           -> cdpdf
cdluagl51 -> cdlua51
          -> cdgl

(*) In Windows, "cdwin" is called "cd", "cdgdiplus" is called "cdcontextplus".
    In Linux and BSD "cdgdk+cdcairo" is called "cd", "cdxrender" is called "cdcontextplus".
    In IRIX, AIX and SunOS "cdx11" is called "cd", "cdxrender" is called "cdcontextplus".
```

As a general rule (excluding system dependencies and included third party libraries): CD has NO external dependencies, and CDLua depends on Lua and IMLua. Just notice that not all CDLua libraries depend on IMLua.

The Lua bindings for IUP, CD and IM (Makefiles and Pre-compiled binaries) depend on the LuaBinaries distribution. So if you are going to build all use the **LuaBinaries** source package also, not the **Lua.org** original source package. If you like to use another location for the Lua files define LUA_SUFFIX, LUA_INC, LUA_LIB and LUA_BIN before using Tecmake or Tecmake Compact.

In Ubuntu you will need to install the following packages:

```
libx11-dev
libxpm-dev
libxmu-dev
libxft-dev (for the XRender driver)
libgtk2.0-dev (for the GDK driver)
```

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 a code of the simplest driver possible, see CDXX.H and CDXX.C. But first you should read the Internal Architecture.

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 CD_IUP driver parameter needs only the Ihandle* of the **IupCanvas**. But **cdCreateCanvas** must be called after the **IupCanvas** element has been mapped into the native system.

But a call to **IupShow** generates an ACTION callback. And a direct call to **IupMap** can generate a RESIZE_CB callback.

So the best way to create a CD canvas for a IUP canvas is to use the **IupCanvas** MAP_CB callback. This callback will be always called before any other callback.

The application must be linked with the **iupcd** library that it is available in the IUP package.

Resizing the IUP Canvas

When a IupCanvas is resized the CD canvas must be notified of the size change. To do that simply call **cdCanvasActivate** in the RESIZE_CB callback.

Building Lua, IM, CD and IUP in Linux

This is a guide to build all the Lua, IM, CD and IUP libraries in Linux. Notice that you may not use all the libraries, although this guide will show you how to build all of them. You may then choose to build specific libraries.

The Linux used as reference is the Ubuntu distribution.

System Configuration

To build the libraries you will have to download the development version of some packages installed on your system. Although the run time version of some of these packages are already installed, the development versions are usually not. The packages described here are for Ubuntu, but you will be able to identify them for other systems as well.

To build Lua you will need:

```
libreadline5-dev
```

To build IM you will need:

```
g++
```

To build CD you will need:

```
libfreetype6-dev
libx11-dev
libxpm-dev
libxmu-dev
  libxft-dev (for the XRender driver, OPTIONAL)
libgtk2.0-dev (for the GDK driver)
```

To build IUP you will need:

```
libgtk2.0-dev (for the GTK driver) [already installed for CD]
  libmotif-dev and x11proto-print-dev (for the Motif driver, OPTIONAL)
libgl1-mesa-dev and libglu1-mesa-dev (for the IupGLCanvas)
```

To install them you can use the Synaptic Package Manager and select the packages, or can use the command line and type:

```
sudo apt-get install package_name
```

Source Download

Download the "xxx-X.X_Sources.tar.gz" package from the "**Docs and Sources**" directory for the version you want to build. Here are links for the **Files** section in **Source Forge**:

Lua - http://sourceforge.net/projects/luabinaries/files/
IM - http://sourceforge.net/projects/imtoolkit/files/
CD - http://sourceforge.net/projects/canvasdraw/files/
IUP - http://sourceforge.net/projects/iup/files/

Unpacking

To extract the files use the tar command at a common directory, for example:

```
mkdir -p xxxx
cd xxxx

[copy the downloaded files, to the xxxx directory]

tar -xpvzf iup-3.2_Sources.tar.gz
tar -xpvzf cd-5.4_Sources.tar.gz
tar -xpvzf im-3.6.2_Sources.tar.gz
tar -xpvzf lua5_1_4_Sources.tar.gz    [optional, see note bellow]
```

If you are going to build all the libraries, the makefiles and projects expect the following directory tree:

```
/xxxx/
      cd/
      im/
      iup/
      lua5.1/    [optional, see note bellow]
```

If you unpack all the source packages in the same directory, that structure will be automatically created.

Lua

Although we use Lua from LuaBinaries, any Lua installation can also be used. In Ubuntu, the Lua run time package is:

```
lua5.1
```

And the Lua development package is:

```
liblua5.1-0-dev
```

To use them, instead of using the directory "/xxxx/lua5.1" described above, you will have to define some environment variables before building IM, CD and IUP:

```
export LUA_SUFFIX=
export LUA_INC=/usr/include/lua5.1
```

Building

As a general rule (excluding system dependencies): IUP depends on CD and IM, and CD depends on IM. So start by build IM, then CD, then IUP.

To start building go the the "**src**" directory and type "**make**". In IUP there are many "srcxxx" folders, so go to the up directory "iup" and type "**make**" that all the sub folders will be built. For example:

```
cd im/src
make
cd ../..

cd cd/src
make
cd ../..

cd iup
make
cd ..
```

TIP: Instead of building all the libraries, try building only the libraries you are going to use. The provided makefiles will build all the libraries, but take a look inside them and you will figure out how to build just one library.

Pre-compiled Binaries

Instead of building from sources you can try to use the pre-compiled binaries. Usually they were build in the latest Ubuntu versions for 32 and 64 bits. The packages are located in the "**Linux Libraries**" directory under the **Files** section in **Source Forge**, with **"xxx-X.X_Linux26g4_lib.tar.gz"** and **"xxx-X.X_Linux26g4_64_lib.tar.gz"** names.

Do not extract different pre-compiled binaries in the same directory, create a subdirectory for each one, for example:

```
mkdir iup
cd iup
tar -xpvzf ../iup-3.2_Linux26g4_lib.tar.gz
cd ..

mkdir cd
cd cd
tar -xpvzf ../cd-5.4_Linux26g4_lib.tar.gz
cd ..

mkdir im
cd im
tar -xpvzf ../im-3.6.2_Linux26g4_lib.tar.gz
cd ..
```

For the installation instructions bellow, remove the "lib/Linux26g4" from the following examples if you are using the pre-compiled binaries.

Installation (System Directory)

After building you can copy the libraries files to the system directory. If you are inside the main directory, to install the run time libraries you can type, for example:

```
sudo cp -f iup/lib/Linux26g4/*.so /usr/lib              [script version: install ]
sudo cp -f cd/lib/Linux26g4/*.so /usr/lib
sudo cp -f im/lib/Linux26g4/*.so /usr/lib
```

To install the development files, then do:

```
sudo mkdir -p /usr/include/iup                          [script version: install_dev ]
sudo cp -f iup/include/*.h /usr/include/iup
sudo cp -f iup/lib/Linux26g4/*.a /usr/lib

sudo mkdir -p /usr/include/cd
sudo cp -f cd/include/*.h /usr/include/cd
sudo cp -f cd/lib/Linux26g4/*.a /usr/lib

sudo mkdir -p /usr/include/im
sudo cp -fR im/include/*.h /usr/include/im
sudo cp -f im/lib/Linux26g4/*.a /usr/lib
```

Then in your makefile use -Iiup -Icd -Iim for includes. There is no need to specify the libraries directory with -L. Development files are only necessary if you are going to compile an application or library in C/C++ that uses there libraries. To just run Lua scripts they are not necessary.

Installation (Build Directory) [Alternative]

If you **don't** want to copy the run time libraries to your system directory, you can use them from build directory. You will need to add the run time libraries folders to the LD_LIBRARY_PATH, for example:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/xxxx/iup/lib/Linux26g4:/xxxx/cd/lib/Linux26g4:/xxxx/im/lib/Linux26g4
```

And in your makefile will will also need to specify those paths when linking using -L/xxxx/iup/lib/Linux26g4, and for compiling use -I/xxxx/iup/include.

Installation (Lua Modules)

Lua modules in Ubuntu are installed in the "/usr/lib/lua/5.1" directory. So to be able to use the Lua "require" with IUP, CD and IM you must create symbolic links inside that directory.

```
sudo mkdir -p /usr/lib/lua/5.1                         [script version: config_lua_module ]
cd /usr/lib/lua/5.1

sudo ln -fs /usr/lib/libiuplua51.so iuplua.so
sudo ln -fs /usr/lib/libiupluacontrols51.so iupluacontrols.so
...
```

Using those links you do not need any extra configuration.

Installation (Lua Modules) [Alternative]

If you use the **alternative** installation directory, and you also do NOT use the LuaBinaries installation, then you must set the LUA_CPATH environment variable:

```
export LUA_CPATH=./\?.so\;./lib\?.so\;./lib\?51.so\;
```

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 **cd_private.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 (is in fact) cdContextPS()
cdCreateCanvas(CD_PS, "teste.ps"); (will do) canvas->Line = context->Line
```

If the context function is not invoqued then that driver does not need to be linked with the application. This is usefull if the application uses a custom build of the CD library and usefull for additional drivers not included in the main library, like IUP and PDF, that have external dependencies.

Structures

The core implementation defines the structures declared in the cd.h header. But declares an undefined structure called cdCtxCanvas. This structure is defined in each driver according to their needs. But the first member of this structure must be a pointer to the cdCanvas structure.

The drivers need not to implement all functions from the function table, only a few are required.

Here is the definition of the cdContext and cdCanvas structures:

```
struct _cdContext
{
  unsigned long caps;

  /* can NOT be NULL */
  void (*CreateCanvas)(cdCanvas* canvas, void *data);
  void (*InitTable)(cdCanvas* canvas);

  /* can be NULL */
  int (*Play)(cdCanvas* canvas, int xmin, int xmax, int ymin, int ymax, void *data);
  int (*RegisterCallback)(int cb, cdCallback func);
};
```

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

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

  cdVectorFont* vector_font;
  cdSimulation* simulation;
  cdCtxCanvas* ctxcanvas;     // context dependent defintion
  cdContext* context;
};
```

Internally each driver defines its cdCtxCanvas strcuture:

```
struct _cdCtxCanvas
{
  cdCanvas* canvas;

  char* filename;

  int last_line_style;
  int last_fill_mode;
  FILE* file;
};
```

Then it must implement the cdcreatecanvas and cdinittable functions:

```
/* In the driver implementation file */

static void cdcreatecanvas(cdCanvas *canvas, void *data)
{
  cdCtxCanvas* ctxcanvas = (cdCtxCanvas *)malloc(sizeof(cdCtxCanvas));

  // parse data parameters
  ...

  ctxcanvas->canvas = canvas;
  canvas->ctxcanvas = ctxcanvas;

  /* update canvas context */
  canvas->w = (int)(w_mm * res);
  canvas->h = (int)(h_mm * res);
  canvas->w_mm = w_mm;
  canvas->h_mm = h_mm;
  canvas->bpp = 24;
  canvas->xres = res;
  canvas->yres = res;
}

static void cdinittable(cdCanvas* canvas)
{
  canvas->Flush = cdflush;
  canvas->Clear = cdclear;
  canvas->Pixel = cdpixel;
  canvas->Line = cdline;
  canvas->Poly = cdpoly;
  ...
 }

static cdContext cdMetafileContext =
{
  CD_CAP_ALL & ~(CD_CAP_GETIMAGERGB|CD_CAP_IMAGESRV|CD_CAP_REGION|CD_CAP_FONTDIM|CD_CAP_TEXTSIZE),
  cdcreatecanvas,
  cdinittable,
  cdplay,
  cdregistercallback,
};

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

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

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

Attributes

The query mechanism of an attribute is done in the core and does not depends on the driver. 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(cdCtxCanvas* ctxcanvas, char* data)
{
  ctxcanvas->fill_attrib[0] = data[0];
}

static char* get_fill_attrib(cdCtxCanvas* ctxcanvas)
{
  return ctxcanvas->fill_attrib;
}

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

At *createcanvas* in the driver:

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

cdRegisterAttribute(canvas, &fill_attrib);
```

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

Samples

Simple Draw

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

```
cdCanvas* canvas = cdCreateCanvas(CD_NATIVEWINDOW, IupGetAttribute(IupCanvas,"CONID"));
cdCanvasLineStyle(canvas, CD_DASHED);
cdCanvasLine(canvas, 0, 0, 100, 100);
cdKillCanvas(canvas);
```

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

```
cdCanvas* canvas = cdCreateCanvas(CD_NATIVEWINDOW, IupGetAttribute(IupCanvas,"CONID"));
wdCanvasViewport(canvas, 0, 100, 0, 100);
wdCanvasWindow(canvas, -1.5, 1.5, -3000, 3000);
cdCanvasLineStyle(canvas, CD_DASHED);
wdCanvasLine(canvas, -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"));
cdCanvas* db_canvas = cdCreateCanvas(CD_DBUFFER, canvas); cdCanvasActivate(db_canvas); // update canvas size, window could be resized
cdCanvasLineStyle(db_canvas, CD_DASHED);
cdCanvasLine(db_canvas, 10, 10, 50, 50);
cdCanvasFlush(db_canvas);  // swap to the window canvas
cdKillCanvas(db_canvas);
cdKillCanvas(canvas);
```

To draw in a RGB image, use:

```
cdCanvas* canvas = cdCreateCanvasf(CD_IMAGERGB, "%dx%d", width, height);
cdCanvasLineStyle(canvas, CD_DASHED);
cdCanvasLine(canvas, 10, 10, 50, 50);
unsigned char* red = cdCanvasGetAttribute(canvas, "REDIMAGE");
// do something with the raw image data
cdKillCanvas(canvas);
```

Lua Samples

To draw in a RGB image in CDLua for Lua 5:

```
bitmap = cd.CreateBitmap(200,200,cd.RGB)
canvas = cd.CreateCanvas(cd.IMAGERGB, bitmap)
canvas:Font("Times", cd.BOLD, 12)
canvas:Text(10, 10, "Test")
```

```
canvas:KillCanvas()
```

Check the file samples_cdlua5.zip or samples_cdlua5.tar.gz for several samples in Lua. For some of them you will need also the IUP libraries. You can also browse the examples folder.

Screen Capture in Windows

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

```
cdCanvas *canvas = cdCreateCanvas(CD_NATIVEWINDOW, NULL);
cdCanvasGetSize(canvas, &width, &height, NULL, NULL);
// allocate red, green and blue pointers
cdCanvasGetImageRGB(canvas, red, green, blue, 0, 0, width, height);
cdKillCanvas(canvas);
```

Generating an EMF file that contains an IM Image in Lua

```
image = im.FileImageLoadBitmap(image_filename)
canvas = cd.CreateCanvas(cd.EMF,emf_filename.." "..image:Width().."x"..image:Height())
image:cdCanvasPutImageRect(canvas,0,0,0,0)
cd.KillCanvas(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 file simple.zip.

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

All the CD functions are available in Lua, with a few exceptions. We call it **CDLua**. To use them the general application will do require"cdlua", and require"cdluaxxxx" to all other secondary libraries that are needed. The functions and definitions will be available under the table "cd" using the following name rules:

```
cdXxx  -> cd.Xxx    (for functions)
wdXxx  -> cd.wXxx   (for WD functions)
CD_XXX -> cd.XXX    (for definitions)
cdCanvasXXX(canvas,... => canvas:XXX(...   (for methods)
```

New functions (without equivalents in C) were implemented to create and destroy objects that do not exist in C. For instance functions were developed to create and destroy images, pattern, stipple and palette. All the metatables have the "tostring" metamethod implemented to help debuging. Some functions were modified to receive those objects as parameters.

Also the functions which receive values by reference in C were modified. Generally, the values of parameters that would have their values modified are now returned by the function in the same order.

Notice that, as opposed to C, in which enumeration flags are combined with the bitwise operator OR, in Lua the flags are added arithmetically.

In Lua all parameters are checked and a Lua error is emitted when the check fails.

All the objects are garbage collected by the Lua garbage collector, except the canvas because there can be different Lua canvases pointing to the same C canvas. The "tostring" metamethod of the Lua canvas will print both values, Lua and C. The equal method will compare the C canvas value.

Initialization

Lua 5.1 "require" can be used for all the **CDLua** libraries. You can use **require**"**cdlua**" and so on, but the LUA_CPATH must also contains the following:

```
"./lib?51.so;"    [in UNIX]

".\\?51.dll;"     [in Windows]
```

The LuaBinaries distribution already includes these modifications on the default search path.

The simplest form **require**"**cd**" and so on, can not be used because there are CD dynamic libraries with names that will conflict with the names used by **require** during search.

Additionally you can statically link the **CDLua** libraries, but you must call the initialization functions manually. The **`cdlua_open`** function is declared in the header file **`cdlua.h`**, see the example below:

```
#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");

lua_close(L);
}
```

Exchanging Values between C and Lua

Because of some applications that interchange the use of CD canvases in Lua and C, we build a few C functions that are available in "cdlua.h":

```
cdCanvas* cdlua_checkcanvas(lua_State* L, int pos);
void cdlua_pushcanvas(lua_State* L, cdCanvas* canvas);
```

CDLua+IMLua

When CD is used togheter with the IM library in Lua, the CD bitmap and the IM image objects in Lua have a few more methods. These methods allow to map the `imImage` structure to the `cdBitmap` structure and add some facilities to draw on an imImage using a CD canvas. See also 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.

You must link the application with the "cdluaim51" library.

See also the IM documentation.

`int **cdluaim_open**(lua_State* L); [in C] [for Lua 5]`

Must be called to enable the additional methods. Can be called only after CDLua and IMLua were initialized. require"cdluaim" can also be used.
Returns 0 (leaves nothing on the top of the stack).

Available methods

`bitmap:imImageCreate() -> image: imImage [in Lua]`

Creates an imImage from a cdBitmap.

`image:cdCreateBitmap() -> bitmap: cdBitmap [in Lua]`

Creates a cdBitmap from an imImage. The imImage must be a bitmap image, see "image:**IsBitmap**".

`image:cdInitBitmap() -> bitmap: cdBitmap [in Lua]`

Creates a cdBitmap from an imImage, but reuses image data. When the cdBitmap is destroyed, the data is preserved.

`image:cdCanvasPutImageRect(canvas: cdCanvas, x: number, y: number, w: number, h: number, xmin: number, xmax: number, ymin: number, ymax`

Draws the imImage into the given cdCanvas. The imImage must be a bitmap image, see **imImageIsBitmap** in IM documentation.

`image:wdCanvasPutImageRect(canvas: cdCanvas, x: number, y: number, w: number, h: number, xmin: number, xmax: number, ymin: number, ymax`

Draws the imImage into the given cdCanvas using world coordinates. The imImage must be a bitmap image, see **imImageIsBitmap** in IM documentation.

`image:cdCanvasGetImage(canvas: cdCanvas, x: number, y: number) [in Lua]`

Retrieve the imImage data from the given cdCanvas. The imImage must be a IM_RGB/IM_BYTE image.

`image:cdCreateCanvas([res: number]) -> canvas: cdCanvas [in Lua]`

Creates a cdCanvas using the CD_IMAGERGB driver. Resolution is optional, default is 3.8 pixels per milimiter (96.52 DPI). The imImage must be a IM_RGB/IM_BYTE image.

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. It is not necessary to activate a canvas using **cdCanvasActivate**, but some drivers may require that call.

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 **cdCanvasFlush** 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 **cdCanvasClear** 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 **cdCanvasGetSize** function. The canvas' origin is at the bottom left corner of the canvas, but an origin change can be simulated with function **cdCanvasOrigin**. Usually user interface libraries have their origin at the upper right corner, oriented top down. In this case, the function **cdCanvasUpdateYAxis** converts the Y coordinate from this orientation to CD's orientation and vice-versa.

Other

Some canvas contents can be interpreted; the **cdCanvasPlay** 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: cdCanvas) [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**).
- **CD_GL** = Native Window (**cdgl.h**).

Device-Based Drivers

- **CD_CLIPBOARD** = Clipboard (**cdclipbd.h**).
- **CD_PRINTER** = Printer (**cdprint.h**).
  CD_PICTURE = Picture in memory (**cdpicture.h**).

Image-Based Drivers

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

File-Based Drivers

- **CD_PDF** = Adobe Portable Document Format (**cdpdf.h**).
- **CD_PS** = PostScript File (**cdps.h**).
- **CD_SVG** = Scalable Vector Graphics (**cdsvg.h**).
- **CD_METAFILE** = Internal CD Metafile (**cdmf.h**).
- **CD_DEBUG** = Internal CD Debug Log (**cddebug.h**).
- **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_WMF** = Microsoft Windows Metafile (**cdwmf.h**). Works only in MS Windows systems.

```
cdCanvas* cdCreateCanvasf(cdContext *ctx, const 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: cdCanvas) [in Lua]
```

Destroys a previously created canvas. If this function is not called in Lua, the garbage collector will call it.

```
int cdCanvasActivate(cdCanvas *canvas); [in C]

canvas:Activate(canvas: cdCanvas) -> (status: number) [in Lua]
```

Activates a canvas for drawing. This is used only for a few drivers. Native Window and IUP drivers will update the canvas size if the window size has changed. Double Buffer driver will recreate the image buffer if the window canvas size has changed. In these cases the function MUST be called, for other drivers is useless. Returns CD_ERROR or CD_OK.

```
void cdCanvasDeactivate(cdCanvas* canvas); [in C]

canvas:Deactivate(canvas: cdCanvas) [in Lua]
```

Called when the application has finished drawing in the canvas. It is optional, but if used for the Native Window driver in Windows when the handle can not be retained, the drawing can only be done again after a **cdCanvasActivate**. On some drivers will simply call Flush.

```
int cdUseContextPlus(int use); [in C]

cd.UseContextPlus(use: boolean) -> (old_use: boolean) [in Lua]
```

Activates or deactivates the use of an external context for the next calls of the **`cdCreateCanvas`** function.

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

cd.InitContextPlus() [in Lua]
```

Initializes the context driver to use another context replacing the standard drivers. This functions is only available when a library containing a "ContextPlus" context driver is used. See the Cairo, GDI+ and XRender base drivers. Those libraries does not support XOR write mode, but has support for anti-aliasing and alpha for transparency.

In Lua, when using require"cdluacontextplus" this function will be automatically called.

```
cdContext* cdCanvasGetContext(cdCanvas *canvas); [in C]

canvas:GetContext(canvas: cdCanvas) -> (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 ('&.html 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 cdCanvasSimulate(cdCanvas* canvas, int mode); [in C]
```

```
canvas:Simulate(mode: number) -> (old_mode: number) [in Lua]
```

Activates the simulation of one or more primitives. It is ignored for the canvas in the ImageRGB context, because 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.

If simulation is activated the driver transformation matrix is disabled.

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

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

CD_SIM_NONE - Deactivates all kinds of simulation.
CD_SIM_LINE
CD_SIM_RECT
CD_SIM_BOX
CD_SIM_ARC
CD_SIM_SECTOR
CD_SIM_CHORD
CD_SIM_POLYLINE
CD_SIM_POLYGON
CD_SIM_TEXT
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, CD_SIM_CHORD and CD_SIM_POLYGON.

Extras

```
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.

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

Releases the memory allocated by the CDLua binding.

```
cdCanvas* cdlua_checkcanvas(lua_State* L, int pos); [for Lua 5]
```

Returns the canvas in the Lua stack at position pos. The function will call lua_error if there is not a valid canvas in the stack at the given position.

```
void cdlua_pushcanvas(lua_State* L, cdCanvas* canvas);
```

Pushes the given canvas into the stack.

Canvas Control

```
void cdCanvasClear(cdCanvas* canvas); [in C]
```

```
canvas: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 cdCanvasFlush(cdCanvas* canvas); [in C]
```

```
canvas: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* cdCanvasSaveState(cdCanvas* canvas); [in C]
```

```
canvas:SaveState() -> (state: cdState) [in Lua]
```

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

```
void cdCanvasRestoreState(cdCanvas* canvas, cdState* state); [in C]
```

```
canvas:RestoreState(state: cdState) [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: cdState) [in Lua]
```

Releases the memory allocated by the **cdSaveState** function. If this function is not called in Lua, the garbage collector will call it.

```
void cdCanvasSetAttribute(cdCanvas* canvas, const char* name, char* data); [in C]
```

```
canvas: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 cdCanvasSetfAttribute(cdCanvas* canvas, const char* name, const 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* cdCanvasGetAttribute(cdCanvas* canvas, const char* name); [in C]
```

```
canvas: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 cdCanvasGetSize(cdCanvas* canvas, int *width, int *height, double *width_mm, double *height_mm); [in C]
```

```
canvas:GetSize() -> (width, height, 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.

```
int cdCanvasYAxisMode(cdCanvas* canvas, int invert); [in C]
```

```
canvas:YAxisMode(invert: number) -> (old_invert: number) [in Lua]
```

Controls the orientation of the Y axis. Internally in some drivers the native axis orientation is top-bottom, so the CD primitives must invert the Y axis since the CD orientation is bottom-top. Using CD_QUERY will return the current Y axis mode, if needs to be invert or not. Using 1 or 0 you can control if the Y axis should be inverted or not independent from the native orientation, with that you can in fact invert the orientation of the CD primitives.

```
int cdCanvasUpdateYAxis(cdCanvas* canvas, int *y); [in C]
double cdfCanvasUpdateYAxis(cdCanvas* canvas, double *y); [in C]
int cdCanvasInvertYAxis(cdCanvas* canvas, int y); [in C]
double cdfCanvasInvertYAxis(cdCanvas* canvas, double y); [in C]
```

```
canvas:UpdateYAxis(yc: number) -> (yr: number) [in Lua]
canvas:InvertYAxis(yc: number) -> (yr: number) [in Lua]
```

Converts the coordinate system of the CD library into the internal system of the active canvas' driver, and the other way round, if they are invert, or else do nothing. This is just "y = height-1 - y". It returns the changed value. The "Invert" will always invert the given value, the "Update" function will invert only if the canvas has the Y axis inverted.

```
void cdCanvasMM2Pixel(cdCanvas* canvas, double mm_dx, double mm_dy, int *dx, int *dy); [in C]
void cdfCanvasMM2Pixel(cdCanvas* canvas, double mm_dx, double mm_dy, double *dx, double *dy); [in C]
```

```
canvas:MM2Pixel(mm_dx, mm_dy: number) -> (dx, dy: number) [in Lua]
canvas:fMM2Pixel(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 cdCanvasPixel2MM(cdCanvas* canvas, int dx, int dy, double *mm_dx, double *mm_dy); [in C]
void cdfCanvasPixel2MM(cdCanvas* canvas, double dx, double dy, double *mm_dx, double *mm_dy); [in C]
```

```
canvas:Pixel2MM(dx, dy: number) -> (mm_dx, mm_dy: number) [in Lua]
canvas:fPixel2MM(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 cdCanvasOrigin(cdCanvas* canvas, int x, int y); [in C]
void cdfCanvasOrigin(cdCanvas* canvas, double x, double y); [in C]

canvas:Origin(x, y: number) [in Lua]
canvas:fOrigin(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 canvas in the respective driver. It is not related with WD nor Transformation Matrix. Default values: (0, 0)

```
void cdCanvasGetOrigin(cdCanvas* canvas, int *x, int *y); [in C]
void cdfCanvasGetOrigin(cdCanvas* canvas, double *x, double *y); [in C]

canvas:GetOrigin() -> (x, y: number) [in Lua]
canvas:fGetOrigin() -> (x, y: number) [in Lua]
```

Returns the origin.

Transformation Matrix

```
void cdCanvasTransform(cdCanvas* canvas, const double* matrix); [in C]

canvas:Transform(matrix: table) [in Lua]
```

Defines a transformation matrix with 6 elements. If the matrix is NULL, the transformation is reset to the identity. Default value: NULL.

The matrix contains scale, rotation and translation elements as follows:

```
|x'|   |sx*cos(angle)    -sin(angle)  dx|   |x|                     |0   2   4|
|y'| = |   sin(angle)  sy*cos(angle)  dy| * |y|      with indices   |1   3   5|
                                            |1|
```

But notice that the indices are different of the **cdCanvasVectorTextTransform**.

Functions that retrieve images from the canvas are not affected by the transformation matrix, such as **GetImage**, **GetImageRGB** and **ScrollArea**.

Transformation matrix is independent of the **World Coordinate** and **Origin** functions. And those are affected if a transformation is set, just like other regular primitives.

```
double* cdCanvasGetTransform(cdCanvas* canvas); [in C]

canvas:GetTransformation() -> (matrix: table) [in Lua]
```

Returns the transformation matrix. If the identity is set, returns NULL.

```
void cdCanvasTransforMultiply(cdCanvas* canvas, const double* matrix); [in C]

canvas:TransformMultiply(matrix: table) [in Lua]
```

Left multiply the current transformation by the given transformation.

```
void cdCanvasTransformTranslate(cdCanvas* canvas, double dx, double dy); [in C]

canvas:TransformTranslate(dx, dy: number) [in Lua]
```

Applies a translation to the current transformation.

```
void cdCanvasTransformScale(cdCanvas* canvas, double sx, double sy); [in C]

canvas:TransformScale(sx, sy: number) [in Lua]
```

Applies a scale to the current transformation.

```
void cdCanvasTransformRotate(cdCanvas* canvas, double angle); [in C]

canvas:TransformRotate(angle: number) [in Lua]
```

Applies a rotation to the current transformation. Angle is in degrees, oriented counter-clockwise from the horizontal axis.

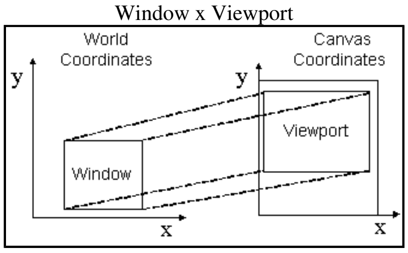
```
void cdCanvasTransformPoint(cdCanvas* canvas, int x, int y, int *tx, int *ty); [in C]
void cdfCanvasTransformPoint(cdCanvas* canvas, double x, double y, double *tx, double *ty); [in C]

canvas:TransformPoint(x, y: number) -> (tx, ty: number) [in Lua]
canvas:fTransformPoint(x, y: number) -> (tx, ty: number) [in Lua]
```

Applies a transformation to a given point.

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.

All World Coordinate drawing in all drivers are simulated using other CD primitives and do NOT depend or use the **cdCanvasTransform** transformation matrix.

```
void wdCanvasWindow(cdCanvas* canvas, double xmin, double xmax, double ymin, double ymax); [in C]
```

```
canvas: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 wdCanvasGetWindow(cdCanvas* canvas, double *xmin, double *xmax, double *ymin, double *ymax); [in C]
```

```
canvas: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 wdCanvasViewport(cdCanvas* canvas, int xmin, int xmax, int ymin, int ymax); [in C]
```

```
canvas: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 wdCanvasGetViewport(cdCanvas* canvas, int *xmin, int *xmax, int *ymin, int *ymax); [in C]
```

```
canvas: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 wdCanvasWorld2Canvas(cdCanvas* canvas, double xw, double yw, int *xv, int *yv); [in C]
```

```
canvas: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 wdCanvasCanvas2World(cdCanvas* canvas, int xv, int yv, double *xw, double *yw); [in C]
```

```
canvas: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.

```
void wdCanvasSetTransform(cdCanvas* canvas, double sx, double sy, double tx, double ty); [in C]
```

```
canvas:wSetTransform(sx, sy, tx, ty: number) [in Lua]
```

Configures the world coordinate system transformation to be used to convert world coordinates (with values in real numbers) into canvas coordinates (with values in integers). The transformation is automatically set by **wdCanvasWindow** and **wdCanvasViewport**. This has NO relation with **cdCanvasTransform**.

```
void wdCanvasGetTransform(cdCanvas* canvas, double *sx, double *sy, double *tx, double *ty); [in C]
```

```
canvas:wGetTransform() -> (sx, sy, tx, ty: number) [in Lua]
```

Queries the current transformation 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 wdCanvasTranslate(cdCanvas* canvas, double dtx, double dty); [in C]
```

```
canvas:wTranslate(dtx, dty: number) [in Lua]
```

Translates the transformation by a delta, by adding the given values to the current tx and ty values.

```
void wdCanvasScale(cdCanvas* canvas, double dsx, double dsy); [in C]
```

```
canvas:wScale(dsx, dsy: number) [in Lua]
```

Scales the transformation by a delta, by multiplying the given values by the current sx and sy values.

Extra

```
void wdCanvasHardcopy(cdCanvas *canvas, cdContext* ctx, void *data, void(*draw_func)(cdCanvas *canvas_copy)); [in C]
```

```
canvas:wCanvasHardcopy(ctx: number, data: string or userdata, draw_func: function) [in Lua]
```

Creates a new canvas, 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.

It is usually used for "hard copies" of drawings (print equivalent copy). The most common used contexts are Printer, PS and PDF.

General Attributes

```
long int cdCanvasForeground(cdCanvas* canvas, long int color); [in C]
void cdCanvasSetForeground(cdCanvas* canvas, long int color); [in C]
```

```
canvas:Foreground(color: lightuserdata) -> (old_color: lightuserdata) [in Lua]
canvas:SetForeground(color: lightuserdata) [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.

Notice that CD_QUERY conflicts with color RGBA=(255,255,255,0) (full transparent white). Use **SetForeground** to avoid the conflict. See also Color Coding.

```
long int cdCanvasBackground(cdCanvas* canvas, long int color); [in C]
void cdCanvasSetBackground(cdCanvas* canvas, long int color); [in C]

canvas:Background(color: lightuserdata) -> (old_color: lightuserdata) [in Lua]
canvas:SetBackground(color: lightuserdata) [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 **Clear** function. The background color only makes sense for **Clear** and for primitives affected by the background opacity attribute. Default value: **`CD_WHITE`**. Value **`CD_QUERY`** simply returns the current value.

Notice that CD_QUERY conflicts with color RGBA=(255,255,255,0) (full transparent white). Use **SetBackground** to avoid the conflict. See also Color Coding.

```
int cdCanvasWriteMode(cdCanvas* canvas, int mode); [in C]

canvas: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 **Clip** function activates and deactivaes the clipping.

```
int cdCanvasClip(cdCanvas* canvas, int mode); [in C]

canvas:Clip(mode: number) -> (old_mode: number) [in Lua]
```

Activates or deactivates clipping. Returns the previous status. Values: **CD_CLIPAREA, CD_CLIPPOLYGON, CD_CLIPREGION** 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.

The defined clipping area, polygon and complex regions are stored internally, so you may define them independently and switch between area, polygon and complex region without having to define them again. Also if the active clipping region is re-defined it immediately becomes the current clipping region.

```
void cdCanvasClipArea(cdCanvas* canvas, int xmin, int xmax, int ymin, int ymax); [in C]
void cdfCanvasClipArea(cdCanvas* canvas, double xmin, double xmax, double ymin, double ymax); [in C]
void wdCanvasClipArea(cdCanvas* canvas, double xmin, double xmax, double ymin, double ymax); (WC) [in C]

canvas:ClipArea(xmin, xmax, ymin, ymax: number) [in Lua]
canvas:wClipArea(xmin, xmax, ymin, ymax: number) (WC) [in Lua]
```

Defines the current rectangle for clipping. Only the points in the interval ***xmin<= x <= xmax*** and ***ymin <= y <= ymax*** will be printed. Default region: (0, w-1, 0, h-1).

```
int cdCanvasGetClipArea(cdCanvas* canvas, int *xmin, int *xmax, int *ymin, int *ymax); [in C]
int cdfCanvasGetClipArea(cdCanvas* canvas, double *xmin, double *xmax, double *ymin, double *ymax); [in C]
int wdCanvasGetClipArea(cdCanvas* canvas, double *xmin, double *xmax, double *ymin, double *ymax); (WC) [in C]

canvas:GetClipArea() -> (xmin, xmax, ymin, ymax, status: number) [in Lua]
canvas: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.

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.

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()**.

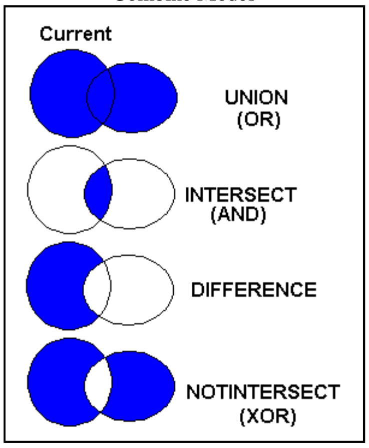
To make the region active you must call **cdClip**(**`CD_CLIPREGION`**). For other clipping regions see Clipping.

Complex clipping regions are not saved by **cdSaveState**.

```
int cdCanvasRegionCombineMode(cdCanvas* canvas, int mode); [in C]

canvas: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 cdCanvasIsPointInRegion(cdCanvas* canvas, int x, int y); [in C]
int wdCanvasIsPointInRegion(cdCanvas* canvas, double x, double y); (WC) [in C]

canvas:IsPointInRegion(x, y: number) -> (status: boolean) [in Lua]
canvas:wIsPointInRegion(x, y: number) -> (status: boolean) [in Lua]
```

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

```
void cdCanvasOffsetRegion(cdCanvas* canvas, int dx, int dy); [in C]
void wdCanvasOffsetRegion(cdCanvas* canvas, double dx, double dy); (WC) [in C]

canvas:OffsetRegion(dx, dy: number) [in Lua]
canvas: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 reconstructed.

```
void cdCanvasGetRegionBox(cdCanvas* canvas, int *xmin, int *xmax, int *ymin, int *ymax); [in C]
void wdCanvasGetRegionBox(cdCanvas* canvas, double *xmin, double *xmax, double *ymin, double *ymax); (WC) [in C]

canvas:GetRegionBox() -> (xmin, xmax, ymin, ymax, status: number) [in Lua]
canvas:wGetRegionBox() -> (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.

All marks in all drivers are simulated using other CD primitives, except **cdCanvasPixel**.

```
void cdCanvasPixel(cdCanvas* canvas, int x, int y, long int color); [in C]
void wdCanvasPixel(cdCanvas* canvas, double x, double y, long int color); (WC) [in C]

canvas:Pixel(x, y: number, color: lightuserdata) [in Lua]
canvas:wPixel(x, y: number, color: lightuserdata) (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. It can be very slow on some drivers. Sometimes it is implemented as a rectangle with size 1x1.

```
void cdCanvasMark(cdCanvas* canvas, int x, int y); [in C]
void wdCanvasMark(cdCanvas* canvas, double x, double y); (WC) [in C]

canvas:Mark(x, y: number) [in Lua]
canvas: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 **cdCanvasBegin** and **cdCanvasEnd** 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 **cdCanvasLine**.

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

```
cdCanvasInteriorStyle(canvas, CD_SOLID);
cdCanvasLineStyle(canvas, CD_CONTINUOUS);
cdCanvasLineWidth(canvas, 1);
```

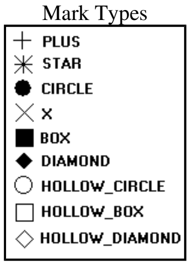
This will greatly increase this function's performance. Also in this case, if the mark is very small, we suggest using the **cdCanvasPixel** function so that the application itself draws the mark. In many cases, this also increases this function's performance.

Attributes

```
int cdCanvasMarkType(cdCanvas* canvas, int type); [in C]

canvas: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 cdCanvasMarkSize(cdCanvas* canvas, int size); [in C]
double wdCanvasMarkSize(cdCanvas* canvas, double size); (WC) [in C]

canvas:MarkSize(size: number) -> (old_size: number) [in Lua]
canvas: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 **Line** function includes the 2 given points and draws the line using the foreground color. Line thickness is controlled by the **LineWidth** function. By using function **LineStyle** 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 cdCanvasLine(cdCanvas* canvas, int x1, int y1, int x2, int y2); [in C]
void cdfCanvasLine(cdCanvas* canvas, double x1, double y1, double x2, double y2); [in C]
void wdCanvasLine(cdCanvas* canvas, double x1, double y1, double x2, double y2); (WC) [in C]

canvas:Line(x1, y1, x2, y2: number) [in Lua]
canvas:fLine(x1, y1, x2, y2: number) [in Lua]
canvas: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 cdCanvasRect(cdCanvas* canvas, int xmin, int xmax, int ymin, int ymax); [in C]
void cdfCanvasRect(cdCanvas* canvas, double xmin, double xmax, double ymin, double ymax); [in C]
void wdCanvasRect(cdCanvas* canvas, double xmin, double xmax, double ymin, double ymax); (WC) [in C]

canvas:Rect(xmin, xmax, ymin, ymax: number) [in Lua]
canvas:fRect(xmin, xmax, ymin, ymax: number) [in Lua]
canvas: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 cdCanvasArc(cdCanvas* canvas, int xc, int yc, int w, int h, double angle1, double angle2); [in C]
void cdfCanvasArc(cdCanvas* canvas, double xc, double yc, double w, double h, double angle1, double angle2); [in C]
void wdCanvasArc(cdCanvas* canvas, double xc, double yc, double w, double h, double angle1, double angle2); (WC) [in C]

canvas:Arc(xc, yc, w, h, angle1, angle2: number) [in Lua]
canvas:fArc(xc, yc, w, h, angle1, angle2: number) [in Lua]
canvas: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.

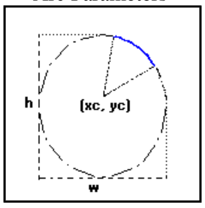
The coordinate **(xc,yc)** defines the center of the ellipse. Dimensions **w** and **h** define the elliptic axes X and Y, respectively.

Angles **angle1** and **angle2** are in degrees and oriented counter-clockwise. They define the arc start 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. If **angle2** is less than **angle1** it will be increased by 360 until it is greater than **angle1**.

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))**.

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

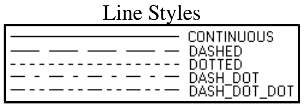


Attributes

```
int cdCanvasLineStyle(cdCanvas* canvas, int style); [in C]

canvas: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. The spaces are drawn with the background color, except when back opacity is transparent then the background is left unchanged. See BackOpacity.

Line Styles



```
void cdCanvasLineStyleDashes(cdCanvas* canvas, const int* dashes, int count); [in C]

canvas:LineStyleDashes(dashes: table, count: number) -> (old_style: number) [in Lua]
```

Defines the custom line style dashes. The first value is the length of the first dash, the second value is the length 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. Sizes are in pixels.

```
int cdCanvasLineWidth(cdCanvas* canvas, int width); [in C]
double wdCanvasLineWidth(cdCanvas* canvas, double width_mm); (WC) [in C]

canvas:LineWidth(width: number) -> (old_width: number) [in Lua]
canvas:wLineWidth(width_mm: number) -> (old_width_mm: number) (WC) [in Lua]
```

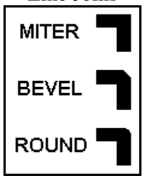
Configures the width of the current line (in pixels). Returns the previous value. Default value: 1. Value **CD_QUERY** simply returns the current value. Valid width interval: >= 1.

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

```
int cdCanvasLineJoin(cdCanvas* canvas, int style); [in C]

canvas: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.

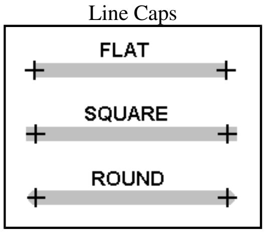
Line Joins



```
int cdCanvasLineCap(cdCanvas* canvas, int style); [in C]

canvas: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



Open, Closed and Filled Polygons, Bezier Lines and Regions Creation

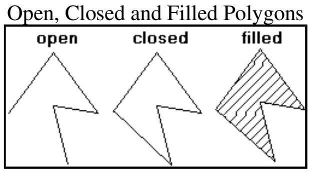
The functions **cdBegin**, **cdVertex** and **cdEnd** are use for many situations. **cdBegin** is called once, **cdVertex** can be called many times, and **cdEnd** is called once to actually do something. If you call **cdBegin** again before **cdEnd** the process is restarted, except for **cdBegin(CD_REGION)** that can contains one or more polygons inside.

```
void cdCanvasBegin(cdCanvas* canvas, int mode); [in C]

canvas:Begin(mode: number) [in Lua]
```

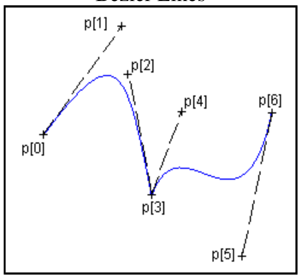
Starts defining a polygon to be drawn (or filled) according to the mode: **CD_CLOSED_LINES**, **CD_OPEN_LINES**, **CD_FILL, CD_CLIP, CD_REGION** or **CD_BEZIER**. Do not create embedded polygons, that is, do not call function **cdBegin** twice without a call to **cdEnd** in between.

- **CD_OPEN_LINES:** connects all the points at cdEnd. Depends on line width and line style attributes.
- **CD_CLOSED_LINES:** connects all the points at cdEnd and connects the last point to the first. Depends on line width and line style attributes.
- **CD_FILL:** connects the last point to the first and fills the resulting polygon according to the current interior style. When the interior style **CD_HOLLOW** is defined the it behaves as if the mode were **CD_CLOSED_LINES.**
- **CD_CLIP: i**nstead of creating a polygon to be drawn, creates a polygon to define a polygonal clipping region.
- **CD_BEZIER:** defines the points of a bezier curve. There must be at least 4 points: *start*, *control*, *control* and *end*. To specify a sequence of curves use 3 more points for each curve: *control*, *control*, *end*, *control*, *control*, *end*, ... The end point is used as start point for the next curve.
- **CD_REGION**: starts the creation of a complex region for clipping. All calls to **cdBox**, **cdSector**, **cdChord, Filled Polygons** and **cdText** will be composed in a region for clipping. See Regions documentation.
- **CD_PATH**: creates a path composed of several primitives that can be line draw, filled or used as clipping. Must call **cdCanvasPathSet** to configure the action between sequences of **cdCanvasVertex**.

Open, Closed and Filled Polygons



Bezier Lines



```
void cdCanvasVertex(cdCanvas* canvas, int x, int y); [in C]
void cdfCanvasVertex(cdCanvas* canvas, double x, double y); [in C]
void wdCanvasVertex(cdCanvas* canvas, double x, double y); (WC) [in C]

canvas:Vertex(x, y: number) [in Lua]
canvas:wVertex(x, y: number) (WC) [in Lua]
```

Adds a vertex to the polygon definition.

```
void cdCanvasEnd(cdCanvas* canvas); [in C]

canvas:End() [in Lua]
```

Ends the polygon's definition and draws it.

```
void cdCanvasPathSet(cdCanvas* canvas, int action); [in C]

canvas:PathSet(action: number) [in Lua]
```

Configures the action between sequences of **cdCanvasVertex**. **action** can be:

- **CD_PATH_NEW** - creates a new empty path. Useful if more than one path is configured. **cdCanvasBegin**(CD_PATH) already creates a new path.
- **CD_PATH_MOVETO** - moves the current position to the given coordinates. Must be followed by 1 call to **cdCanvasVertex**, **cdfCanvasVertex**, or **wdCanvasVertex**.
- **CD_PATH_LINETO** - adds a line to the path from the current position to the given coordinates. The current position is updated to the given coordinates. If there is no current position, nothing is connected and only the current position is updated. Must be followed by 1 call to **cdCanvasVertex**, **cdfCanvasVertex**, or **wdCanvasVertex**.
- **CD_PATH_ARC** - adds an arc to the path. If there is a current position adds also a line from the current position to the start of the arc. The end of the arc becomes the current position. Must be followed by 3 calls to **cdCanvasVertex**, **cdfCanvasVertex**, or **wdCanvasVertex**. One for the center of the arc (xc,yc), one for the bounding rectangle size (w,h), and one for the start and end angles (angle1,angle2). Angles are in degrees and oriented counter-clockwise, but angle2 can be smaller than angle1 to describe a clockwise arc. When using integer coordinates angles must be multiplied by 1000.
- **CD_PATH_CURVETO** - adds a bezier curve to the path. If there is no current position, the first point will be used twice. The end point becomes the current position. Must be followed by 3 calls to **cdCanvasVertex**, **cdfCanvasVertex**, or **wdCanvasVertex**. Must be first control point (x1,y1) + second control point (x2,y2) + end point (x3,y3).
- **CD_PATH_CLOSE** - adds a line to the path that connects the last point with the first point of the path, closing it.
- **CD_PATH_FILL** - fills the path with the current fill attributes, then the path is discarded.
- **CD_PATH_STROKE** - strokes the path with the current line attributes, then the path is discarded.
- **CD_PATH_FILLSTROKE** - fills the path with the current fill attributes, strokes the path with the current line attributes, then the path is discarded.
- **CD_PATH_CLIP** - use the path as a clipping area to be intersected with the current clipping area, then the path is discarded.

So the normal path creation to draw a line will do:

```
cdCanvasBegin(canvas, CD_PATH);
cdCanvasPathSet(canvas, CD_PATH_MOVETO);
cdCanvasVertex(canvas, x1, y1);
cdCanvasPathSet(canvas, CD_PATH_LINETO);
```

```
cdCanvasVertex(canvas, x2, y2);
cdCanvasPathSet(canvas, CD_PATH_CURVETO);
cdCanvasVertex(canvas, x3, y3);  /* control point for start point */
cdCanvasVertex(canvas, x4, y4);  /* control point for end point */
cdCanvasVertex(canvas, x5, y5);  /* end point */
cdCanvasPathSet(canvas, CD_PATH_ARC);
cdCanvasVertex(canvas, x6, y6);  /* center */
cdCanvasVertex(canvas, x7, y7);  /* width, height */
cdCanvasVertex(canvas, x8, y8);  /* start angle, end angle (degrees / 1000) */
cdCanvasPathSet(canvas, CD_PATH_STROKE);
cdCanvasEnd(canvas);
```

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 and STIPPLE style depend on the foreground color, background color and on the back opacity attribute. The hatch lines drawn with this style do not depend on the other line attributes. The PATTERN style depends only on global canvas attributes.

The filled area includes the line at the edge of the area. So if you draw a filled rectangle, sector or polygon on top of a non filled one using the same coordinates, no style and 1 pixel width, the non filled primitive should be obscured by the filled primitive. But depending on the driver implementation some pixels at the edges may be not included. IMPORTANT: In the Postscript and PDF drivers the line at the edge is not included at all.

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 cdCanvasBox(cdCanvas* canvas, int xmin, int xmax, int ymin, int ymax); [in C]
void cdfCanvasBox(cdCanvas* canvas, double xmin, double xmax, double ymin, double ymax); [in C]
void wdCanvasBox(cdCanvas* canvas, double xmin, double xmax, double ymin, double ymax); (WC) [in C]

canvas:Box(xmin, xmax, ymin, ymax: number) [in Lua]
canvas:fBox(xmin, xmax, ymin, ymax: number) [in Lua]
canvas: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 cdCanvasSector(cdCanvas* canvas, int xc, int yc, int w, int h, double angle1, double angle2); [in C]
void cdfCanvasSector(cdCanvas* canvas, double xc, double yc, double w, double h, double angle1, double angle2); [in C]
void wdCanvasSector(cdCanvas* canvas, double xc, double yc, double w, double h, double angle1, double angle2); (WC) [in C]

canvas:Sector(xc, yc, w, h, angle1, angle2: number) [in Lua]
canvas:fSector(xc, yc, w, h, angle1, angle2: number) [in Lua]
canvas: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.

The coordinate **(xc,yc)** defines the center of the ellipse. Dimensions **w** and **h** define the elliptic axes X and Y, respectively.

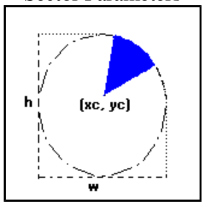
Angles **angle1** and **angle2** are in degrees and oriented counter-clockwise. They define the arc start and end, but they are not the angle relative to the center, except when w==h and the ellipse is reduced to a circle. The arc starts at the point **(xc+(w/2)*cos(angle1), yc+(h/2)*sin(angle1))** and 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. If **angle2** is less than **angle1** it will be increased by 360 until it is greater than **angle1**.

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))**.

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 **cdCanvasArc**, plus two lines connecting to the center.

Sector Parameters

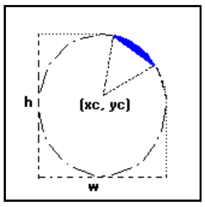


```
void cdCanvasChord(cdCanvas* canvas, int xc, int yc, int w, int h, double angle1, double angle2); [in C]
void cdfCanvasChord(cdCanvas* canvas, double xc, double yc, double w, double h, double angle1, double angle2); [in C]
void wdCanvasChord(cdCanvas* canvas, double xc, double yc, double w, double h, double angle1, double angle2); (WC) [in C]

canvas:Chord(xc, yc, w, h, angle1, angle2: number) [in Lua]
canvas:fChord(xc, yc, w, h, angle1, angle2: number) [in Lua]
canvas: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



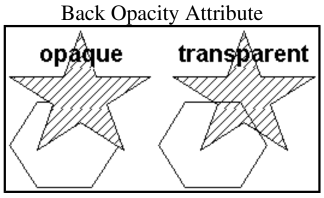
Attributes

```
int cdCanvasBackOpacity(cdCanvas* canvas, int opacity); [in C]
```

```
canvas:BackOpacity(opacity: number) -> (old_opacity: number) [in Lua]
```

Configures the background opacity to filling primitives based on the foreground and background colors. Note that only when InteriorStyle is **CD_HATCH** or **CD_STIPPLE** that backopacity is used. Values: **CD_TRANSPARENT** or **CD_OPAQUE**. If it is opaque the primitive will erase whatever is in the background with the background color. If it is transparent, only the foreground color is painted. It returns the previous value. Default value: **CD_TRANSPARENT**. Value **CD_QUERY** simply returns the current value. In some drivers is always opaque.

Back Opacity Attribute

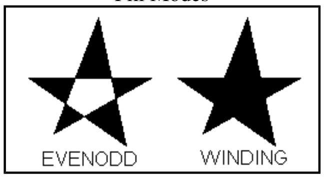


```
int cdCanvasFillMode(cdCanvas* canvas, int mode); [in C]
```

```
canvas: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 cdCanvasInteriorStyle(cdCanvas* canvas, int style); [in C]
```

```
canvas: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 only **CD_HATCH** and **CD_STIPPLE** are affected by the backopacity. 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, when they are selected the state of the attribute is not changed.

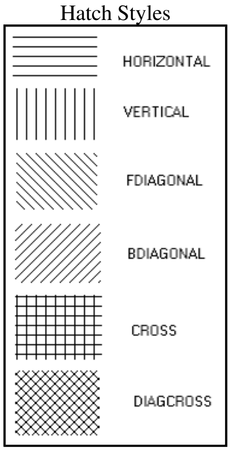
When the style **CD_HOLLOW** is defined, functions **cdBox** and **cdSector** behave as their equivalent **cdRect** and **cdArc+Lines**, and the polygons with style **CD_FILL** behave like **CD_CLOSED_LINES**.

```
int cdCanvasHatch(cdCanvas* canvas, int style); [in C]
```

```
canvas: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**. The lines are drawn with the foreground color, and the background is drawn with the background color if back opacity is opaque. Returns the previous value. Default value: **CD_HORIZONTAL**. Value **CD_QUERY** simply returns the current value. The foreground and background colors must be set before setting the style. In some drivers is always opaque.

Hatch Styles



```
void cdCanvasStipple(cdCanvas* canvas, int w, int h, const unsigned char *fgbg) [in C]
```

```
canvas:Stipple(stipple: cdStipple) [in Lua]
```

Defines a **wxh** matrix of zeros (0) and ones (1). 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 does not need to be stored by the application, as it is internally replicated by the library. In some drivers is always opaque. The foreground and background colors must be set before setting the style.

```
void wdCanvasStipple(cdCanvas* canvas, int w, int h, const unsigned char *fgbg, double w_mm, double h_mm); [in C]
```

```
canvas:wStipple(stipple: cdStipple, 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 new size in pixels will be an integer factor of the original size that is closets to the size in millimeters. The use of this function may produce very large or very small stipples.

```
unsigned char* cdCanvasGetStipple(cdCanvas* canvas, int* w, int* h); [in C]
```

```
canvas:GetStipple() - > (stipple: cdStipple) [in Lua]
```

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

```
void cdCanvasPattern(cdCanvas* canvas, int w, int h, const long int *color); [in C]
```

```
canvas:Pattern(pattern: cdPattern) [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 does not need to be stored by the application, as it is internally replicated by the library.

```
void wdCanvasPattern(cdCanvas* canvas, int w, int h, const long int *color, double w_mm, double h_mm); [in C]
```

```
canvas:wPattern(pattern: cdPattern, 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 new size in pixels will be an integer factor of the original size that is closets to the size in millimeters. The use of this function may produce very large or very small patterns.

```
long int* cdCanvasGetPattern(cdCanvas* canvas, int* w, int* h); [in C]
```

```
canvas:GetPattern() - > (pattern: cdPattern) [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: cdPattern)
```

Creates a pattern in Lua.

```
cd.KillPattern(pattern: cdPattern)
```

Destroys the created pattern and liberates allocated memory. If this function is not called in Lua, the garbage collector will call it.

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

Creates a stipple in Lua.

```
cd.KillStipple(stipple: cdStipple)
```

Destroys the created stipple and liberates allocated memory. If this function is not called in Lua, the garbage collector will call it.

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 lightuserdata, 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 at least 4 standard typefaces: "System" (which depends on the driver and platform), "Courier" (mono spaced with serif), "Times" (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 other typefaces or native driver typefaces with the function **NativeFont**, but they may work in a reduced set of drivers.

You may retrieve the dimensions of the selected font with function **GetFontDim**. Also you may retrieve the bounding box of a specific text before drawing by using the **GetTextSize** and **GetTextBox** functions.

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

```
void cdCanvasText(cdCanvas* canvas, int x, int y, const char* text); [in C]
void cdfCanvasText(cdCanvas* canvas, double x, double y, const char* text); [in C]
void wdCanvasText(cdCanvas* canvas, double x, double y, const char* text); (WC) [in C]
```

```
canvas:Text(x, y: number, text: string) [in Lua]
canvas:fText(x, y: number, text: string) [in Lua]
canvas:wText(x, y: number, text: string) (WC) [in Lua]
```

Draws a text in the position `(x,y)` according to the current font and text alignment. It expects an ANSI string. Can have line breaks.

Attributes

```
void cdCanvasFont(cdCanvas* canvas, const char* typeface, int style, int size); [in C]
void wdCanvasFont(cdCanvas* canvas, const char* typeface, int style, double size); (WD) [in C]

canvas:Font(typeface, style, size: number) [in Lua]
canvas:wFont(typeface, style, size: number) (WD) [in Lua]
```

Selects a text font. The font type can be one of the standard type faces or other driver dependent type face. Since font face names are not a standard between drivers, a few names are specially handled to improve application portability. If you want to use names that work for all systems we recommend using: "**Courier**", "**Times**" and "**Helvetica**".

The style can be a combination of: **CD_PLAIN**, **CD_BOLD**, **CD_ITALIC**, **CD_UNDERLINE** or **CD_STRIKEOUT**. Only the Windows and PDF drivers support underline and strikeout. The size is provided in points (1/72 inch) or in pixels (using negative values).

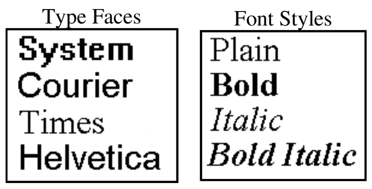
Default values: "**System**", **CD_PLAIN**, 12.

You can specify partial parameters using NULL, -1 and 0 for typeface, style and size. When these parameters are specified the current font parameter is used. For example: **CanvasFont(NULL, -1, 10)** will only change the font size.

To convert between pixels and points use the function **cdPixel2MM** to convert from pixels to millimeters and use the formula "**(value in *points*) = CD_MM2PT * (value in millimeters)**".

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

Fonts can heavily benefit from the ANTIALIAS attribute where available in the driver.



```
void cdCanvasGetFont(cdCanvas* canvas, char* typeface, int *style, int *size); [in C]
void wdCanvasGetFont(cdCanvas* canvas, char* typeface, int *style, double *size); (WC) [in C]

canvas:GetFont() -> (typeface: string, style, size: number) [in Lua]
canvas:wGetFont() -> (typeface: string, style, size: number) (WC) [in Lua]
```

Returns the values of the current font. It is not necessary to provide all return pointers; you can provide only the desired values.

In WC, the size is returned in millimeters.

```
char* cdCanvasNativeFont(cdCanvas* canvas, const char* nativefont); [in C]

canvas:NativeFont(font: string) -> (old_font: string) [in Lua]
```

Selects a font based on a string description. The description can depend on the driver and the platform, but a common definition is available for all drivers. It does not need to be stored by the application, as it is internally replicated by the library. The string is case sensitive. It returns the previous string.

The string is parsed and the font typeface, style and size are set according to the parsed values, as if **cdCanvasFont** was called. The native font string is cleared when a font is set using **cdCanvasFont**.

The common format definition is similar to the the Pango library Font Description, used by GTK+2. It is defined as having 3 parts: <font family>, <font styles> <font size>. For ex: "Times, Bold 18", or "Arial,Helvetica, Italic Underline -24". The supported styles include: Bold, Italic, Underline and Strikeout. Underline, Strikeout, and negative pixel values are not supported by the standard Pango Font Description. The Pango format include many other definitions not supported by the CD format, they are just ignored.

The IUP "FONT" attribute internal formats are also accepted in all drivers and platforms.

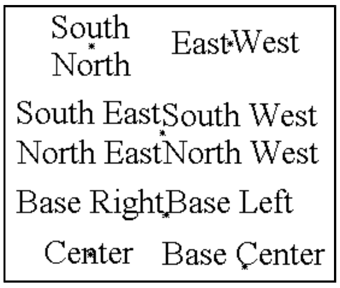
Using "NULL" as a parameter, it only returns the previous string and does not change the font. The value returned is the last attributed value, which may not correspond exactly to the font selected by the driver.

Using "(char*)CD_QUERY" as a parameter, it returns the current selected font in the common format definition.

```
int cdCanvasTextAlignment(cdCanvas* canvas, int alignment); [in C]

canvas: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 cdCanvasTextOrientation(cdCanvas* canvas, double angle); [in C]

canvas:TextOrientation(angle: number) -> (old_angle: number) [in Lua]
```

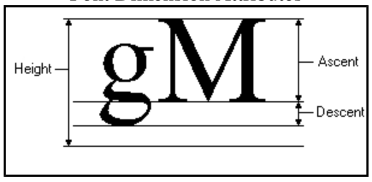
Defines the text orientation, which is an angle provided in degrees relative to the horizontal line according to which the text is drawn. Returns the previous value. Value **CD_QUERY** simply returns the current value. The default value is 0.

Properties

```
void cdCanvasGetFontDim(cdCanvas* canvas, int *max_width, int *height, int *ascent, int *descent); [in C]
void wdCanvasGetFontDim(cdCanvas* canvas, double *max_width, double *height, double *ascent, double *descent); (WC) [in C]

canvas:GetFontDim() -> (max_width, height, ascent, descent: number) [in Lua]
canvas:wGetFontDim() -> (max_width, height, ascent, descent: number) (WC) [in Lua]
```

Returns the maximum width of a character, the line's height, the *ascent* and *descent* of the characters of the currently selected font. The line's height is the sum of the *ascent* and *descent* of a given additional space (if this is the case). All values are given in pixels and are positive. 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 cdCanvasGetTextSize(cdCanvas* canvas, const char* text, int *width, int *height); [in C]
void wdCanvasGetTextSize(cdCanvas* canvas, const char* text, double *width, double *height); (WC) [in C]

canvas:GetTextSize(text: string) -> (width, heigth: number) [in Lua]
canvas:wGetTextSize(text: string) -> (width, heigth: number) (WC) [in Lua]
```

Returns the text size independent from orientation. It is not necessary to provide all return pointers, you can provide only the desired values and NULL for the others.

```
void cdCanvasGetTextBounds(cdCanvas* canvas, int x, int y, const char *text, int *rect); [in C]
void wdCanvasGetTextBounds(cdCanvas* canvas, double x, double y, const char* text, double *rect); (WC) [in C]

canvas:GetTextBounds(x, y: number, text: string) -> (rect: table) [in Lua]
canvas:wGetTextBounds(x, y: number, text: string) -> (rect: table) (WC) [in Lua]
```

Returns the oriented bounding rectangle occupied by a text at a given position. The rectangle has the same dimentions returned by **GetTextSize**. The rectangle corners are returned in counter-clock wise order starting with the bottom left corner, arranged (x0,y0,x1,y1,x2,y2,x3,y3).

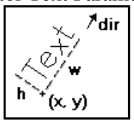
```
void cdCanvasGetTextBox(cdCanvas* canvas, int x, int y, const char* text, int *xmin, int *xmax, int *ymin, int *ymax); [in C]
void wdCanvasGetTextBox(cdCanvas* canvas, double x, double y, const char* text, double *xmin, double *xmax, double *ymin, double *ymax)

canvas:GetTextBox(x, y: number, text: string) -> (xmin, xmax, ymin, ymax: number) [in Lua]
canvas:wGetTextBox(x, y: number, text: string) -> (xmin, xmax, ymin, ymax: number) (WC) [in Lua]
```

Returns the horizontal bounding rectangle occupied by a text at a given position. If orientation is not 0 then its area is always larger than the area of the rectangle returned by **GetTextBounds**. It is not necessary to provide all return pointers, you can provide only the desired values and *NULL* for the others.

Vector Text

It is a text that uses a font created only with line segments. It is very useful to be scaled and very fast. You must set the text size before drawing any text. The default direction is horizontal from left to right.

Vector Text Parameters



All vector text drawing in all drivers are simulated with other CD primitives using polygons only.

```
void cdCanvasVectorText(cdCanvas* canvas, int x, int y, const char* text); [in C]
void wdCanvasVectorText(cdCanvas* canvas, double x, double y, const char* text); (WC) [in C]

canvas:VectorText(x, y: number, text: string) [in Lua]
canvas: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 accepts strings with multiple lines using '\n'. It is ESSENTIAL to call **cdVectorTextSize** or **cdVectorCharSize** before using this function.

The **wdCanvasVectorText** is the only function that actually depends on World Coordinates. The other Vector Text functions although use the "**wd**" prefix they do not depend on World Coordinates. They are kept with these names for backward compatibility. The correct prefix would be "**cdf**".

Attributes

```
void cdCanvasVectorTextDirection(cdCanvas* canvas, int x1, int y1, int x2, int y2); [in C]
void wdCanvasVectorTextDirection(cdCanvas* canvas, double x1, double y1, double x2, double y2); [in C]

canvas:VectorTextDirection(x1, y1, x2, y2: number) [in Lua]
canvas:wVectorTextDirection(x1, y1, x2, y2: number) [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. It is independent from the transformation matrix.

```
double* cdCanvasVectorTextTransform(cdCanvas* canvas, const double* matrix); [in C]

canvas: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 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 scale, 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'|   | scl_x*cos(ang)       -sin(ang)  trans_x |   |x|                     | 3   4   5|
|y'| = |       sin(ang)  scl_y*cos(ang)  trans_y | * |y|      with indices   | 0   1   2|
                                                   |1|
```

It has the same effect of the **cdCanvasTransform,** but notice that the indices are different.

```
void cdCanvasVectorTextSize(cdCanvas* canvas, int width, int height, const char * text); [in C]
void wdCanvasVectorTextSize(cdCanvas* canvas, double width, double height, const char* text); [in C]

canvas:VectorTextSize(width, height: number, text: string) [in Lua]
canvas:wVectorTextSize(width, height: number, text: string) [in Lua]
```

Modifies the font size of the vector text so that it fits the string in the box defined by `width` and `height`.

```
double cdCanvasVectorCharSize(cdCanvas* canvas, int size); [in C]
double wdCanvasVectorCharSize(cdCanvas* canvas, double size); [in C]

canvas:VectorCharSize(size: number) -> (old_size: number) [in Lua]
canvas:wVectorCharSize(size: number) -> (old_size: number) [in Lua]
```

Modifies the font size by specifying the height of the characters. Returns the previous value. `CD_QUERY` returns the current value.

```
void cdCanvasVectorFontSize(cdCanvas* canvas, double size_x, double size_x); [in C]

canvas:VectorFontSize(size_x, size_y: number) [in Lua]
```

Directly modifies the font size. Set size_x==size_y to maintain the original aspect ratio of the font.

```
void cdCanvasGetVectorFontSize(cdCanvas* canvas, double *size_x, double *size_x); [in C]

canvas:GetVectorFontSize() -> (size_x, size_y: number) [in Lua]
```

Returns the font size. It is not necessary to provide all return pointers, you can provide only the desired values and NULL for the others.

```
char* cdCanvasVectorFont(cdCanvas* canvas, const char *filename); [in C]

canvas:VectorFont(filename: string) -> (fontname: 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 current directory, if it fails then it will try the directory defined by the environment variable "`CDDIR`", if it fails, it will attempt to load it using the filename as a string containing the font as if the file was loaded into that string, if it fails again the font is reset to the default font and returns NULL. The file format is compatible with the GKS file format (text mode).

Properties

```
void cdCanvasGetVectorTextSize(cdCanvas* canvas, const char* text, int *width, int *height); [in C]
void wdCanvasGetVectorTextSize(cdCanvas* canvas, const char* text, double *width, double *height); [in C]

canvas:GetVectorTextSize(text: string) -> (width, height: number) [in Lua]
canvas:wGetVectorTextSize(text: string) -> (width, height: number) [in Lua]
```

Returns the text size independent from orientation. It is not necessary to provide all return pointers, you can provide only the desired values and NULL for the others.

```
void cdCanvasGetVectorTextBounds(cdCanvas* canvas, char* text, int x, int y, int *rect); [in C]
void wdCanvasGetVectorTextBounds(cdCanvas* canvas, char* text, double x, double y, double *rect); [in C]

canvas:GetVectorTextBounds(text: string, x, y: number) -> (rect: table) [in Lua]
canvas:wGetVectorTextBounds(text: string, x, y: number) -> (rect: table) [in Lua]
```

Returns the oriented bounding rectangle occupied by a text at a given position. The rectangle has the same dimentions returned by **GetVectorTextSize**. The rectangle corners are returned in counter-clock wise order starting with the bottom left corner, arranged (x0,y0,x1,y1,x2,y2,x3,y3).

```
void cdCanvasGetVectorTextBox(cdCanvas* canvas, int x, int y, const char* text, int *xmin, int *xmax, int *ymin, int *ymax); [in C]
void wdCanvasGetVectorTextBox(cdCanvas* canvas, double x, double y, const char* text, double *xmin, double *xmax, double *ymin, double

canvas:GetVectorTextBox(x, y: number, text: string) -> (xmin, xmax, ymin, ymax: number) [in Lua]
canvas:wGetVectorTextBox(x, y: number, text: string) -> (xmin, xmax, ymin, ymax: number) [in Lua]
```

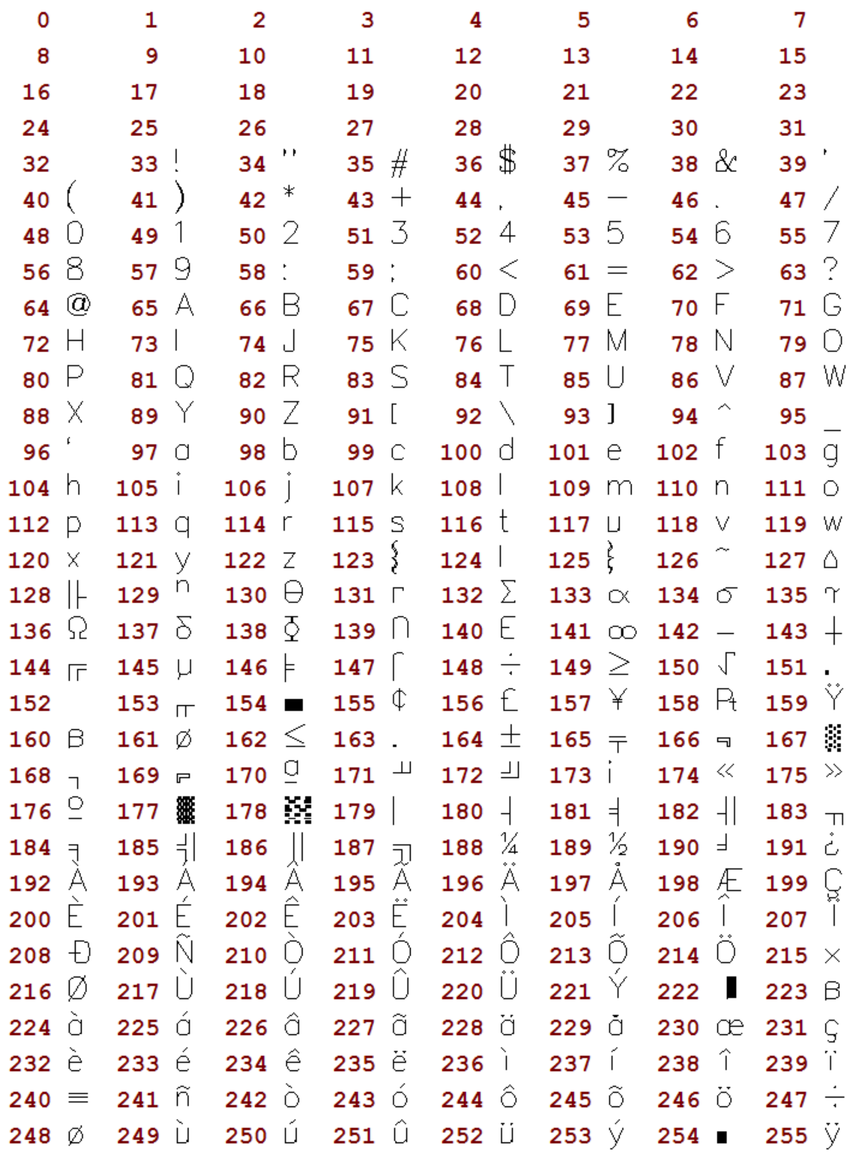
Returns the horizontal bounding rectangle occupied by a text at a given position. If orientation is not 0 then its area is always larger than the area of the rectangle returned by **GetVectorTextBounds**. It is not necessary to provide all return pointers, you can provide only the desired values and *`NULL`* for the others.

Character Codes

The old GKS format contains ASCII codes so a convertion from ANSI to ASCII is done when possible, unmapped characters are left unchanged, but some rearrage was necessary to acomodate the convertion.

The default vector font was changed from the original Simplex II to contain all ANSI accented characters. So some ASCII characters were replaced.

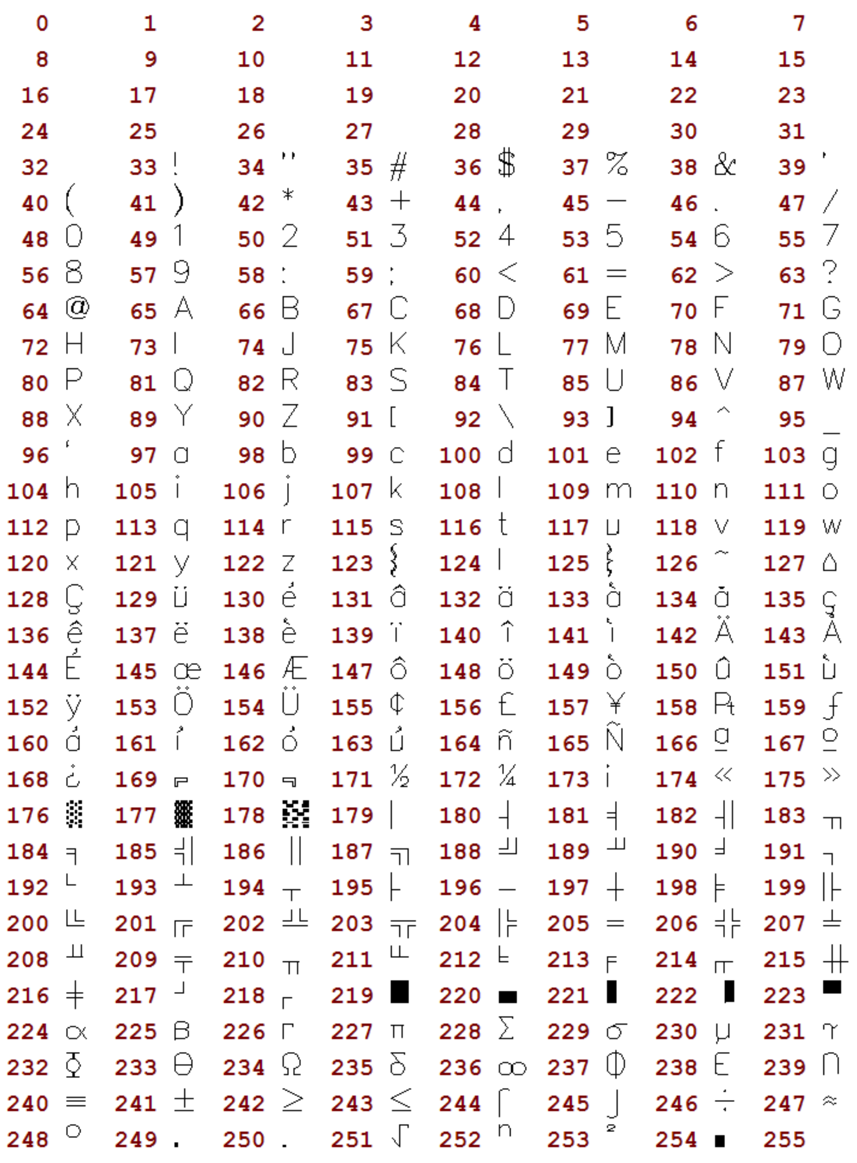
Bellow is the character code table of the default font.



0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33 !	34 "	35 #	36 $	37 %	38 &	39 '
40 (	41)	42 *	43 +	44 ,	45 −	46 .	47 /
48 0	49 1	50 2	51 3	52 4	53 5	54 6	55 7
56 8	57 9	58 :	59 ;	60 <	61 =	62 >	63 ?
64 @	65 A	66 B	67 C	68 D	69 E	70 F	71 G
72 H	73 I	74 J	75 K	76 L	77 M	78 N	79 O
80 P	81 Q	82 R	83 S	84 T	85 U	86 V	87 W
88 X	89 Y	90 Z	91 [	92 \	93]	94 ^	95 _
96 `	97 a	98 b	99 c	100 d	101 e	102 f	103 g
104 h	105 i	106 j	107 k	108 l	109 m	110 n	111 o
112 p	113 q	114 r	115 s	116 t	117 u	118 v	119 w
120 x	121 y	122 z	123 {	124 |	125 }	126 ~	127 Δ
128 ╟	129 ⁿ	130 Θ	131 Γ	132 Σ	133 α	134 σ	135 τ
136 Ω	137 δ	138 Φ	139 ∩	140 ε	141 ∞	142 −	143 ┼
144 ╓	145 µ	146 ╞	147 ⌠	148 ÷	149 ≥	150 √	151 .
152	153 ╥	154 ■	155 ¢	156 £	157 ¥	158 ₧	159 Ÿ
160 ß	161 ø	162 ≤	163 .	164 ±	165 ╤	166 ╕	167 ▒
168 ┐	169 ╒	170 ª	171 ╜	172 ╝	173 ¡	174 «	175 »
176 º	177 ▓	178 ▒	179 │	180 ┤	181 ╡	182 ╢	183 ╖
184 ╕	185 ╣	186 ║	187 ╗	188 ¼	189 ½	190 ╛	191 ¿
192 À	193 Á	194 Â	195 Ã	196 Ä	197 Å	198 Æ	199 Ç
200 È	201 É	202 Ê	203 Ë	204 Ì	205 Í	206 Î	207 Ï
208 Đ	209 Ñ	210 Ò	211 Ó	212 Ô	213 Õ	214 Ö	215 ×
216 Ø	217 Ù	218 Ú	219 Û	220 Ü	221 Ý	222 ▌	223 ß
224 à	225 á	226 â	227 ã	228 ä	229 å	230 œ	231 ç
232 è	233 é	234 ê	235 ë	236 ì	237 í	238 î	239 ï
240 ≡	241 ñ	242 ò	243 ó	244 ô	245 õ	246 ö	247 ÷
248 ø	249 ù	250 ú	251 û	252 ü	253 ý	254 ■	255 ÿ

Default Font

The original Simplex II font is available in the file "cd/etc/vectorfont00.txt". Bellow is the character code table of the original font (the table displays the characters after the convertion from ANSI to ASCII):


Original Simplex II

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 cdCanvasGetImageRGB(cdCanvas* canvas, unsigned char *r,
                   unsigned char *g,
                   unsigned char *b,
                   int x, int y, int w, int h); [in C]

canvas:GetImageRGB(imagergb: cdImageRGB; 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 cdCanvasPutImageRectRGB(cdCanvas* canvas, int iw, int ih,
                       const unsigned char *r,
                       const unsigned char *g,
                       const unsigned char *b,
                       int x, int y, int w, int h,
                       int xmin, int xmax, int ymin, int ymax); [in C]
void wdCanvasPutImageRectRGB(cdCanvas* canvas, int iw, int ih,
                       const unsigned char *r,
                       const unsigned char *g,
                       const unsigned char *b,
                       double x, double y, double w, double h,
                       int xmin, int xmax, int ymin, int ymax); (WC) [in C]
```

```
canvas:PutImageRectRGB(imagergb: cdImageRGB; x, y, w, h, xmin, xmax, ymin, ymax: number) [in Lua]
canvas:wPutImageRectRGB(imagergb: cdImageRGB; 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 cdCanvasPutImageRectRGBA(cdCanvas* canvas, int iw, int ih,
                        const unsigned char *r,
                        const unsigned char *g,
                        const unsigned char *b,
                        const unsigned char *a,
                        int x, int y, int w, int h,
                        int xmin, int xmax, int ymin, int ymax); [in C]
void wdCanvasPutImageRectRGBA(cdCanvas* canvas, int iw, int ih,
                        const unsigned char *r,
                        const unsigned char *g,
                        const unsigned char *b,
                        const unsigned char *a,
                        double x, double y, double w, double h,
                        int xmin, int xmax, int ymin, int ymax); (WC) [in C]

canvas:PutImageRectRGBA(imagergba: cdImageRGBA; x, y, w, h, xmin, xmax, ymin, ymax: number) [in Lua]
canvas:wPutImageRectRGBA(imagergba: cdImageRGBA; 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 **cdPutImageRectRGB** is used, as long as it is defined.

```
void cdCanvasPutImageRectMap(cdCanvas* canvas, int iw, int ih,
                       const unsigned char *index,
                       const long int *colors,
                       int x, int y, int w, int h,
                       int xmin, int xmax, int ymin, int ymax); [in C]
void wdCanvasPutImageRectMap(cdCanvas* canvas, int iw, int ih,
                       const unsigned char *index,
                       const long int *colors,
                       double x, double y, double w, double h,
                       int xmin, int xmax, int ymin, int ymax); (WC) [in C]

canvas:PutImageRectMap(imagemap: cdImageMap; palette: cdPalette; x, y, w, h, xmin, xmax, ymin, ymax: number) [in Lua]
canvas:wPutImageRectMap(imagemap: cdImageMap; palette: cdPalette; 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 iw, int ih,
               const unsigned char *r,
               const unsigned char *g,
               const unsigned char *b,
               unsigned char *index,
               int pal_size, long *color); [in C]

cd.RGB2Map(imagergb: cdImageRGB, imagemap: cdImageMap, palette: cdPalette) [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) -> (bitmap: cdBitmap) [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.

```
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, unsigned char* alpha)
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(bitmap: cdBitmap) [in Lua]
```

Liberates the memory allocated for the image. If this function is not called in Lua, the garbage collector will call it.

```
unsigned char* cdBitmapGetData(cdBitmap* image, int dataptr); [in C]

cd.BitmapGetData(bitmap: cdBitmap; dataptr: number) -> (data: cdImageChannel) [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. cdImageChannel in Lua.
CD_IGREEN - green component of an RGB image. cdImageChannel in Lua.
CD_IBLUE - blue component of an RGB image. cdImageChannel in Lua.
CD_IALPHA - alpha component of an RGBA image. cdImageChannel in Lua.
CD_INDEX - indices of a MAP image. cdImageChannel in Lua.
CD_COLORS - color table of a MAP image. In this case, a type conversion must be made to (long int*).  cdPalette in Lua.
```

In Lua, channels are also available as tables, see Data Access.

```
void cdBitmapSetRect(cdBitmap* image, int xmin, int xmax, int ymin, int ymax); [in C]

cd.BitmapSetRect(bitmap: cdBitmap; 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 cdCanvasPutBitmap(cdCanvas* canvas, cdBitmap* image, int x, int y, int w, int h); [in C]
void wdCanvasPutBitmap(cdCanvas* canvas, cdBitmap* image, double x, double y, double w, double h); (WC) [in C]

canvas:PutBitmap(image: cdBitmap; x, y, w, h: number) [in Lua]
canvas:wPutBitmap(bitmap: cdBitmap; 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 cdCanvasGetBitmap(cdCanvas* canvas, cdBitmap* image, int x, int y); [in C]

canvas:GetBitmap(bitmap: cdBitmap; 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(bitmap_rgb: cdBitmap, bitmap_map: cdBitmap) [in Lua]
```

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

Extras in Lua (Deprecated)

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

Creates an RGB image in Lua. Deprecated use **cd.CreateBitmap**.

```
cd.KillImageRGB(imagergb: cdImageRGB)
```

Destroys the created RGB image and liberates allocated memory. If this function is not called in Lua, the garbage collector will call it. Deprecated use **cd.KillBitmap**.

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

Creates an RGBA image in Lua. Deprecated use **cd.CreateBitmap**.

```
cd.KillImageRGBA(imagergba: cdImageRGBA)
```

Destroys the created RGBA image and liberates allocated memory. If this function is not called in Lua, the garbage collector will call it. Deprecated use **cd.KillBitmap**.

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

Creates a Map image in Lua. Deprecated use **cd.CreateBitmap**.

```
cd.KillImageMap(imagemap: cdImageMap)
```

Destroys the created Map image and liberates allocated memory. If this function is not called in Lua, the garbage collector will call it. Deprecated use **cd.KillBitmap**.

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:

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

The order of the tables *is* important, so that image[n].r has no meaning to CDLua and the expression will cause an error. Finally, the user could expect the value of image[n] to be of type lightuserdata. Unfortunately, this is not the case, and such expression will cause the same error.

In the old cdImageMap images, the channel must be not specified: imagemap[y*100+x].

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 cdPalette object.
```

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. Only the base drivers Win32, GDI+, X-Windows, XRender and Cairo, support server images.

```
cdImage* cdCanvasCreateImage(cdCanvas* canvas, int w, int h); [in C]

canvas:CreateImage(w, h: number) -> (image: cdImage) [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(cdImage* image); [in C]

image:KillImage() [in Lua]
```

Liberates memory allocated for the image. If this function is not called in Lua, the garbage collector will call it.

```
void cdCanvasGetImage(cdCanvas* canvas, cdImage* image, int x, int y); [in C]

canvas:GetImage(image: cdImage; 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 cdCanvasPutImageRect(cdCanvas* canvas, cdImage* image, int x, int y, int xmin, int xmax, int ymin, int ymax); [in C]
void wdCanvasPutImageRect(cdCanvas* canvas, cdImage* image, double x, double y, int xmin, int xmax, int ymin, int ymax); (WC) [in C]

canvas:PutImageRect(image: cdImage; x, y, xmin, xmax, ymin, ymax: number) [in Lua]
canvas:wPutImageRect(image: cdImage; 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 cdCanvasScrollArea(cdCanvas* canvas, int xmin, int xmax, int ymin, int ymax, int dx, int dy); [in C]

canvas: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.

```
char* cdVersionDate(void); [in C]

cd.VersionDate() -> (versiondate: string) [in Lua]
```

Returns the release date of the current version of the library.

```
int cdVersionNumber(void); [in C]

cd.VersionNumber() -> (version: number) [in Lua]
```

Returns the current version number of the library.

```
[in C]
CD_NAME           "CD - Canvas Draw"
CD_DESCRIPTION    "A 2D Graphics Library"
CD_COPYRIGHT      "Copyright (C) 1994-2007 Tecgraf/PUC-Rio and PETROBRAS S/A"
CD_VERSION        "5.0"
CD_VERSION_DATE   "2007/04/09"
CD_VERSION_NUMBER 500000

[in Lua]
cd._NAME
cd._DESCRIPTION
cd._COPYRIGHT
cd._VERSION
cd._VERSION_DATE
cd._VERSION_NUMBER
```

Useful definitions. They have the same value returned by **cdVersion*** functions, except that they do not include the build number.

Metafile Interpretation

```
int cdCanvasPlay(cdCanvas* canvas, cdContext* ctx, int xmin, int xmax, int ymin, int ymax, void *data); [in C]
```

```
canvas: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 given canvas. The primitives are drawn inside the region defined by the given limits. If limits are 0 (xmin, xmax, ymin and ymax) the primitives will be drawn with their coordinates having the original values in the file.

Only some drivers implement this function:

- **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** = CD Metafile, data is a **char*** for the string "*filename*".
- **CD_PICTURE** = CD Picture, data is a **cdCanvas*** of the Picture canvas.

```
int cdContextRegisterCallback(cdContext *ctx, int cb, int(*func)(cdCanvas* canvas, ...)); [in C]
```

```
cd.ContextRegisterCallback(ctx, cb: number, func: function) -> (status: number) [in Lua]
```

Used to customize the behavior of the **Play** function. If you register a known callback function, it will be called during the processing loop of **cdCanvasPlay**. Returns CD_OK if the specified callback is supported or CD_ERROR otherwise.

The callback itself 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(cdCanvas* canvas, int width, int height, double mm_width, double mm_height)
```

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 **cdCanvasGetSize** 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 **Palette** to give the driver an idea of which colors to prioritize. **Palette'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: lightuserdata) [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: lightuserdata) -> (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: lightuserdata, alpha: number) -> (color: lightuserdata) [in Lua]
```

Returns the given color coded with the alpha information. ATTENTION: At the moment only the Cairo, GDI+, XRender and IMAGERGB drivers support alpha components in color coding. The internal representation of the component is inverted, because the default value must be 0 and opaque for backward compatibility, so you should use the **cdDecodeAlpha** function or the **cdAlpha** macro to retrieve the alpha component. 0 is transparent, 255 is opaque.

```
unsigned char cdDecodeAlpha(long int color) [in C]
```

`cd.DecodeAlpha(color: lightuserdata) -> (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 cdAlpha(long int color); [in C]`

`cd.Alpha(color: lightuserdata) -> (r: number) [in Lua]`

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

`unsigned char cdRed(long int color); [in C]`

`cd.Red(color: lightuserdata) -> (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: lightuserdata) -> (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: lightuserdata) -> (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 cdCanvasGetColorPlanes(cdCanvas* canvas); [in C]`

`canvas: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 cdCanvasPalette(cdCanvas* canvas, int n, const long int *color, int mode); [in C]`

`canvas:Palette(palette: cdPalette; 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.

Palette

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

Creates a palette.

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

Destroys the created palette and liberates allocated memory. If this function is not called in Lua, the garbage collector will call it.

Palette Data Access

Data access in Lua is done directly using the array access operators. The colors can have their values checked or changed directly as if they were Lua tables:

```
palette[index] = cd.EncodeColor(r, g, b)
count = #palette
...
color = palette[index]
r, g, b = cd.DecodeColor(color)
```

Notice that the type of value returned or received by palette[index] is a lightuserdata, the same type used with functions **cdEncodeColor**, **cdDecodeColor**, **cdPixel**, **cdForeground** and **cdBackground**.

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. See IUP documentation.

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.

To use this driver, it must be linked with the "**`iupcd`**" library available in the IUP distribution.

In Lua, it is necessary to call function **`cdluaiup_open()`** after a call to function **`cdlua_open()`**, apart from linking with the "**`iupluacd`**" library. This is not necessary if you do require"iupluacd".

Behavior of Functions

This driver is very platform-dependent, although little dependent on the IUP library.

For further detail, see the **Behavior of Functions** in each base driver: GDI, GDK and X-Win. To use this driver with a context plus base driver is necessary to call **`cdUseContextPlus(1)`** before creating the canvas, see the GDI+, Cairo and XRender base drivers.

However, it should be noted that some functions behave differently from the basic functions of each platform.

Control

- **`cdCanvasActivate`**: 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:

GDI and GDI+: 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.
X-Windows: is a string in the format "`display window`" or, in C, "`%p %lu`" (uses the default screen).
GDK and Cairo: is a GdkDrawable* handle.

The given parameters must exists until **`cdKillCanvas`** is called. In Windows, 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 for the same window.

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

Exclusive Functions

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

Equivalent to function **`cdCanvasGetSize`**, 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]
cd.GetScreenColorPlanes() -> (bpp: number) [in Lua]
```

Equivalent to function **`cdCanvasGetColorPlanes`**, 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 very platform-dependent.

For further detail, see the **Behavior of Functions** in each base driver: GDI, GDK and X-Win. To use this driver with a context plus base driver is necessary to call **`cdUseContextPlus(1)`** before creating the canvas, see the GDI+, Cairo and XRender base drivers.

However, it should be noted that some functions behave differently from the basic functions of each platform.

Control

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

  IMPORTANT: For the standard Win32 base driver (not GDI+) if your Windows does not have one of the styles CS_OWNDC or CS_CLASSDC, then a temporary HDC will be created everytime a **cdCanvasActivate** is called. To release this HDC call **cdCanvasDeactivate** after drawing. The IupCanvas control of the IUP library in the Win32 driver have the style, so this should be ignored. But the IupCanvas in the GTK driver running in Win32 does not have this style so **cdCanvasDeactivate** should be used.

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.

GL Driver

This driver represents a driver for drawing using OpenGL. The implementation uses the OpenGL functions only. For the font support, this driver uses the FTGL API functions, which it was written against the Free Type library.

The driver is not dependent of system functions. It uses only the OpenGL portable functions. So if the window canvas changes its size the attribute "SIZE" must be set with the new size or cdCanvasGetSize will return an incorrect value.

Use

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

*`"width`**`x`**`height [resolution]"`*   `in C` **`"%dx%d %g"`**

It must include the initial canvas' dimensions. `Width` and `height` are provided in pixels (note the lowercase "x" between them). The resolution is optional, its default value is "3.78 pixels/mm" (96 DPI).

To use this driver, the application must be linked with the "**cdgl**", the ftgl library and the OpenGL library. The FTGL library is dependent also on the GLU library. In UNIX **cdgl** is also dependent on **iconv**.

In Lua, it is necessary to call function **`cdluagl_open()`** after a call to function **`cdlua_open()`**, apart from linking with the "**`cdluagl`**" library. This is not necessary if you do require"cdluagl".

Behavior of Functions

Control

- **`Play`**: does nothing, returns `CD_ERROR`.

Coordinate System and Clipping

- **`UpdateYAxis`**: does nothing.
- **Clipping**: only support rectangular areas.
- **Complex Regions**: not supported.

Primitives

- **`Begin`**: **`CD_PATH`** is simulated. CD_FILL allows convex polygons only.
- Floating point primitives are supported.

Client Images

- Images are bitmaps, and cannot be directly rotated or scaled.

Attributes

- **`LineStyle`**: in **`CD_CUSTOM`**, style patterns more than 16 bits are not supported.
- **`LineCap`**: does nothing.
- **`LineJoin`**: does nothing.
- **`Stipple`**: does nothing. There is no support for patterns more than 16 bits.
- **`Pattern`**: does nothing.
- **`FillMode`**: does nothing.
- **`NativeFont`**: also accepts the X-Windows font string format.
- **`Font`**: In Windows, "Courier" is mapped to "Courier New", "Helvetica" is mapped to "Arial", and "Times" is mapped to "Times New Roman". In UNIX, "Courier" is mapped to "freemono", "Helvetica" is mapped to "freesans", and "Times" is mapped to "freeserif". Underline and Strikeout are NOT supported. If not found then the font file is searched using the same logic of the Simulation driver, but **ADDFONTMAP** is not supported. If still is not found then the typeface is used as file name for the font.

Colors

- **`Palette`**: does nothing.

Exclusive Attributes

- "**`ALPHA`**": allows the usage of an alpha channel for the drawing shapes. Assumes values "1" (active) and "0" (inactive). Default value: "1".

- "**`ANTIALIAS`**": allows the use of anti-aliasing for the drawing shapes. Assumes values "1" (active) and "0" (inactive). Default value: "1".

- **`"GLVERSION":`** returns a string with the OpenGL version or release number. It is empty if the OpenGL is not available.

- "**`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).

- "**SIZE**": sets the canvas size. Must be used after the window is resized. The format is the same of the data parameter in cdCreateCanvas, *"width**x**height [resolution]"* or in C "**`%dx%d %g`**".

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 and with GDK 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 at the end, it means that a **Bitmap** must be created instead of a metafile, and, if the string `"-m"` is present at the end, a **CD Metafile** will be created. For a **Bitmap** the remaining string must contains the bitmap size and optionally its resolution: *"widthxheight [resolution] -b"* or in C "**`%dx%d %g -b`**", 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 only CD Metafiles are 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. The **cdRegisterCallback** must be called for the driver that will interpret the file, except for bitmaps that the **CD_CLIPBOARD** driver must be used.

Behavior of Functions

This driver is very platform-dependent.

For further detail, see the **Behavior of Functions** in each base driver: GDI, GDK and X-Win. To use this driver with a context plus base driver is necessary to call **`cdUseContextPlus(1)`** before creating the canvas, see the GDI+, Cairo and XRender base drivers.

CD_PRINTER - Printer Driver (cdprint.h)

This driver provides access to a System Default Printer.

The driver works only with the GDI, GDI+ and Cairo base drivers, 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 pre-defined printer dialog 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 **`Flush`** 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 very platform-dependent.

For further detail, see the **Behavior of Functions** in each base driver: GDI. To use this driver with a context plus base driver is necessary to call **`cdUseContextPlus(1)`** before creating the canvas, see the GDI+ and Cairo base drivers.

However, it should be noted that some functions behave differently from the basic functions of each platform.

A printer created in Windows has the same limitations as the EMF driver.

Control

- **`Flush`**: 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

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

Exclusive Attributes

- **`"PRINTERNAME"`**: Returns the name of the selected printer.

Notes

Patterns

Usually when printing regions filled with patterns you have to compensate for the printer high resolution or the pattern will come out very small. If you don't want to create a high resolution version of your pattern, then the simplest way is to use **wdCanvasPattern** to resize the pattern to an expected millimeter size. This will increase the pattern raster size so it will be more visible in the printer.

But on some printers the result were not what we expect:

- Laser printers automatically increase the real size of the pattern, so it seems that **wdCanvasPattern** is not necessary on those printers.
- PDF and Postscript based printer drivers (like Adobe PDF Creator and CutePDF Writer) need that the pattern has a width multiple of 8, if not they will appear distorted with an increasing horizontal shift on every line. This does NOT applies to the CD_PDF or CD_PS drivers.

CD_PICTURE - CD Picture (cdpicture.h)

This driver allows the creation of a CD Picture. It store primitives and attributes in memory that can be played and resized in any other driver. It does not includes clipping and WriteMode.

Use

The file is created by calling function **`cdCreateCanvas`**`(CD_PICTURE, Data)`. The `Data` parameter is a string that can contain the resolution in the following format:

`"[resolution]" or in C use "%lg"`

`Resolution` is the number of pixels per millimeter; its default value is "3.78 pixels/mm" (96 DPI).

The canvas size is automatically calculated to be the bounding box of all the primitives inside the picture.

Any amount of such canvases may exist simultaneously. It is important to note that a call to function **`cdKillCanvas`** is required to release the picture memory.

Behavior of Functions

Coordinate System and Clipping

- **`Play`**: implemented.
- **`UpdateYAxis`**: does nothing.
- **Clipping:** not supported.
- **Transformation Matrix**: not supported.
- cdGetCanvasSize: returns the size of the bounding box that includes all primitives inside the picture.

Attributes

- **`WriteMode`**: does nothing.
- **`FontDim`**: uses a size estimator, returning approximate values.
- **`TextSize`**: uses a size estimator, returning approximate values.

Colors

- **`GetColorPlanes`**: always returns 24.

Primitives

- Floating point primitives are supported.

Client Images

- **`GetImageRGB`**: does nothing.

Server Images

- All functions do nothing.

CD_IMAGERGB - RGB Client Image Driver (cdirgb.h)

This driver allows access to a Client Image, an imaged based in RGB colors with 24 or 32 bits per pixel (8 per channel). It is used to implement high-quality off-screen 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]"      in C "%dx%d %p %p %p -r%g"
or
"widthxheight [r g b a] -r[resolution] -a"    in C "%dx%d %p %p %p %p -r%g -a"
```

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, [r g b] or [r g b a] are pointers to the component buffer, just like PutImageRectRGB/A. The resolution can be defined with parameter `-r`; its default value is "3.78 pixels/mm" (96 DPI).

When the parameter -a is specified an alpha channel will be added to the canvas underlying image. All primitives will be composed using an over operator if the foreground or background colors have alpha components. This channel is initialized with transparent (0). The other channels are initialized with white (255, 255, 255). After drawing in the RGBA image the resulting alpha channel can be used to compose the image in another canvas.

All channels are initialized only when allocated internally by the driver. They are not initialized when allocated by the application.

Any amount of such canvases may exist simultaneously. It is important to note that a call to function **cdKillCanvas** is required to release internal allocated memory.

In Lua, 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 instead of the string, created by functions **cd.CreateImageRGB**, **cd.CreateImageRGBA** or **cd.CreateBitmap** in Lua. The resolution must be passed in a second parameter after the image.

Exclusive Functions

```
cd.ImageRGB(canvas: cdCanvas) -> (imagergb: cdImageRGB or cdImageRGBA) [in Lua]
cd.ImageRGBBitmap(canvas: cdCanvas) -> (bitmap: cdBitmap) [in Lua]
```

Returns the canvas' internal image.

Behavior of Functions

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

Control

- **`Flush`**: does nothing.
- **`Play`**: does nothing, returns `CD_ERROR`.

Coordinate System and Clipping

- **UpdateYAxis**: does nothing. The axis orientation is the same as the CD library's.

Attributes

- **WriteMode**: if alpha transparency is used in colors or images, then XOR or NOT_XOR behave as REPLACE.

Colors

- **GetColorPlanes**: returns 24 if no alpha, returns 32 if exists an alpha channel.
- **Palette**: does nothing.
- **Foreground** & **Background**: accepts the transparency information encoded in the color.

Exclusive Attributes

- "**REDIMAGE**", "**GREENIMAGE**", "**BLUEIMAGE**", "**ALPHAIMAGE**": return the respective pointers of the canvas image (read-only). Not accessible in Lua.

- "**ANTIALIAS**": controls the use of anti-aliasing for line primitives. Assumes values "1" (active) and "0" (inactive). Default value: "1".

- "**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). In this driver will change the current transformation matrix, if removed will reset the current transformation matrix.

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 **cd.CreateImage** in Lua.

Behavior of Functions

This driver is very platform-dependent.

For further detail, see the **Behavior of Functions** in each base driver: GDI, GDK and X-Win. To use this driver with a context plus base driver is necessary to call **cdUseContextPlus(1)** before creating the canvas, see the GDI+, Cairo and XRender base drivers.

CD_DBUFFERRGB - Double Buffer Driver using a RGB image (cdirgb.h)

Implements the concept of offscreen drawing. It is based on a Image RGB (the back buffer) and any other canvas (the front buffer).

Use

The canvas is created by means of a call to function **cdCreateCanvas**`(CD_DBUFFERRGB, Data)`, after which other functions in the CD library can be called as usual. This function creates a CD canvas to use with any existing canvas. 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** for this driver before calling **cdKillCanvas** for the client canvas driver.

The drawing functions will work normally as if they were drawn on the image RGB driver. When function **cdCanvasFlush** is executed, the image is drawn in the canvas passed as parameter in the canvas creation.

When the window's size changes, the RGB image is automatically recreated using the same size as the canvas. This is done in the function **cdCanvasActivate**.

Behavior of Functions

This driver depends on the RGB Client Image Driver.

Control

- **Flush**: draws the contents of the image into the window. It is affected by **Origin** and **Clipping**, but not by **WriteMode**.

CD_DBUFFER - Double Buffer Driver using a server image (cddbuf.h)

Implements the concept of offscreen drawing. It is based on a Server Image (the back buffer) and a Window canvas (the front buffer).

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 to use with 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** for this driver before calling **cdKillCanvas** for the window driver.

The drawing functions will work normally as if they were drawn on the server image driver. When function **cdCanvasFlush** 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 using the same size as the canvas. This is done in the function `cdCanvasActivate`.

We suggest you to implement rubber bands using XOR directly on the front buffer.

Behavior of Functions

This driver is very platform-dependent.

For further detail, see the **Behavior of Functions** in each base driver: GDI, GDK and X-Win. To use this driver with a context plus base driver is necessary to call **`cdUseContextPlus(1)`** before creating the canvas, see the GDI+, Cairo and XRender base drivers.

However, it should be noted that some functions behave differently from the basic functions of each platform.

Control

- `Flush`: draws the contents of the image into the window. It is affected by **Origin** and **Clipping**, but not by **WriteMode**.

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.

By default the pre-compiled library in the distribution uses the PDF Lite version code. The configuration of the PDF Lite code included does not supports image file formats. The current PDF Lite version is 7.0.4p4.

PDFlib Copyright (c) 1997-2009 Thomas Merz and PDFlib GmbH. All rights reserved. Applications that use this driver are subject to the PDFlib GmbH License Agreement.

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.

To use this driver, the application must be linked with the "**cdpdf**" and "**pdflib**" libraries.

In Lua, it is necessary to call function **`cdluapdf_open()`** after a call to function **`cdlua_open()`**, apart from linking with the "**`cdluapdf`**" library. This is not necessary if you do require"cdluapdf".

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

- `Play`: does nothing, returns `CD_ERROR`.
- `Flush`: changes to a new page, preserving the previous one.
- `Clear`: does nothing.

Coordinate System & Clipping

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

Attributes

- **WriteMode**: does nothing, returns CD_REPLACE.
- **Font**: the old "System" font is mapped to the "Courier" font. For the PDF core fonts styles are added to the font name, for other fonts styles are simulated by PDFlib. Underline and Strikeout are supported. Following is the core fonts:

```
Courier, Courier-Bold, Courier-Oblique, Courier-BoldOblique,
Helvetica, Helvetica-Bold, Helvetica-Oblique, Helvetica-BoldOblique,
Times-Roman, Times-Bold, Times-Italic, Times-BoldItalic,
Symbol,
ZapfDingbats
```

Colors

- **GetColorPlanes**: returns 24.
- **Palette**: does nothing.

Client Images

- **GetImageRGB**: does nothing.
- **PutImageMap**: stores an RGB image.

Primitives

- **Pixel**: does not exist in PDF, is simulated using a circle with radius=1.
- Floating point primitives are supported.
- Filled primitives do not include the line at the edges of the filled area.
- When constructing a PATH the CD_PATH_ARC can add only a circular arc. The largest dimension will be used.

Server Images

- All functions do nothing.

Exclusive Attributes

- **"POLYHOLE"**: defines the index of the 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. Default: NULL.

- **"HATCHBOXSIZE"**: defines the size of smallest hatch box pattern. This affects the spacing between the hatch lines. The value passed must be a string containing an integer ("%d"). If the value of the attribute passed is NULL, the value is rest to the default. When consulted returns the current value ("%d"). Default: "8".

- "**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). Can not be set if a transformation is already set.

- "**OPACITY**": allows the usage of a global opacity value. The value passed must be a string containing an integer ("%d") [0=full transparent, 255=full opaque]. Use NULL to reset to the default. Default: 255.

- "**PATTERN**": creates a pattern with regular primitives (except images). The value passed must be a string containing two integeres with the pattern size ("%dx%d") [widthxheight]. Just call regular primitives. Use NULL to end the pattern creation and set the interior style.

- "**PDF**": Returns the "PDF*" handle of the PDFLib.

- "**PDFLIBVERSION**": Returns the full PDFlib version string in the format <major>.<minor>.<revision>.

- **"SUBJECT","TITLE","CREATOR","AUTHOR","KEYWORDS"**: write only attributes that allows to set a description string for the respective document information field.

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

- **`Play`**: does nothing, returns `CD_ERROR`.
- **`Flush`**: changes to a new page, preserving the previous one. Does nothing in EPS mode.
- **`Clear`**: does nothing.

Coordinate System & Clipping

- **`GetCanvasSize`**: returns the page's size within the margins (drawing area).
- **`UpdateYAxis`**: does nothing.
- **Complex Regions**: not supported.
- **Clipping**: not supported in EPS.

Attributes

- **`Background`** does nothing, returns `CD_WHITE`.
- **`BackOpacity`**: does nothing, returns `CD_TRANSPARENT`.
- **`WriteMode`**: does nothing, returns `CD_REPLACE`.
- **`FontDim`**: is simulated.
- **`TextSize`**: is simulated.
- **`Hatch`**: is always opaque (to be implemented).
- **`Stipple`**: is always opaque (to be implemented).
- **`TextAlignment`**: `Baseline` is the same as `South`.
- **`Font`**: old name "System" is mapped to "Courier". Styles are added to the Postscript font name.

Colors

- **`GetColorPlanes`**: returns 24.
- **`Palette`**: does nothing.

Client Images

- **`GetImageRGB`**: does nothing.
- **`PutImageMap`**: stores an RGB image in the file (to be implemented).
- **`PutImageRGBA`**: alpha is ignored (to be implemented).

Primitives

- **`Pixel`**: does not exist in PS, is simulated using a circle with radius=1.
- Floating point primitives are supported.
- Filled primitives do not include the line at the edges of the filled area.

Server Images

- All functions do nothing.

Exclusive Attributes

- "**`POLYHOLE`**": defines the index of the 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). Can not be set if a transformation is already set.

CD_SVG - Scalable Vector Graphics Driver (cdsvg.h)

This driver allows the generation of a SVG file, a modularized language for describing two-dimensional vector and mixed vector/raster graphics in XML. The SVG specification is an open standard that has been under development by the World Wide Web Consortium (W3C) since 1999.

Use

The file is created by calling function **`cdCreateCanvas`**`(CD_SVG, 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 "%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 "3.78 pixels/mm" (96 DPI). `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.

IMPORTANT: because the SVG specification states that floating point number must use dots "." for floating point separators, we set the numeric locale to "C" when the canvas is created, and restore it when it is destroyed. But since it uses the global C function **setlocale** if you use other C functions that are locale dependent while the SVG canvas is being used then be aware that they will be affected.

Behavior of Functions

Control

- **`Play`**: does nothing, returns `CD_ERROR`.
- **`Clear`**: does nothing.

Coordinate System and Clipping

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

Attributes

- **`FontDim`**: is simulated.
- **`TextSize`**: is simulated.
- **`WriteMode`**: does nothing, returns `CD_REPLACE`.

Colors

- **`GetColorPlanes`**: always returns 24.
- **`Palette`**: does nothing.

Primitives

- **`Pixel`**: does not exist in SVG, is simulated using a circle with radius=0.1.
- Floating point primitives are supported.

Client Images

- **`GetImageRGB`**: does nothing.

Server Images

- All functions do nothing.

Exclusive Attributes

- "**`CMD`**": saves a string directly to the file. Allows adding SVG commands to the file generated by the CD library. (set only)

- "**`OPACITY`**": allows the usage of a global opacity value. The value passed must be a string containing an integer ("%d") [0=full transparent, 255=full opaque]. Use NULL to reset to the default. Default: 255.

- **`"HATCHBOXSIZE"`**: defines the size of smallest hatch box pattern. This affects the spacing between the hatch lines. The value passed must be a string containing an integer ("%d"). If the value of the attribute passed is NULL, the value is rest to the default. When consulted returns the current value ("%d"). Default: "8".

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 "3.78 pixels/mm" (96 DPI). `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

- `Play`: implemented.
- `UpdateYAxis`: does nothing.
- **Complex Regions**: not supported.
- `Clear`: removes all primitives from the picture.

Attributes

- `FontDim`: uses a size estimator, returning approximate values.
- `TextSize`: uses a size estimator, returning approximate values.

Colors

- `GetColorPlanes`: always returns 24.

Primitives

- Floating point primitives are supported.

Client Images

- `GetImageRGB`: does nothing.

Server Images

- All functions do nothing.

CD_DEBUG - CD Debug Driver (cddebug.h)

This driver creates a text file with a log of all function calls. But for only the functions that have a driver implementation and in the order that the driver implements sequece of functions like Begin/Vertex/End. Pointers are stored as addresses, and definitions are stored as the CD definition "CD_XXX".

Use

The file is created by calling function `cdCreateCanvas(CD_DEBUG, 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 "3.78 pixels/mm" (96 DPI). `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.

Behavior of Functions

Coordinate System and Clipping

- `Play`: NOT implemented.
- `UpdateYAxis`: does nothing.

Attributes

- `FontDim`: uses a size estimator, returning approximate values.
- `TextSize`: uses a size estimator, returning approximate values.

Colors

- `GetColorPlanes`: always returns 24.

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), and their default value in pixels is `INT_MAX` for both dimensions. 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 "3.78 pixels/mm" (96 DPI). `Width`, `height` and `resolution` are real values. `Width`, `height` and `resolution` are used only by **`cdGetCanvasSize`** and in pixel-millimeter conversion. Parameter `-t` modifies the codification. 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

- **Clear**: does nothing.
- **Flush**: creates a new image, preserving the previous one. The CGM format supports multiple images in a file.
- **Play**: 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

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

Primitives

- **Begin**: if parameter **CD_CLIP** is specified, does nothing. **CD_BEZIER** and **CD_PATH** are simulated with lines.
- **Pixel**: does not exist in CGM, is simulated using a mark with size 1.
- **Chord**: does nothing.
- Floating point primitives are supported.

Attributes

- **WriteMode**: does nothing, returns `CD_REPLACE`.
- **FontDim**: is simulated.
- **FillMode**: does nothing.
- **LineCap**: does nothing.
- **LineJoin**: does nothing.
- **TextSize**: is simulated.
- **TextOrientation**: does nothing.
- **Font**: see the table bellow for the generated font names. No other fonts are supported.

Font Mapping

CD Fonts	Generated Font Names			
	CD_PLAIN	**CD_BOLD**	**CD_ITALIC**	**CD_BOLD|CD_ITALIC**
"System"	"SYSTEM"	"SYSTEM_BOLD"	"SYSTEM_ITALIC"	"SYSTEM_BOLDITALIC"
"Courier"	"COURIER"	"COURIER_BOLD"	"COURIER_ITALIC"	"COURIER_BOLDITALIC"
"Times"	"TIMES_ROMAN"	"TIMES_ROMAN_BOLD"	"TIMES_ROMAN_ITALIC"	"TIMES_ROMAN_BOLDITALIC"
"Helvetica"	"HELVETICA"	"HELVETICA_BOLD"	"HELVETICA_ITALIC"	"HELVETICA_BOLDITALIC"

Colors

- **GetColorPlanes**: returns 24.
- **Palette**: does nothing.

Client Images

- **GetImageRGB**: does nothing.
- **PutImageRGBA**: 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 "3.78 pixels/mm" (96 DPI). 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 **cdCanvasGetSize** 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

- **Clear**: does nothing.
- **Play**: does nothing, returns CD_ERROR.

Coordinate System and Clipping

- **Clip**: does nothing (no clipping function is supported), returns CD_CLIPOFF.
- **UpdateYAxis**: does nothing. The axis orientation is the same as the CD library.
- **Transformation Matrix**: not supported.

Primitives

- **Begin**: if parameter **CD_CLIP** is specified, does nothing. **CD_BEZIER** and **CD_PATH** are simulated with lines.
- **cdChord**: does nothing.

Attributes

- **BackOpacity**: does nothing, returns CD_OPAQUE.
- **WriteMode**: does nothing, returns CD_REPLACE.
- **InteriorStyle**: does nothing.
- **FillMode**: does nothing.
- **LineCap**: does nothing.
- **LineJoin**: does nothing.
- **Hatch**: does nothing.
- **Stipple**: does nothing.
- **Pattern**: does nothing.
- **TextSize**: returns a bounding box which is usually larger than the text (the computation is based on the widest character).
- **TextAlignment**: uses **cdTextSize**, therefore is not precise.
- **Font**: See the font mapping table for the equivalence used to map CD fonts into MicroStation fonts. Styles are not supported.

Font Mapping

CD Fonts	MicroStation Font Index
System	0
Courier	1
Times	2
Helvetica	3

Colors

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

Client Images

- **GetImageRGB**: does nothing.
- **PutImageRGB**: considering that the format supports only 256 colors, image quality is quite poor.
- **PutImageRGBA**: alpha is ignored.
- **PutImageMap**: 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 "3.78 pixels/mm" (96 DPI). Width, height and resolution are given in real values and are used only by **cdCanvasGetSize** 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 **Flush** to change the current layer.

Behavior of Functions

Control

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

Coordinate System and Clipping

- **Clip**: does nothing (no clipping function is supported), returns CD_CLIPOFF.
- **UpdateYAxis**: does nothing. Axis orientation is the same as in the CD library.
- **Transformation Matrix**: not supported.

Primitives

- **Sector**: draws a "hollow" sector, that is, only its borders.
- **Begin**: if parameter **CD_CLIP** is specified, does nothing. **CD_BEZIER** and **CD_PATH** are simulated with lines.
- **Chord**: does nothing.
- Floating point primitives are supported.
- Solid filled primitives are supported only for polygons and rectangles.

Attributes

- **BackOpacity**: does nothing, returns CD_TRANSPARENT.
- **WriteMode**: does nothing, returns CD_REPLACE.
- **Hatch**: does nothing.
- **FillMode**: does nothing.
- **LineCap**: does nothing.
- **LineJoin**: does nothing.
- **Stipple**: does nothing.
- **Pattern**: does nothing.
- **TextSize**: returns a bounding box usually larger than the text (the computation is based on the widest character).
- **TextOrientation**: does nothing.
- **Font**: 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. No other fonts are supported.

Font Mapping

CD Fonts	AutoCAD Fonts
System	STANDARD (sem arquivo)
Courier	ROMAN (romanc.shx)
Courier + CD_BOLD	ROMAN_BOLD (romant.shx)
Times	ROMANTIC (rom_____.pfb)
Times + CD_BOLD	ROMANTIC_BOLD (romb_____.pfb)
Helvetica	SANSSERIF (sas_____.pfb)
Helvetica + CD_BOLD	SANSSERIF_BOLD (sasb____.pfb)

Colors

- **Foreground**: indexes long int *color in the fixed palette (AutoCAD uses a 256-color palette - for further detail, see AutoCAD's Reference Manual).
- **Background**: does nothing, returns CD_WHITE.
- **GetColorPlanes**: returns 8.
- **Palette**: 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 with the GDI, GDI+ and Cairo base drivers, 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 very platform-dependent.

For further detail, see the **Behavior of Functions** in each base driver: GDI. To use this driver with a context plus base driver is necessary to call **`cdUseContextPlus (1)`** before creating the canvas, see the GDI+ and Cairo base drivers.

However, it should be noted that some functions behave differently from the basic functions of each platform.

It has been noticed that EMFs, when saved in the Windows 9x environment, is not totally compatible with EMFs saved in the Windows NT environment.

If you intend to use **cdCanvasPlay** to interpret the EMF, then do not use GDI+ to generate the metafile. GDI+ extensively use internal transformations that will affect the **cdCanvasPlay** interpretation. Also some interior style will not be correctly interpreted.

Control Functions

- **`Play`**: different from the basic driver, is implemented. Not implemented using GDI+.
- **`Clear`**: different from the basic driver, does nothing.

Client Images

- **`GetImageRGB`**: does nothing.

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.

It is recomended to use EMF instead of WMF whenever is possible.

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.

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

- `Play`: different from the basic driver, is implemented.
- `Clear`: different from the basic driver, does nothing.

Coordinate System and Clipping

- `Clip`: does nothing, returns `CD_CLIPOFF`.

Attributes

- `Stipple`: is always opaque and smaller than 8x8 pixels.
- `Pattern`: does nothing.
- `LineWidth`: is always 1.
- `TextAlignment`: `CD_CENTER/CD_WEST/CD_EAST` is saved as `CD_BASE_CENTER/CD_BASE_LEFT/CD_BASE_RIGHT`, but the position error is compensated.
- `TextOrientation`: does nothing

Client Images

- `GetImageRGB`: does nothing.
- `PutImageRGBA`: the alpha component is ignored.

Server Images

- All functions do nothing.

GDK Base Driver

This driver represents a basic driver for all system-dependent drivers implemented in the X-Windows system. The implementation uses the GDK and Pango API functions. This driver was designed for the GTK+ version 2, and can be compiled and used in all systems GDK is supported.

This is included in the main library in Linux and BSD. To use it in other Unices and in Win32 you must link with the **cdgdk** library instead of the **cd** main library.

Behavior of Functions

Control

- `Play`: does nothing, returns `CD_ERROR`.

Coordinate System and Clipping

- `UpdateYAxis`: the orientation of axis Y is the opposite to its orientation in the CD library.

Primitives

- `Begin`: **`CD_BEZIER`** and **`CD_PATH`** are simulated with lines.

Attributes

- `LineWidth`: if `width` is 1, the driver will use 0 for a better performance.
- `LineStyle`: thick lines have style only in the line's direction. For example, you will see small rectangles in a thick dotted line.
- `NativeFont`: also accepts the X-Windows font string format.
- `Font`: "Courier" is mapped to "Monospace", "Helvetica" is mapped to "Sans", and "Times" is mapped to "Serif". Underline and Strikeout are supported.

Colors

- `Palette`: 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. 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.

Exclusive Attributes

- "**`GC`**": returns the GDK graphics context (get only). In Lua is returned as an user data.

- "**`IMGDITHER`**": changes how dithering is used in images when bpp<=8. Can be "NORMAL" or "NONE". Default: "NONE".

- "**`IMGINTERP`**": changes how interpolation is used in image scale. Can be "BILINEAR" or "NEAREST". Default: "NEAREST".

- "**`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). In this driver will change the current transformation matrix, if removed will reset the current transformation matrix.

- **`"PANGOVERSION":`** returns a string with the Pango version number. It is empty if the Pango is not available.

GDI 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

- **Flush**: does nothing.
- **Play**: does nothing, returns `CD_ERROR`.

Coordinate System and Clipping

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

Primitives

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

Attributes

- **WriteMode**: for the client and server image functions, the mode **NOT_XOR** works as **XOR**.
- **Stipple**: 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.
- **Pattern**: 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.
- **TextAlignment**: the vertical alignment of CD_CENTER, CD_EAST, CD_WEST is manually calculated.
- **LineWidth**: If not in Windows NT line width is always 1. If line width is 1, then a cosmetic pen is used for fast drawing.
- **LineStyle**: 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.
- **NativeFont**: also accepts ***"-d"*** to show the font-selection dialog box.
- **Font**: "Courier" is mapped to "Courier New", "Helvetica" is mapped to "Arial", and "Times" is mapped to "Times New Roman". Underline and Strikeout are supported. The System font does not have orientation.

Client Images

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

Colors

- **Palette**: 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.

- "**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 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** and for every **cdPutImageRect**, 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.

- "**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.

- "**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.

- "**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). Can not be set if a transformation is already set.

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.

This is included in the main library in AIX, SunOS and IRIX. To use it in Linux and BSD you must link with the **cdx11** library instead of the **cd** main library.

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

- **Play**: does nothing, returns `CD_ERROR`.

Coordinate System and Clipping

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

Primitives

- `Text`: text orientation is simulated using XVertex rotines. Generic transformation matrix affects only the position of the text.
- `Begin`: Filled polygons have an error of one pixel to the right and below. `CD_BEZIER` and `CD_PATH` are simulated with lines.
- `Box`: in Linux with ATI board, is being drawn with one extra pixel to the right and below.

Attributes

- `LineWidth`: if `width` is 1, the driver will use 0 for a better performance.
- `LineStyle`: thick lines have style only in the line's direction. For example, you will see small rectangles in a thick dotted line.
- `NativeFont`: also accepts the X-Windows font string format. You can use program **xfontsel** to select a font and obtain the string. For ex: "-*-times-bold-r-*-*-24-*-*-*-*-*-*-*" (equivalent of **Font**("Times", CD_BOLD, -24).
- `Font`: the old name "System" is mapped to "fixed".

Colors

- `Palette`: 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

- `GetImageRGB`: 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.

- "`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). In this driver will change the current transformation matrix, if removed will reset the current transformation matrix.

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, 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.

In many drivers, the behavior of a given primitive may not be the expected. Usually this is documented in the manual. If 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

Clipping

- Clipping is not implemented in the simulation base driver. The primary driver must implement its own clipping.

Attributes

- `LineCap`: only `CD_CAPFLAT` is supported.
- `LineJoin`: only `CD_MITER` is supported.
- `LineStyle`: If line width is greater than 1, the style is always continuous.
- `Font`: Selects a True Type font file for the FreeType library to render the text. Notice that TTF fonts have different files 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.
  Old name "System" is mapped to "Courier". For the know font names "Courier" (`cour`), "Times" (`times`) and "Helvetica" (`arial`), the styles are added to the font file name as a suffix: "bd", "i" and "bi" are used for bold, italic and bold-italic. For other fonts, it will first check for a font map added using the attribute **ADDFONTMAP**, if failed it will try to load the type_face name without any change, if fail it will add the style suffix to the type_face and try to load again. The ".ttf" file extension is always automatically added to the end of the file name.

Primitives

- `Pixel`: always uses the client's pixel function. When clipping simulation is active, it executes area and polygon clipping.
- `Line`: draws lines pixel per pixel.
- `Rect`: simulated using the client's **Line**.
- `Arc`: simulated using the client's **Line**.
- `Sector`: simulated using the client's **Poly**.
- `Chord`: simulated using the client's **Poly**
- `Box`: simulated using the client's **Poly**.
- `Begin`, `Vertex` and `End`: simulated using the **Line** or **Pixel** functions, depending on the interior style.
- `Text`: text simulation is made using TrueType font files in a transparent way for the user. Oriented text is not supported.

Exclusive Attributes

- "**ADDFONTMAP**": Add a font map between a type face name and a file name. It has the format "Type Face=filename", For ex: "Arial Narrow Bold=ARIALNB". "Type Face" is not case sensitive.

- "**FREETYPEVERSION**": Returns a version string with the current Freetype library being used, in the format "FreeType <major>.<minor>.<patch>".

CAIRO Base Driver

This driver represents a basic driver for all system-dependent drivers implemented in the X-Windows and MS-Windows systems. The implementation uses the Cairo and Pango functions. This driver can be compiled and used in all systems Cairo is supported. The drivers **Native Window**, **Image**, **EMF**, **Printer** and **Double Buffer** were implemented.

It can be used as the context plus driver of the GDI, GDK and X-Win based drivers. But its primary focus is the GDK base driver.

The main motivation for the use of Cairo was transparency for all the primitives. Beyond that we got other features like anti-aliasing, gradient filling, transformations and other back-ends (support to rendering: PDF, PS, SVG and IMAGERGB surfaces).

This driver still does not completely replace the X-Windows, GDK and GDI Windows base drivers, because Cairo does not have support for bitwise XOR operations and for complex clipping regions.

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 Cairo for the available GDK, Win32 or X-Win based drivers. This function affects only the **`cdCreateCanvas`** function call, once created the canvas will be always a Cairo canvas. In fact the function affects primary the definitions **`CD_NATIVEWINDOW`**, **`CD_IMAGE`**, **`CD_EMF`**, **`CD_PRINTER`** and **`CD_DBUFFER`**, because they are function calls and not static defines. **`CD_PRINTER`** can be used with the GDK base driver in UNIX, or with the Win32 base driver in Windows. **`CD_EMF`** can be used in Windows only, with GDK or Win32 base drivers.

Using Cairo 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 GDK, Win32 or X-Windows canvas.

To enable the use of Cairo based drivers you must call the initialization function **`cdInitContextPlus`** once, and do not need to link with any additional library when using the GDK base driver. But when using with the GDI and X-Win base drivers you need to link to the libraries "**cdcairo**" and "**cairo**".

Also the Cairo library must be installed in your system.

In CDLua it is not necessary any additional initialization, and **require"cdluacontextplus"** can be used when using dynamic libraries. But it is available only in Linux and only for the GDK base driver.

As an alternative you can use **require**"**cdluacairo"**, but there are some restrictions: in Windows only the GDI base driver can be used; in Linux only the GDK base driver can be used; in other UNICES only the X-Win base driver can be used.

Extra Drivers (cdcairo.h)

Only available in Lua when **require"cdluacairo"** is used.

CD_CAIRO_PS - PostScript Driver

Similar to CD_PS, uses the same creation parameters. But margins are not supported and Postscript level can be 2 (parameter -2) or 3 (parameter -3). The "CMD" attribute is not supported, and the new attribute "DSCCOMMENT" accepts a string that is saved as a DSC comment.

CD_CAIRO_PDF - PDF Driver

Similar to CD_PDF, uses the same creation parameters. The driver also does not depends on the PDFLib. The additional attributes "OPACITY", "PATTERN", "PDF", "PDFLIBVERSION" and the description strings, are not supported.

CD_CAIRO_SVG - Scalable Vector Graphics Driver

Similar to CD_SVG, uses the same creation parameters. The additional attributes "OPACITY" and "CMD" are not supported.

CD_CAIRO_IMAGERGB - RGB Client Image Driver

Similar to CD_IMAGERGB, uses almost the same creation parameters. The main difference is that the data pointers are packed in RGBARGBARGBA... format. So it is used only 1 pointer for data, instead of 3. Also the attributes "REDIMAGE", "GREENIMAGE", "BLUEIMAGE" and "ALPHAIMAGE are not supported and replaced by the "RGBDATA" attribute. There are also two new attributes, "STRIDE" that returns the line size in bytes, when data is specified during creation then stride is always width*32. Even when there is not alpha channel, data is stored in 32 bits per pixel. Image data is also organized in top-bottom orientation, it means the data pointer points to the top-left corner. And the "WRITE2PNG" attribute that accepts a filename to save the image as a PNG file (this does not depends of the IM library).

Behavior of Functions

Control

- **`Play`**: does nothing, returns `CD_ERROR`.

Coordinate System and Clipping

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

Primitives

- Floating point primitives are supported.

Attributes

- **`WriteMode`**: does nothing. There is no support for XOR or NOT_XOR.
- **`NativeFont`**: also accepts the X-Windows font string format.
- **`Font`**: "Courier" is mapped to "Monospace", "Helvetica" is mapped to "Sans", and "Times" is mapped to "Serif". Underline and Strikeout are supported.

Colors

- **`Palette`**: NOT supported.

Exclusive Attributes

- "**`ANTIALIAS`**": controls the use of anti-aliasing for the text and drawing shapes. Assumes values "1" (active) and "0" (inactive). Default value: "1".
- "**`CAIRODC`**": returns the Cairo drawing context (get only). In Lua is returned as a user data.
- **`"CAIROVERSION":`** returns a string with the Cairo version number. It is empty if the Cairo is not available.
- **`"HATCHBOXSIZE"`**: defines the size of smallest hatch box pattern. This affects the spacing between the hatch lines. The value passed must be a string containing an integer ("%d"). If the value of the attribute passed is NULL, the value is rest to the default. When consulted returns the current value ("%d"). Default: "8".
- "**`IMGINTERP`**": changes how interpolation is used in image scale. Can be "BEST" (highest-quality), "BILINEAR" (linear interpolation), "GOOD" (quality similar to BILINEAR), "NEAREST" (nearest-neighbor filtering) or "FAST" (quality similar to NEAREST). Default: "GOOD".
- "**`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).
- "**PATTERNIMAGE**": defines a filled interior style using a server image as pattern. Data must be a server image handle created with he Cairo base driver.
- "**`POLYHOLE`**": defines the index of the 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. Default: NULL.
- "**`RADIALGRADIENT`**": defines a filled interior style that uses a radial gradient between two colors. It uses 2 points and 2 radius ("%d %d %g %d %d %g" = x1 y1 rad1 x2 y2 rad2), one for the starting point using (using the foreground color), and another one for the end point (using the background color).
- "**`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). Can not be set if a transformation is already set.

GDI+ Base Driver

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+.

It can be used only as the context plus driver of the GDI based drivers.

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 and filled cardinal splines.

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 GDI based 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_NATIVEWINDOW`**, **`CD_IMAGE`**, **`CD_PRINTER`**, **`CD_EMF`**, **`CD_DBUFFER`** and **`CD_CLIPBOARD`**, 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 **`cdInitContextPlus`** once and link to the libraries "**cdcontextplus.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. The gdiplus files can be obtained from Microsoft or from here.

In CDLua it is not necessary any additional initialization, but the application must still be linked with the **cdcontextplus.lib** library or a **require"cdluacontextplus"** can be used when using dynamic libraries.

Behavior of Functions

Control

- **`Play`**: does nothing, returns `CD_ERROR`.

Coordinate System and Clipping

- **`UpdateYAxis`**: the orientation of axis Y is the opposite to its orientation in the CD library. Except when using transformations.

Primitives

- Floating point primitives are supported.
- **`Pixel`**: uses GDI. Excepting when the canvas is an image so it is done using GDI+.
- **`Sector`**: it also draws an arc in the same position to complete the size of the sector.
- **`Text`**: opaque text is simulated using a rectangle in the back.
- **`Begin`**: 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

- **`WriteMode`**: does nothing. There is no support for XOR or NOT_XOR.
- **`Pattern`**: each pixel can contain transparency information.
- **`LineStyle`**: uses a custom GDI+ style when line width is 1. In World Coordinates the line style has its scaled changed.
- **`FontDim`**: the maximum width is estimated from the character "W".
- **`TextAlignment`**: is simulated. Although GDI+ has text alignment, the results do not match the CD text alignment.

- `NativeFont`: also accepts ***"-d"*** to show the font-selection dialog box.
- `Font`: "System" is mapped to "MS Sans Serif", "Courier" is mapped to "Courier New", "Helvetica" is mapped to "Arial", and "Times" is mapped to "Times New Roman". Underline and Strikeout are supported.

Colors

- `Palette`: works only when the canvas is a server image.
- `Foreground` & `Background`: accepts the transparency information encoded in the color.

Client Images

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

Server Images

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

Exclusive Attributes

- "**`GDI+`**": returns "1". So the application can detect if the driver uses the GDI+ base driver. Other drivers that do not implement this attribute will return NULL.

- "**`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 for 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 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 in the **`cdCreateImage`** and for every **`cdPutImageRect`**, 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). Can not be set if a transformation is already set.

- **`"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).

XRender Base Driver

This driver represents a basic driver for all system-dependent drivers implemented in the X-Windows system using the XRender extension. The implementation uses the XRender and Xft API functions.

It can be used only as the context plus driver of the X-Win based drivers.

The main motivation for the use of XRender was transparency for all the primitives. Beyond that we got other features like anti-aliasing, gradient filling and transformations.

This driver still does not completely replace the X-Windows base driver, because XRender does not have support for XOR and for line styles.

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 X-Render for the available X-Win based drivers. This function affects only the **`cdCreateCanvas`** function call, once created the canvas will be always a XRender canvas. In fact the function affects primary the definitions **`CD_NATIVEWINDOW`**, **`CD_IMAGE`** and **`CD_DBUFFER`**, because they are function calls and not static defines.

Using XRender 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 X-Windows canvas.

To enable the use of XRender based drivers you must call the initialization function **`cdInitContextPlus`** once and link to the libraries "**cdcontextplus**", "**Xrender**" and "**Xft**". Also the libraries "**Xrender**" and "**Xft**" must be installed in your system. The XRender extension must be available in the X-Windows server for the driver to work.

Currently, pre-compiled binaries are available for Linux and BSD. It is not available for the systems we have with AIX, SunOS and IRIX.

In CDLua it is not necessary any additional initialization, but the application must still be linked with the **cdcontextplus.lib** library or a **require"cdluacontextplus"** can be used when using dynamic libraries. When using require, it is NOT available in Linux.

Behavior of Functions

Control

- **Play**: does nothing, returns `CD_ERROR`.

Coordinate System and Clipping

- **UpdateYAxis**: the orientation of axis Y is the opposite to its orientation in the CD library. Except when using transformations.

Primitives

- **Line**: simulated using the client's **Poly**.
- **Text**: Generic transformation matrix affects only the position of the text. Complex clipping regions can not contain text regions.
- **Begin**: **CD_BEZIER** and **CD_PATH** are simulated with lines.
- **Rect**: simulated using the client's **Line**.
- **Arc**: simulated using the client's **Line**.
- **Sector**: simulated using the client's **Poly**.
- **Chord**: simulated using the client's **Poly**
- **Box**: simulated using the client's **Poly**.
- Floating point primitives are supported.

Attributes

- **LineWidth**: the driver will use a polygon that fits to the line extents, even when linewidth==1.
- **LineStyle**: NOT supported.
- **Pattern**: each pixel can contain transparency information.
- **NativeFont**: also accepts the X-Windows font string format. You can use program **xfontsel** to select a font and obtain the string. For ex: "-*-times-bold-r-*-*-24-*-*-*-*-*-*-*" (equivalent of **Font**("Times", CD_BOLD, -24).
- **Font**: font support is implemented using the Xft library. Internally the Xft library uses the Freetype library.

Colors

- Use the X-Windows base driver support for colors.

Client and Server Images

- All functions use the X-Windows base driver functions.

Exclusive Attributes

- "**GC**": returns the X11 graphics context (get only). In Lua is returned as a user data.

- "**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). In this driver will change the current transformation matrix, if removed will reset the current transformation matrix.

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

- **"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). (available only if Xrender version >= 0.10)

- **"XRENDERVERSION":** returns a string with the XRender version number. It is empty if the XRender extension is not available in the X-Windows server.