A Font Library VFlib

VFlib version 3.6.14 User's manual
Final Revision: 26 Feb 2006

Hirotsugu Kakugawa

1 Copyright

2 Copying

[This is the first released version of the library GPL. It is
numbered 2 because it goes with version 2 of the ordinary GPL.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users.

This license, the Library General Public License, applies to some specially designated Free Software Foundation software, and to any other libraries whose authors decide to use it. You can use it for your libraries, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library, or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link a program with the library, you must provide complete object files to the recipients so that they can relink them with the library, after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

Our method of protecting your rights has two steps: (1) copyright the library, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the library.

Also, for each distributor's protection, we want to make certain that everyone understands that there is no warranty for this free library. If the library is modified by someone else and passed on, we want its recipients to know that what they have is not the original version, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that companies distributing free software will individually obtain patent licenses, thus in effect transforming the program into proprietary software. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License, which was designed for utility programs. This license, the GNU Library General Public License, applies to certain designated libraries. This license is quite different from the ordinary one; be sure to read it in full, and don't assume that anything in it is the same as in the ordinary license.

The reason we have a separate public license for some libraries is that they blur the distinction we usually make between modifying or adding to a program and simply using it. Linking a program with a library, without changing the library, is in some sense simply using the library, and is analogous to running a utility program or application program. However, in a textual and legal sense, the linked executable is a combined work, a derivative of the original library, and the ordinary General Public License treats it as such.

Because of this blurred distinction, using the ordinary General Public License for libraries did not effectively promote software sharing, because most developers did not use the libraries. We concluded that weaker conditions might promote sharing better.

However, unrestricted linking of non-free programs would deprive the users of those programs of all benefit from the free status of the libraries themselves. This Library General Public License is intended to permit developers of non-free programs to use free libraries, while preserving your freedom as a user of such programs to change the free libraries that are incorporated in them. (We have not seen how to achieve this as regards changes in header files, but we have achieved it as regards changes in the actual functions of the Library.) The hope is that this will lead to faster development of free libraries.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, while the latter only works together with the library.

Note that it is possible for a library to be covered by the ordinary General Public License rather than by this special one.

GNU LIBRARY GENERAL PUBLIC LICENSE

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

1. This License Agreement applies to any software library which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Library General Public License (also called "this License"). Each licensee is addressed as "you".

   A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

   The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

   "Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

2. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on

each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

3. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
   a. The modified work must itself be a software library.
   b. You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
   c. You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
   d. If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

      (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

   In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

4. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

   Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

   This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

5. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that

you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

6. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

   However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

   When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

   If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

   Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

7. As an exception to the Sections above, you may also compile or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

   You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

   a. Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)
   b. Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.
   c. If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

   d. Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

   For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

8. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:
   1. Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.
   2. Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.
9. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
10. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.
11. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
12. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

13. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

14. The Free Software Foundation may publish revised and/or new versions of the Library General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

15. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

16. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

17. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH

ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the library's name and a brief idea of what it does.
Copyright (C) year  name of author

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Library General Public
License as published by the Free Software Foundation; either
version 2 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Library General Public License for more details.

You should have received a copy of the GNU Library General Public
License along with this library; if not, write to the Free
Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
```

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the
library 'Frob' (a library for tweaking knobs) written by James Random Hacker.

signature of Ty Coon, 1 April 1990
Ty Coon, President of Vice
```

That's all there is to it!

3 Introduction

Today many font files are available in various font file formats. When we need a software to display or print characters which does not depend on a windowing system and/or an operating system, we must write interface routines for accessing font files in each application software again and again. To do this, programmers must have knowledge on font file formats; it will be a hard task for programmers if the number of font formats that an application software supports becomes large.

VFlib is a font library written in C providing several functions to obtain bitmaps of characters. VFlib hides the font format of font files and provides a unified API for all supported font formats. Thus, programmers for application software need not have knowledge on font file formats. Instead, any software using VFlib can support various font file formats immediately.

This document describes the fundamental concepts of VFlib and gives a brief introduction in writing programs using VFlib.

As described above, VFlib supports many font file formats and absorbs differences between font file formats. Currently, VFlib supports the following font file formats: PCF, BDF, HBF, TrueType, Type1, GF, PK, Virtual Fonts, TFM, SyotaiKurabu (a vector font format for Japanese Kanji characters), JG (another vector font format for Japanese Kanji characters), and eKanji (a bitmap font format for Kanji characters).

The VFlib system consists of two parts:

A library (‘`libVFlib.a`’ and/or ‘`libVFlib.so`’)
: It provides several C functions. Any application software using VFlib must link this library.

  VFlib (optionally) uses kpathsea, FreeType, and T1Lib libraries. Application software linked with VFlib must be linked with these external libraries, if you want to use them. Application software must be linked against kpathsea, FreeType and T1Lib in addition to VFlib if you configured VFlib to use them.

A font database file (‘`vflibcap`’)
: When we open a font, information about the font file (font format, location, possibly necessary glyph modifications etc.) is necessary. This file describes such information; it is read when the initialization function of VFlib is called.

Basic concept of VFlib

Font Classes and Font Drivers
: VFlib can handle multiple font file formats. Reading a font file according to its font file format is done by an internal module in VFlib corresponding to its font file format. This internal module is called a *font driver*. Service units provided by a font driver is called *font class*. From an end-user's point of view, various font formats are distinguished by various names of font classes. Font drivers themselves are internal of VFlib and invisible for end-users.

  Some font drivers may not read font files on disk; they may generate glyph and outlines by internal computation only. In addition, some font drivers may return glyph which are obtained as glyph by another font class (hence the name ‘Virtual Font library’).

A View of VFlib Font From The End-User
: Each (virtual) font by VFlib has its inherent information of point size, pixel size, and resolution of the target device. In addition to these font metrics are defined for each glyph.

Some font file formats does not have such concepts; in such case, (1) lacking information is given in a font database file '`vflibcap`' or (2) the specific font driver gives such information as default values. For instance, a TrueType font file is a vector font file and does not has information on point size and resolution of the target device (such information is unnecessary since vector fonts can be scaled at any size). SyotaiKurabu font format (a vector font for Japanese Kanji) does not have font metric information at all. A font driver for this font format virtually generates font metrics by information given in a vflibcap file.

Font Names and Font Searching Mechanism

In VFlib, a font is specified by a font name when a font is requested to open. First, VFlib checks whether the font name is given in vflibcap or not. If the font name is found, VFlib reads the description for the font in vflibcap. The description contains a font class name; VFlib then invokes a font driver corresponding to the font class name. Finally the font driver opens the font file (if necessary).

If the font name is not given in a vflibcap file, a font searching mechanism is invoked. Since there are many font files for X Window and TeX, this feature is introduced to avoid writing an entry for each font file. Various font drivers will be called to see whether the font can be opened. If a font driver succeeds in opening the font, font searching finishes and the VFlib font opening function returns successfully. Otherwise, font open fails.

Fonts described in a vflibcap file are called *explicit fonts* and fonts that are searched for by the font search feature are called *implicit fonts*. Support for explicit and implicit fonts depends on font driver.

Obtaining Bitmaps (Glyph)

Two interfaces are provided to obtain glyph (bitmaps) of a font.

High resolution device oriented glyph

This method is suitable for devices of high resolution such as laser printers. The size of glyph is specified by the physical size of glyph and device resolution. When the size of a glyph in the source font is different from the target size, VFlib scales the source glyph internally.

Low resolution device oriented glyph

This method is suitable for low resolution devices such as CRT displays. Glyph sizes are specified by pixel size rather than by device resolution. When the size of a glyph in the source font is different from the target size, VFlib scales the source glyph internally.

4 Installing VFlib

VFlib uses GNU autoconf and GNU libtool to compile. According to the following procedure, compile and install VFlib.

VFlib (optionally) uses FreeType 1.3.0 or later, T1Lib 5.1 or later, and kpathsea 3.2 libraries. They must be installed before compiling VFlib if you want use them. They are available from the following sites:

- FreeType VFlib is tested with FreeType 1.3. (FreeType 1.0 does not work with current VFlib3.)
  - `http://www.freetype.org/`
  - `ftp://ftp.freetype.org/pub/freetype/freetype-1.3.tar.gz`
- T1Lib VFlib is tested with T1Lib 5.1.
  - `ftp://sunsite.unc.edu/pub/Linux/libs/graphics/`
  - `http://www.neuroinformatik.ruhr-uni-bochum.de/ini/PEOPLE/rmz/t1lib/t1lib.html`
  - `ftp://ftp.neuroinformatik.ruhr-uni-bochum.de/pub/software/t1lib/`
- kpathsea VFlib is tested with kpathsea 3.2 in web2c-7.2b.
  - `ftp://ftp.ctan.org/tex-archive/system/web2c/`

1. VFlib is tested on the following platforms:
   - FreeBSD 2.2.2 and 3.2 on IBM PC-clones
   - Solaris 2.5.1 on Sun SPARC Stations

   Porting to Unix-like system is easy since the source code is not specific system dependent. Porting to non-Unix-like system is not difficult; please try.
2. Go into the directory '`VFlib3-3.6.14`'.
3. Run the `configure` script.
   ```
   % ./configure  [RET]
   ```
   By default, VFlib does *not* support for TrueType and Type1 fonts and TEX font searching by kpathsea library for searching TEX-related font files such as GF, PK, TFM, VF. For such features, VFlib (optionally) uses FreeType library version 1.2 or later for rendering TrueType font files, T1Lib library version 5.1 or later for rendering Type 1 font files, and kpathsea library version 3.2.

   VFlib can be built to use these libraries by giving options when you run `configure` script. Probably, the following is the typical options to configure script to use TrueType and Type 1 fonts and font search feature by kpathsea library.
   ```
   % ./configure \
           --with-kpathsea \
           --with-kpathsea-includedir=/usr/local/include \
           --with-kpathsea-libdir=/usr/local/lib
           --with-freetype \
           --with-freetype-includedir=/usr/local/include/freetype \
           --with-freetype-libdir=/usr/local/lib"
           --with-t1lib \
           --with-t1lib-includedir=/usr/local/include \
           --with-t1lib-libdir=/usr/local/lib            [RET]
   ```
   Note: See the `configure-site` script; it invokes the `configure` script with typical settings shown above.

   Options for configure script is as follows:

`--enable-shared`
: Enable to build a shared library version of VFlib. By default, shared library version is created, if the target system supports shared library.

`--disable-shared`
: Disable to build a shared library version of VFlib.

`--disable-static`
: Disable to build a static library version of VFlib. By default, static library version is not created, if the target system supports shared library.

`--enable-static`
: Enable to build a shared library version of VFlib.

`--disable-bdf`
: VFlib is built without the BDF font driver.

`--disable-pcf`
: VFlib is built without the PCF font driver.

`--disable-hbf`
: VFlib is built without the HBF font driver.

`--disable-gf`
: VFlib is built without the TEX GF font driver.

`--disable-pk`
: VFlib is built without the TEX PK font driver.

`--disable-tfm`
: VFlib is built without the TEX TFM font driver.

`--disable-jtex`
: VFlib is built without the Japanese TEX Kanji font driver.

`--disable-tex-fonts`
: VFlib is built without all TEX-related font drivers, i.e., GF, PK, VF, TFM, and ASCII Japanese TEX Kanji.

`--disable-zeit`
: VFlib is built without the Zeit (Syotai Kurabu) font driver.

`--disable-jg`
: VFlib is built without the JG font driver.

`--disable-ekanji`
: VFlib is built without the eKanji font driver.

`--disable-comic`
: VFlib is built without the Japanese comic font driver.

`--disable-try`
: VFlib is built without the Try font driver.

`--disable-mojikmap`
: VFlib is built without the Mojikyo font mapping driver.

`--with-freetype`*`[=LIB]`*
: FreeType library file is *LIB*. Default value is `ttf`. (Application programs must be linked against '`lib`*`LIB`*`.a`' or '`lib`*`LIB`*`.so`'.)

`--with-freetype-includedir=`*`DIR`*
: FreeType include files are in *DIR*.

`--with-freetype-libdir=`*DIR*
: FreeType library files are in *DIR*.

`--with-t1lib`*[=LIB]*
: T1Lib library file is *LIB*. Default value is `t1`. (Application programs must be linked against '`lib`*LIB*`.a`' or '`lib`*LIB*`.so`'.)

`--with-t1lib-includedir=`*DIR*
: T1Lib include files are in *DIR*.

`--with-t1lib-libdir=`*DIR*
: T1lib library files are in *DIR*.

`--with-kpathsea=`*LIB*
: Kpathsea library file is *LIB*. Default value is `kpathsea`. (Application programs must be linked against '`lib`*LIB*`.a`' or '`lib`*LIB*`.so`'.)

`--with-kpathsea-includedir=`*DIR*
: Kpathsea include files are in *DIR*.

`--with-kpathsea-libdir=`*DIR*
: Kpathsea library files are in *DIR*.

4. Run `make` to compile VFlib.

```
% make  [RET]
```

5. Become a super user (root) and run `make` with '`install`' option to install. (Run `make` with '`uninstall`' option to uninstall.)

```
# make  install [RET]
```

6. If installation is successful, the following has been created:

'`libVFlib.a`' and/or '`libVFlib.so`'
: These are the library files and linked with application programs.

`vflserver`, `vflmkcaptex`, `vflx11`, `vfltest`, etc.
: — A VFlib server and test programs on X11

: By `vflserver`, the functionality of VFlib is available via network if `vflserver` is registered in '`/etc/inetd.conf`'. It can be used interactively by invocation from shell. Interactive use of VFlib is useful for testing or debugging purposes.

: `vflx11` is a test program that displays characters on X Window System. You can use it to test if a font is correctly configured in vflibcap file.

: `vflmkcaptex` is an automatic vflibcap generator for TEX DVI drivers, especially software in the TEX-Guy package. It is a Unix Shell script, and it invokes many subprograms (written in C) to generate font definitions for each font format.

: `vfltest` is a test program that displays characters on terminal by ascii-art form.

Installation directories are as follows:

'`/usr/local/share/VFlib/3.6.14/`' (= '`$prefix/share/VFlib/3.6.14/`')
: Runtime files such as vflibcap are installed here.

: This runtime root directory can be changed on runtime by an environment variable `VFLIB_RUNTIME_DIRECTORY`. If this environment variable is not set, the default directory ('`/usr/local/share/VFlib/3.6.14/`') is used.

: Under this directory, there are following subdirectories:

'ccv' In this directory, code conversion files are installed. This directory can be changed on runtime by setting an environment variable `VFLIB_CCV_DIRECTORY`. If this variable is set, default runtime directory and the value by `VFLIB_RUNTIME_DIRECTORY` variable are ignored.

't1lib' Encoding vector files for T1Lib (for Type 1 fonts) are stored in this directory. Note that the file format for encoding vector files used by T1Lib is different from those used by standard PostScript. To convert standard PostScript encoding vector files into T1Lib format, use '`mkt1enc.sh`' program in this directory. By default, this directory contains files converted from encoding vector files in the `dvips` distribution.

'ascii-jtex'
In this directory, runtime files for Japanese TEX by ASCII Coop. are installed. This directory can be changed on runtime by setting an environment variable `VFLIB_ASCII_JTEX_DIRECTORY`. If this variable is set, default runtime directory and the value by `VFLIB_RUNTIME_DIRECTORY` variable are ignored.

'doc' This directory contains several papers on VFlib, written by Hirotsugu Kakugawa.

'/usr/local/share/VFlib/site/'
Runtime files (vflibcap, ccv files, etc.) that are created by each site should be placed here. For each versiion of VFlib, it has own runtime directory ('`/usr/local/share/VFlib/`*x.y.z*`/`') for default settings; and therefore, the directory where runtime files such as vflibcap in differs by versions of VFlib. In oder to use your own runtime files regardless VFlib versions, runtime files modified for your system environment should be installed in '`/usr/local/share/VFlib/site/`', which is called "site directory".

Before searching in '`/usr/local/share/VFlib/`*x.y.z*`/`', VFlib searches a runtime file in site directory. Note that there is no directory hierarchy in site directory; all runtime files are in the same directory. The site directory can be changed by an environment variable `VFLIB_RUNTIME_SITE_DIRECTORY`.

'/usr/local/bin/'
Binary programs such as `vflserver`, `vflx11`, etc are installed here.

'/usr/local/include/'
Include file for C programs '`VFlib-3_6.h`' is installed here.

'/usr/local/lib/'
VFlib library files such as '`libVFlib.a`', '`libVFlib.so`' are installed here.

Install directories can be changed when you run `configure` script by the '`--prefix=`' option, for example. Invoke `configure` with '`--help`' option for details.

5 Programming with VFlib

5.1 Data types

5.1.1 bitmap type

A bitmap object is a structure of the following:

```
struct vf_s_bitmap {
  int             bbx_width, bbx_height; /* in pixels */
  int             off_x, off_y;          /* in pixels */
  int             mv_x,  mv_y;           /* in pixels */
  unsigned char*  bitmap;
  int             raster;
};
typedef struct vf_s_bitmap*  VF_BITMAP;
```

`bbx_width` and `bbx_height` are the bitmap width and height. A pair of `off_x` and `off_y`) forms a vector to the left-upper corner of the bitmap from the reference point. A pair of `mv_x` and `mv_y` is a vector to the next reference point from the current reference point. (Positive values indicate a move into the right and upper direction respectively.)

The unit of `bbx_width`, `bbx_height`, `off_x`, `off_y`, `mv_x`, and `mv_y` is pixels. `bitmap` is a pointer to the bitmap data; one pixel corresponds to one bit.

The left upper corner is the beginning of the bitmap data, and a raster line is defined as a horizontal line from the left to the right corner of the glyph bitmap. `bitmap` is a sequence of raster lines starting from the top to the bottom. The distance (in bytes) of two consecutive raster lines in memory is given by `raster`. Although the raster line length of a bitmap is (`bbx_width`+7)/8, it is not guaranteed that this value is the same as `raster`. The type of `bitmap` is a pointer to `unsigned char` data object, and each `unsigned char` data object contains eight pixels. Let `P[0]` be the start address of a raster line. The `x`-th pixel counted from the leftmost pixel (which is pixel 0) is 1 if and only if `P[x/8] & (0x80>>(x%8))` is non-zero.

If `bbx_width` is not a multiple of 8, there exist bits that do not correspond to any pixels but their values are always zero. Even if `bbx_width` and/or `bbx_height` are zero, at least one byte is allocated for the bitmap data. Thus, `bitmap` is always non-NULL.

5.1.2 metric1 type

A metric1 object is a structure of the following:

```
struct vf_s_metric1 {
  double    bbx_width, bbx_height;  /* in points */
  double    off_x, off_y;           /* in points */
  double    mv_x, mv_y;             /* in points */
};
typedef struct vf_s_metric1*  VF_METRIC1;
```

The members of this structure are the same as the members of a bitmap object but the members' unit is point.

5.1.3 metric2 type

A metric2 object is a structure of the following:

```
struct vf_s_metric2 {
  int    bbx_width, bbx_height;     /* in pixels */
  int    off_x, off_y;              /* in pixels */
  int    mv_x, mv_y;                /* in pixels */
};
typedef struct vf_s_metric2*  VF_METRIC2;
```

The members of this structure are the same as the members of a bitmap object, and the members' unit is pixel also.

5.1.4 outline type

VFlib defines its private outline data formats for presenting vector data of characters. This data format is used by VFlib API functions `VF_GetOutline()` and `VF_OutlineToBitmap()`.

Each font driver returns a outline data of a character of a font if a font driver of that font supports `VF_GetOutline()` function. Even if data format of a font is different from VFlib-format, a font driver converts outline data to VFlib-format data. For instance, the PCF font driver (note that PCF is a bitmap font format) supports `VF_GetOutline()` function and it constructs and return an outline data which is a set of square; each square corresponds to a pixel of a bitmap.

Note that not all font drivers support `VF_GetOutline()` function, but most of them do. The developer of font drivers are strongly recommented to implement this feature even if the font font format is bitmap-based. (The function is supported by BDF, PCF, HBF, PK, GF, TFM, Zeit, JG, TrueType, and Type 1 font drivers.)

Outline data is defined as follows:

```
/* Outline data */
typedef long               VF_OUTLINE_ELEM;
typedef VF_OUTLINE_ELEM    *VF_OUTLINE;
```

According to CPU architecture, `VF_OUTLINE_ELEM` is defined as `int` if size of `long` is 8.

```
typedef long               VF_OUTLINE_ELEM;
```

Outline data is an array of VF_OUTLINE_ELEM type (`long` or `int` type). Outline data consists from two parts: header and outline. The header part contains metric data and outline part contains outline representation of a character.

Documentation for this feature is not finished. See the source code (e.g., `VFlib-3_6.h`, `raster.c`, `bm2ol`, for example) for further information.

5.2 Functions and variables

5.2.1 `VF_Init()`

`int  VF_Init(char* `*`vflibcap`*`, char* `*`variable_list`*`)`

Functionality
: Initialization of VFlib.

Arguments

vflibcap is a file name of vflibcap (this file is a font database). If the null pointer is given, the default path name given on compile time is used (a typical default value is `/usr/local/lib/VFlib/3.6.14/vflibcap`). Searching of a vflibcap file is done in the following way. First, VFlib try to open a file as given to the first argument. (That is, VFlib searches it relative to current working directory.) If not found, then VFlib try to open the file under default runtime directory (e.g., `/usr/local/lib/VFlib/3.6.14/`). For example, `vflibcap-tex` is given, VFlib first look for `vflibcap-tex` in current directory, and then, it looks for the file under runtime directory.

Default runtime directory can be changed by an environment variable `VFLIB_RUNTIME_DIRECTORY` on runtime. If an environment variable `VFLIB_VFLIBCAP_PATH` is set, the first argument of this function is ignored and its value is used. If an environment variable `VFLIB_VFLIBCAP_DIRECTORY` is set, a vflibcap file is searched under a directory sepecified by this environment variable.

variable_list is a list of parameters passed to VFlib. This is used to specify values of parameterized vflibcap files. (See basic.txt for parameterized vflibcap file.) The type of this argument is a string and its syntax is a sequence of *Variable*=*Value*, separated by a comma `,`. For example, `DPI=400, LEVEL=1, FOO=bar`.

Return value

If initialization succeeds, a non-negative integer is returned. If initialization fails, a negative integer is returned.

5.2.2 `vf_error`

```
int vf_error
```

Functionality

This is a global variable. Holding the error code of VFlib. If no error, it keeps 0. If an error occurs, the corresponding error code is set.

5.2.3 `VF_ClearError()`

```
void VF_ClearError(void)
```

Functionality

Clear the error code variable of VFlib.

5.2.4 `VF_OpenFont1()`

```
int VF_OpenFont1(char* font_name,
                 double dpi_x, double dpi_y, double point_size,
                 double mag_x, double mag_y)
```

Functionality

Open a font. (If the same font is opened multiple times, VFlib keeps track of the number of opened font instances of the font.) Since the font is opened with device resolution, point size and magnification, a font opened by this function may be useful for high resolution devices such as laser printers.

Arguments

The argument *font_name* is a name of the font to be opened. The device resolution of the target device is specified by *dpi_x* (horizontal) *dpi_y* (vertical). These values are given in DPI (dots per inch). The argument *point_size* specifies the size of the bitmap. If this argument is negative the bitmap size will be the inherent size of the font.

To obtain a magnified bitmap, give a magnification factor to the argument *mag_x* (horizontal) and *mag_y* (vertical). If the argument *point_size* is non-negative, font size will be *point_size* times *mag_x* (*mag_y*) large for horizontal (vertical) direction.

Return Value
: A non-negative integer is returned on success. This value is a font identifier (font id); it is used to specify a font for further font operations. If `VF_OpenFont1()` fails, a negative integer is returned.

5.2.5 `VF_OpenFont2()`

```
int  VF_OpenFont2(char* font_name,
                  int pixel_size, double mag_x, double mag_y)
```

Functionality
: Open a font. (If the same font is opened multiple times, VFlib keeps track of the number of opened font instances of the font.) Since the font is opened with pixel size and magnification, a font opened by this function may be useful for low resolution devices such as CRT display.

Arguments
: The argument *font* is a name of the font to be opened. The argument *pixel_size* specifies the size of the bitmap. If this argument is negative the bitmap size will be the inherent size of the font. To obtain a magnified bitmap, give a magnification factor to the argument *mag_x* (horizontal) and *mag_y* (vertical). If the argument *point_size* is non-negative, font size will be *pixel_size* times *mag_x* (*mag_y*) large for horizontal (vertical) direction.

Return Value
: A non-negative integer is returned on success. This value is a font identifier (font id); it is used to specify a font for further font operations. If `VF_OpenFont2()` fails, a negative integer is returned.

5.2.6 `VF_CloseFont()`

```
int  VF_CloseFont(int font_id)
```

Functionality
: Close a font.

Arguments
: The argument *font_id* is a font id to be closed.

Return Value
: A non-negative integer is returned on success. A negative integer is returned on failure.

5.2.7 `VF_GetBitmap1()`

```
VF_BITMAP  VF_GetBitmap1(int font_id, long code_point,
                         double mag_x, double mag_y)
```

Functionality
: Obtain a glyph bitmap of given font id and code point. The font id *font_id* must be an id by `VF_OpenFont1()`. Size of bitmap to be obtained can be specified by *mag_x* and *mag_y* arguments.

Arguments
: *font_id* specifies the font; *code_point* specifies the code point of a character. To obtain a magnified bitmap, give a magnification factor to the argument *mag_x* (horizontal) and *mag_y* (vertical). If a font is opened with magnification factor 2 and an bitmap is obtained by this function with magnification factor 2, then the size of yielding bitmap will be 4 times larger than the original size.

Return Value
: The return value is a pointer to a newly allocated bitmap object. If it fails to obtain a bitmap, the null pointer is returned. If the bitmap object is no longer needed, it must be released by the function `VF_FreeBitmap()`. The font may not have the specified size; in such case, VFlib internally enlarges or shrinks the glyph to obtain a bitmap of the requested size.

5.2.8 VF_GetBitmap2()

```
VF_BITMAP  VF_GetBitmap2(int font_id, long code_point,
                            double mag_x, double mag_y)
```

Functionality
: Obtain a glyph bitmap of given font id and code point. The font id 'font_id' must be an id by `VF_OpenFont2()`. Size of bitmap to be obtained can be specified by *pixel_size*, *mag_x* and *mag_y* arguments.

Arguments
: *font_id* specifies the font; *code_point* specifies the code point of a character. To obtain a magnified bitmap, give a magnification factor to the argument *mag_x* (horizontal) and *mag_y* (vertical). If a font is opened with magnification factor 2 and an bitmap is obtained by this function with magnification factor 2, then the size of yielding bitmap will be 4 times larger than the original size.

Return Value
: The return value is a pointer to a newly allocated bitmap object. If it fails to obtain a bitmap, the null pointer is returned. If the bitmap object is no longer needed, it must be released by the function `VF_FreeBitmap()`. The font may not have the specified size; in such case, VFlib internally enlarges or shrinks the glyph to obtain a bitmap of the requested size.

5.2.9 VF_GetMetric1()

```
VF_METRIC1  VF_GetMetric1(int font_id, long code_point,
                             VF_METRIC1 metric1,
                             double mag_x, double mag_y)
```

Functionality
: Obtain font metrics of a given font and code point.

Arguments
: Same arguments as of `VF_GetBitmap1()`.

Return Value

A pointer to a metric1 object is returned. If an error occurs, the NULL pointer is returned. The obtained metric is a metric for a bitmap obtained by `VF_GetBitmap1()` with the same arguments, but the unit of the obtained metric is point. If the metric1 object is no longer needed it must be released by the function `VF_FreeMetric1()`.

5.2.10 `VF_GetMetric2()`

```
VF_METRIC2  VF_GetMetric2(int font_id, long code_point,
                          VF_METRIC2 metric2,
                          double mag_x, double mag_y)
```

Functionality

Obtain font metrics of a given font and code point.

Arguments

Same arguments as of `VF_GetBitmap2()`.

Return Value

A pointer to a metric2 object is returned. If an error occurs, the NULL pointer is returned. The obtained metric is a metric for a bitmap obtained by `VF_GetBitmap2()` with the same arguments, but the unit of the obtained metric is pixel. If the metric2 object is no longer needed, it must be released by the function `VF_FreeMetric2()`.

5.2.11 `VF_GetOutline()`

```
VF_OUTLINE  VF_GetOutline(int font_id, long code_point,
                          double mag_x, double mag_y)
```

Functionality

Obtain outline data from a given font and code point.

Arguments

Same as `VF_GetBitmap1()`.

Return Value

Return value is a pointer to a newly allocated outline data object. If it fails to obtain a outline data, the NULL pointer is returned. Even if the original font is a bitmap, VFlib internally creates outline data from the bitmap. If the source font is a vector font, VFlib internally converts the data format to VFlib outline data style. A bitmap of any specified size can be obtained from outline data by the function `VF_Outline2Bitmap()`. (Default point size and device resolution is also kept in the outline data.)

5.2.12 `VF_OutlineToBitmap()`

```
VF_OUTLINE  VF_OutlineToBitmap(VF_OUTLINE outline,
                               double dpi_x, double dpi_y,
                               double point_size,
                               double mag_x, double mag_y)
```

Functionality

Obtain a bitmap from outline data.

Arguments

The argument *outline* is a pointer to an outline object to be rasterised. The arguments *dpi_x*, *dpi_y*, *point_size*, *mag_x* and *mag_y* are the same as the corresponding arguments of `VF_GetBitmap1()`. The outline data contains information on device resolution and point size specified by `VF_GetOutline()`. (If not specified, default values are used. Thus, bitmaps with a default size can be obtained by giving -1 for the arguments).

Return Value
: A pointer to a bitmap object is returned. The NULL pointer is returned on failure. If the bitmap object is no longer needed it must be released by the function `VF_FreeBitmap()`.

5.2.13 VF_GetFontBoundingBox1()

```
int  VF_GetFontBoundingBox1(int font_id,
                            double mag_x, double mag_y,
                            double* w, double* h,
                            double* xoff, double* yoff)
```

Functionality
: Obtain font bounding box information of a given font.

Arguments
: The argument *font_id* specify a font in interest. The function writes the bounding box information to the locations pointed by *w*, *h*, *xoff*, and *yoff*. *w* and *h* point to data objects for width and height of bounding box, respectively. *xoff* and *yoff* point to data objects for largest horizontal and vertical displacement of lower left corner of bounding box from reference points. Note that these values does not guarantee the minimality; they only guarantee that all characters can be contained in a box descrived by them. If some values of *w*, *h*, *xoff*, or *yoff* are not in interest, NULL pointer can be given.

The argument *mag_x* and *mag_y* are maginification factor to be scaled for a given font *font_id*.

Return Value
: If font bounding information is successfully obtained, a non-negative integer is returned; otherwize, negative integer is returned.

Units of bounding box information is in point.

5.2.14 VF_GetFontBoundingBox2()

```
int  VF_GetFontBoundingBox2(int font_id,
                            double mag_x, double mag_y,
                            int* w, int* h,
                            int* xoff, int* yoff)
```

Same as `VF_GetFontBoundingBox1()` except units of font bounding box parameters are pixel.

5.2.15 VF_GetProp()

```
char*  VF_GetProp(int font_id, char* prop_name)
```

Functionality
: Obtain a property of given font. (This function is font class dependent. You must be very careful to use it!)

Arguments
: The argument *font_id* specifies a font from which to obtain a property. *property_name* specifies the property name.

Return Value

If the given property exists, its value is returned as a string. The string for the property value is newly allocated and must be released by *free()* if it is no longer needed. If the given property is undefined, the NULL pointer is returned.

5.2.16 VF_CopyBitmap()

```
VF_BITMAP  VF_CopyBitmap(VF_BITMAP bm)
```

Functionality

Make a copy of a bitmap object.

Arguments

The argument *bm* is a pointer to a bitmap object to be copied.

Return Value

A new bitmap object is allocated; all values are copied. Return value is a pointer to a new bitmap. The source bitmap *bm* remains unaffected. If an error occurs, the NULL pointer is returned. The obtained bitmap object must be released by `VF_FreeBitmap()` if it is no longer needed.

5.2.17 VF_MakeScaledBitmap()

```
VF_BITMAP  VF_MakeScaledBitmap(VF_BITMAP bm,
                               double mag_x, double mag_y)
```

Functionality

Make an enlarged or shrinked bitmap.

Arguments

The argument *bm* specifies the source bitmap object, *mag_x* and *mag_y* give the magnification factor in the horizontal and vertical direction respectively. If the magnification factor is less than 1, a shrinked bitmap is obtained. Values for *mag_x* and *mag_y* can be arbitrary such as (*mag_x* $>$ 1 and *mag_y* $<$ 1) or (*mag_x* $<$ 1 and *mag_y* $>$ 1).

Return Value

A bitmap object whose bitmap is enlarged or shrinked is created and a pointer to the new bitmap is returned. If an error occurs, the NULL pointer is returned. The source bitmap *bm* remains unaffected. Use `VF_FreeBitmap()` if the returned bitmap object is no longer necessary.

5.2.18 VF_ReflectedBitmap()

```
VF_BITMAP  VF_ReflectedBitmap(VF_BITMAP bm,
                              int refl_x, double refl_y)
```

Functionality

Make a bitmap with horizontally and/or vertically reflected image.

Arguments

The argument *bm* specifies the source bitmap object, *refl_x* and *refl_y* specify the reflection, respectively. If *refl_x* is non-zero, the image is holizontally reflected; if *refl_y* is non-zero, the image is vertically reflected. In case *relf_x* and *refl_y* are both zero, the effect is the same as `VF_CopyBitmap()`.

Return Value

A new bitmap object is created and a pointer to the new bitmap is returned. If an error occurs, the NULL pointer is returned. Metrics of created bitmap is the same as that of the original bitmap. The source bitmap *bm* remains unaffected. Use `VF_FreeBitmap()` if the returned bitmap object is no longer necessary.

5.2.19 VF_RotatedBitmap()

```
VF_BITMAP  VF_RotatedBitmap(VF_BITMAP bm, int angle)
```

Functionality

Make a bitmap image with rotated image.

Arguments

The argument *bm* specifies the source bitmap object, *angle* gives rotation angle in degree. By the limitation of implementation, rotatin angle must be multiple of 90. The rotation angle *angle* must be one of the following:

`VF_BM_ROTATE_0`

Rotation angle is zero. Thus, the effect is the same as `VF_CopyBitmap()`.

`VF_BM_ROTATE_90`

Rotation angle is 90 degree.

`VF_BM_ROTATE_180`

Rotation angle is 180 degree.

`VF_BM_ROTATE_270`

Rotation angle is 270 degree.

Return Value

A bitmap object whose bitmap is rotated is created and a pointer to the new bitmap is returned. If an error occurs, the NULL pointer is returned. The source bitmap *bm* remains unaffected. Use `VF_FreeBitmap()` if the bitmap object is no longer necessary.

This function rotates a bitmap with the reference point as origin. The vector to the next reference point is also rotated. Therefore, position of the reference point and a vector to the next reference point of `VF_RotatedBitmap(bm, VF_BM_ROTATE_180)` and that of `VF_ReflectedBitmap(bm, 1, 1)` are different.

5.2.20 VF_DumpBitmap()

```
void  VF_DumpBitmap(VF_BITMAP bm)
```

Functionality

Print a bitmap in ASCII-art-style to stdout.

Arguments

The argument *bm* specifies a bitmap to be displayed.

5.2.21 VF_ImageOut_PBMAscii()

```
int  VF_ImageOut_PBMAscii(VF_BITMAP bm, FILE *fp,
                          int image_width, int image_height,
                          int position_x, int position_y,
                          int margin_l, int margin_r,
                          int margin_t, int margin_b,
```

```
                          int reverse, int shrink,
                          char *prog, char *title)
```

Functionality

Print a bitmap *bm* in PBM ASCII format to a file stream *fp*.

Arguments

bm is a bitmap to be written to a file stream *fp*. Size of output image (in pixel) is specified by *image_width* and *image_height*. If -1 is given for these arguments, the image size should be minimum to contain the bitmap *bm*.

Arguments *position_x* and *position_y* specifies the horizontal and vertical position of a source bitmap *bm* in an output image file, respectively. These parameters have effect when *image_width* and *image_height* are specified. Possible values for *position_x* is

`VF_IMAGEOUT_POSITION_NONE`
: Same as `VF_IMAGEOUT_POSITION_LEFT`.

`VF_IMAGEOUT_POSITION_CENTER`
: *bm* is centered in output image.

`VF_IMAGEOUT_POSITION_LEFT`
: *bm* is flushed left in output image.

`VF_IMAGEOUT_POSITION_RIGHT`
: *bm* is flushed righted in output image.

Possible values for *position_y* is

`VF_IMAGEOUT_POSITION_NONE`
: Same as `VF_IMAGEOUT_POSITION_TOP`.

`VF_IMAGEOUT_POSITION_CENTER`
: *bm* is centered in output image.

`VF_IMAGEOUT_POSITION_TOP`
: *bm* is placed at the top in output image.

`VF_IMAGEOUT_POSITION_BOTTOM`
: *bm* is placed at the bottom in output image.

Arguments *margin_l*, int *margin_r* are used to speficy left and right margins, respectively. Arguments *margin_t*, int *margin_b* are used to speficy top and bottom margins, respectively.

If the argument *reverse* is not 0, black and white in an output image is reversed. Argument *shrink* specifys shrink factor of image *bm*. (If this value is 1, *bm* is not shrinked. Note: Currently, shrinking image is not supported in PBM ASCII format.)

Arguments *prog* and *title* are used to emmbed program name and title in an image file.

5.2.22 `VF_ImageOut_PGMAscii()`

```
int  VF_ImageOut_PGMAscii(VF_BITMAP bm, FILE *fp,
                          int image_width, int image_height,
                          int position_x, int position_y,
                          int margin_l, int margin_r,
                          int margin_t, int margin_b,
                          int reverse, int shrink,
                          char *prog, char *title)
```

Functionality

Print a bitmap *bm* in PGM ASCII format to a file stream *fp*.

Arguments

Arguments are the same as that of `VF_ImageOut_PGMAscii()`. If *shrink* is greater than 1, output image is anti-aliased (gray-scaled).

5.2.23 VF_ImageOut_PGMRaw()

```
int  VF_ImageOut_PGMRaw(VF_BITMAP bm, FILE *fp,
                        int image_width, int image_height,
                        int position_x, int position_y,
                        int margin_l, int margin_r,
                        int margin_t, int margin_b,
                        int reverse, int shrink,
                        char *prog, char *title)
```

Functionality

Print a bitmap *bm* in PGM Raw format to a file stream *fp*.

Arguments

Arguments are the same as that of `VF_ImageOut_PGMAscii()`. If *shrink* is greater than 1, output image is anti-aliased (gray-scaled).

5.2.24 VF_ImageOut_EPS()

```
int  VF_ImageOut_EPS(VF_BITMAP bm, FILE *fp,
                     int image_width, int image_height,
                     int position_x, int position_y,
                     int margin_l, int margin_r,
                     int margin_t, int margin_b,
                     int reverse, int shrink,
                     char *prog, char *title,
                     double ptsize, int pixsize)
```

Functionality

Print a bitmap *bm* in EPS (Encapsulated PostScript) format to a file stream *fp*.

Arguments

Arguments are the same as that of `VF_ImageOut_PGMAscii()`. If *shrink* is greater than 1, output image is anti-aliased (gray-scaled). Arguments *ptsize* and *pixsize* specify size of EPS bounding box; *pixsize* pixels occupy *ptsize* points in physical paper.

5.2.25 VF_ImageOut_ASCIIArt()

```
int  VF_ImageOut_ASCIIArt(VF_BITMAP bm, FILE *fp,
                          int image_width, int image_height,
                          int position_x, int position_y,
                          int margin_l, int margin_r,
                          int margin_t, int margin_b,
                          int reverse, int shrink)
```

Functionality

Print a bitmap *bm* in ASCII art format to a file stream *fp*.

Arguments

Arguments are the same as that of `VF_ImageOut_PGMAscii()`.

5.2.26 `VF_ImageOut_ASCIIArtV()`

```
int  VF_ImageOut_ASCIIArtV(VF_BITMAP bm, FILE *fp,
                           int image_width, int image_height,
                           int position_x, int position_y,
                           int margin_l, int margin_r,
                           int margin_t, int margin_b,
                           int reverse, int shrink)
```

Functionality

Print a bitmap *bm* in ASCII art format to a file stream *fp*. Image is rotated in clockwise, 90 degree.

Arguments

Arguments are the same as that of `VF_ImageOut_PGMAscii()`.

5.2.27 `VF_FreeBitmap()`

```
void  VF_FreeBitmap(VF_BITMAP bm)
```

Functionality

Release a bitmap object.

Arguments

The argument *bm* is a pointer to a bitmap object to be released.

5.2.28 `VF_FreeMetric1()`

```
void  VF_FreeMetric1(VF_METRIC1 metric)
```

Functionality

Release a metric1 object.

Arguments

The argument *metric* is a pointer to a metric1 object.

5.2.29 `VF_FreeMetric2()`

```
void  VF_FreeMetric2(VF_METRIC2 metric)
```

Functionality

Release a metric2 object.

Arguments

The argument *metric* is a pointer to a metric2 object.

5.2.30 `VF_InstallFontDriver()`

```
int  VF_InstallFontDriver(char* class_name,
                          int(*driver)(VF_FONT font,
                                       char* class_name,
                                       char* font_name,
                                       int implicit_flag));
```

Functionality
Install a font driver.

Arguments
class_name is a font class name, *driver* is a pointer to a function of a font driver of the font class. The function given by *driver* is called when a font of this font class is opened by `VF_OpenFont1()` and `VF_OpenFont2()`. The function *driver* is called with parameters of the font to be opened: *font* is a data object for font management defined by VFlib internally. *class_name* is the font class name. *font_name* is the name of the font to be opened. This argument is the same as the argument of `VF_OpenFont1()` and `VF_OpenFont2()`. *implicit_font* is a flag whose value is 1 if a font is to be opened as an implicit font (a font which does not explicitly appear in vflibcap) and 0 if it is to be opened as an explicit font (a font that does appear in vflibcap).

Return Value
If successful, a non-negative integer is returned. A negative integer is returned if the installation of the font driver fails.

5.3 Building an application software with VFlib

An application software that use VFlib must include a header file '`VFlib-3_6.h`'. Typically, this file is installed '`/usr/local/include/`' directory.

Never forget, that application software that uses VFlib must be linked against FreeType 1.2 or later ('`libttf.a`' or '`libttf.so`'), T1Lib 5.1 or later ('`libt1.a`' or '`libt1.so`'), and kpathsea 3.2 ('`libkpathsea.a`' or '`libkpathsea.so`'), if you configure VFlib to use them. (If VFlib is configured not to use them, they are not necessary.)

I recommend shared library versions for these optional libraries if you built a shared library version of VFlib.

```
#include <VFlib-3_6.h>
```

VFlib must be initialized before it is used.

```
char* vflibcap = "vflibcap";
char* params =  "TeX_DPI=300, KPATHSEA_MODE=cx";

if (VF_Init(vflibcap, params) < 0){
  fprintf(stderr, "Initializing VFlib: failed\n");
  exit(1);
}
```

Following program fragment opens a font, obtains a bitmap, and print obtained bitmap.

```
int         fid;
VF_BITMAP   bm;

if ((fid = VF_OpenFont2("timR24.pcf", -1, 1.0, 1.0)) < 0){
  fprintf(stderr, "Opening font: failed\n");
  exit(1);
}

bm = VF_GetBitmap2(fid, 0x67, 1.0, 1.0);

VF_DumpBitmap(bm);
```

5.4 A simple example

The following program code is a "banner" like using VFlib. For simplicity, this program accepts only 1-byte encoded characters. It reads an input from standard input and prints characters in ascii-art form to standard output.

```
/*
 * vflbanner.c - a banner by VFlib
 * by Hirotsugu Kakugawa
 *
 *
 */
/*
 * Copyright (C) 1998 Hirotsugu Kakugawa.
 * All rights reserved.
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2, or (at your option)
 * any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include "VFlib-3_6.h"

#define  DEFAULT_FONT  "timR18.pcf"


char    *vflibcap;
char    *fontname;

void  usage(void);
void  vflbanner(FILE *fp);


int
main(int argc, char **argv)
{
  vflibcap = NULL;
  fontname = DEFAULT_FONT;
```

```
  --argc; argv++;
  while (argc > 0){
    if ((argc >= 1)
        && ((strcmp(argv[0], "-h") == 0) || (strcmp(argv[0], "--help") == 0))){
      usage();
      exit(0);
    } else if ((argc >= 2) && (strcmp(argv[0], "-v") == 0)){
      --argc; argv++;
      vflibcap = argv[0];
      --argc; argv++;
    } else if ((argc >= 2) && (strcmp(argv[0], "-f") == 0)){
      --argc; argv++;
      fontname = argv[0];
      --argc; argv++;
    } else {
      break;
    }
  }

  vflbanner(stdin);

  return 0;
}

void usage(void)
{
  printf("vflbanner - a banner program using VFlib\n");
  printf("Usage: vflbanner [-v vflibcap] [-f fontname]\n");
  printf("This program reads a text from standard input.  It supports\n");
  printf("1-bit encoded font only. Thus, ‘ctextpgm’ is better than this.\n");
}


void
vflbanner(FILE  *fp)
{
  int  fid;
  int  ch;
  int  pos_x, pos_y;
  VF_BITMAP  bm, page_bm;
  struct vf_s_bitmaplist  PageBuff;

  if (VF_Init(vflibcap, NULL) < 0){
    printf("VFlib initialization error");
    switch (vf_error){
    case VF_ERR_INTERNAL:
      printf(" - Internal error.\n"); break;
    case VF_ERR_NO_MEMORY:
      printf(" - Server runs out of memory.\n"); break;
    case VF_ERR_NO_VFLIBCAP:
      printf(" -  No vflibcap.\n"); break;
```

```
    default:
      printf(" -  Error code %d\n", vf_error); break;
    }
    fflush(stdout);
    exit(1);
  }

  if ((fid = VF_OpenFont1(fontname, -1, -1, -1, 1, 1)) < 0)
    return;

  VF_BitmapListInit(&PageBuff);

  pos_x = pos_y = 0;
  while ((ch = getc(fp)) != EOF){
    if (!isprint(ch))
      ch = ' ';
    if ((bm = VF_GetBitmap1(fid, (long)ch, 1, 1)) == NULL)
      continue;
    VF_BitmapListPut(&PageBuff, bm, pos_x, pos_y);
    pos_x = pos_x + bm->mv_x;
  }

  page_bm = VF_BitmapListCompose(&PageBuff);
  VF_DumpBitmap(page_bm);
  VF_BitmapListFinish(&PageBuff);
  VF_FreeBitmap(page_bm);

  VF_CloseFont(fid);
}

/*EOF*/
```

By the following commands is used to comple the program.

```
% gcc -c `VFlib3-config --cflags` vflbanner.c
% gcc -o vflbanner vflbanner.o `VFlib3-config --libs`
```

`VFlib3-config` is a program to print misc information on configuration of VFlib. It prints C compiler option to specify include directory (‘`--cflags`’), dependent libraries (‘`--libs`’), for example. Run `VFlib3-config` with ‘`--help`’ option for detail.

6 Writing a vflibcap

6.1 Introduction to vflibcap

A vflibcap file is a database of font definitions for the VFlib library. A vflibcap font definition is described in a format similar to termcap and printcap. Vflibcap provides logical font names and logical font names may not corresponds to font files to be accessed. In this document, we simply say "font" to denote logical fonts.

Each VFlib fonts have its own parameters listed below:

- Pixel size,
- Point size, and
- Resolution of target device.

These parameters may not be available in font file. For instance, these parameters are lacking in TrueType fonts. Lacking information is given in vflibcap file, or it is given by a font driver as default values.

Note: If you want to use VFlib for TeX DVI drivers, such as previewers and pronter driver, use `vflmkcaptex` program to generate vflibcap file automatically.

6.2 Syntax of vflibcap

The syntax of vflibcap file is lisp-like form. A semicolon `;` starts a comment and following text is ignored until the end of line. A colon in a string which is enclosed by double colons is not considered as a comment character and forms a part of string. In the following explanation, we ignore comments.

A vflibcap file is a sequence of expressions called s-expressions. Basic data item of s-expression is string. Unlike lisp, there is no "number" type. A sequence of digits is parsed as a string. To include a special characters in a string such as double quotation, control code, and parenthesis, escape sequence can be used. String is a sequence of characters of the following form:

String Form 1:
: Sequence of characters enclosed by double quotations.

  Examples:

```
"hello world"
"a, b, c, d"
"He said \"Thanks!\"."
"a*(b+c)"
```

String Form 2:
: Sequence of characters except space, tab, newline, and closing parenthesis. A start character must not be a double quotation.

  Examples:

```
font-file
hello\ world
He\ said\ \"Thanks!\".
a*\(b+c\)
```

Unlink lisp, there is no distinction between string and symbol in vflibcap; they are the same data type. That is, `HELLO` and `"HELLO"` are the same.

Parentheses is used to form a "list" like in lisp. For example, `(A B C)` is a list of three strings. List can be nested any depth, e.g., `(A (B1 B2 B3) (C1 (C21 C22) C3))`.

A vflibcap must be a sequence of s-expression of the following forms:

`(define-default` *`FONT-CLASS CAPABILITY-DEF`* `... )`
: This expression defines a default values for a font class.

`(define-font` *`NAME CAPABILITY-DEF`* `... )`
: This expression defines a font.

`(define-macro` *`NAME CAPABILITY-DEF`* `... )`
: This expression defines a macro *NAME*.

CAPABILITY-DEF must be a list of form `(`*`CAPABILITY-NAME VALUE`*`)`, e.g., `(font-file "/usr/local/share/fonts/"`. Each font class defines its own set of capabilities and capability sets can be different by font classes.

This is an example of vflibcap file.

```
;; vflibcap
(define-default VFlib
  (extension-hints         (".bdf" bdf) (".pcf" pcf))
  (uncompression-programs  (".Z" "zcat") (".gz" "gzip -cd")
                           ("pk" ascii-jtex-kanji))
  (implicit-font-classes   bdf pcf hbf ascii-jtex-kanji)
  (variable-values     ("TeX_DPI" "300"))

(define-default bdf
  (filename-extensions ".bdf")
  (font-directories
     "/usr/X11R6/lib/X11/fonts//" "/usr/local/share/fonts/X11//")
  (compression-extensions ".gz" ".Z"))

(define-font timR24  ; times roman 24
  (font-class bdf)
  (font-file "timR24.bdf"))

(define-font timR18  ; times roman 18
  (font-class bdf)
  (font-file "timR18.bdf"))
```

6.3 Macros in vflibcap

To avoid writing the same capabilities, macro feature is provided in vflib. In case of *CAPABILITY-DEF* is a string, it is treated as a macro and a macro definition for it is expanded. For instance,

```
(define-font timR18
  MACRO-NAME
  (font-file "timR18.bdf"))
```

is a font definition using a macro `MACRO-NAME`. Suppose a macro `MACRO-NAME` is defined as follows.

```
(define-macro MACRO-NAME
  (font-class bdf)
  (dpi 300))
```

Then, the font definition for `timR24` is the same as follow.

```
(define-font timR18
  (font-class bdf)
  (dpi 300))
  (font-file "timR18.bdf"))
```

The rule of macr expand is as following procedure.

1. Looks for use of macros. From the first *CAPABILITY-DEF* to the last one, it is checked if it is a string (thus a macro) or not in order. If it is a macro, corresponding macro definition is substituted. Then, next *CAPABILITY-DEF* is checked.
2. Macro expand is done recursively. Thus, a macro can be used in another macro.

6.4 Searching font files

Some font classes (e.g., BDF, PCF) defines a `font-directories` capability in vflibcap file. This capability specifies a list of font directories, for instance, `(font-directories "/usr/local/fonts/" "/opt/fonts" "/usr/local/share/fonts//")`. A font file can be searched recursively in a directory tree if a font directory name ends by double slashes `//`.

Some font drivers support file search by kpathsea. Typically, font files are located under '`/usr/local/share/texmf`'. This directory is used to hold TEX-related files. If a font driver supports searching by kpathsea, a special name `TEXMF` can be given in a list of `font-directories` capability. For instance, suppose that `(font-directories "/opt1/fonts//" "TEXMF" "/opt2/fonts//")` is specified. Then files are searched under `/opt1/fonts`, by kpathsea, and then `/opt2/fonts`, in this order.

Currently, pk, gf, tfm, vf, truetype, and type1 font classes suport searcing files by kpathsea.

6.5 Fast font file search

In case there are many font directories and sub-directory which contains many font file, searching a font file take long time, since font directories are traversed to find a requested font file. For fast font file search, font file hint database (FDB for short) can be used. It is placed in a root of a font directory, and it contains pairs of font file name and relative pathname of the font file from the font directory. The file name of FDB is `VFlib.fdb`.

The following is an example of FDB file.

```
times__m.pfb type1/t/times__m.pfb
times__m.afm type1/t/times__m.afm
zac_____.ttf ttf/z/zac_____.ttf
zalescap.ttf ttf/z/zalescap.ttf
```

Suppose that this FDB file is located in '`/foo/bar/`', for instance. The file tells us that there is a file '`times__m.pfb`' and absolute path name of the file is '`/foo/bar/type1/t/times__m.pfb`'.

If FDB file is found in a root directory of font directory, the FDB file is opened to find a requested font file. If a requested font file is not found, other font directory is searched, i.e., the directory is not traversed at all. In case FDB file is not found, a font directory is traversed to find a requested font file.

It is important to remember that you must not forget to update FDB file after you added new font files in a font directory. If you forget, installed font files are not found evenif they are in a font directory. To update a FDB file, run the utility program `vflmkfdb`. See Section 9.3 [vflmkfdb], page 81, for details of the program.

A FDB file must be located in a root of a font directory and its name must be '`VFlib.fdb`'. Even if there is a FDB file in a sub-directory of a font directory, VFlib does not look for it.

6.6 Compressed font files

To reduce disk storage, compressed font files and uncompression on the fly is supported by some font class. Note that this feature is font class dependent and not all font class support this.

In a vflibcap file, a font file name need not have a compressed type extension, such as `.gz`. When VFlib searches a font file, it internally adds compressed type extension and finds a file.

6.7 Explicit and implicit fonts

Fonts explicitly defined in a vflibcap file are called *explicit fonts*. Fonts does not appear vflibcap file and searched by font drivers on demand are called *implicit fonts*.

6.8 Variables in vflibcap

In a vflibcap file, variables can be used as capability values. A capability value can be a value of a variable if a dollar sign (`$`) followed by a variable name is given.

For instance, `(dpi $TeX_DPI)` can be used instead of `(dpi 300)`. The value for a variable must be defined somewhere. Default value can be given in `(define-default VFlib ...)`, which will be explained later.

Default values can be overridden on initialization function of VFlib `VF_Init()`, or Unix environment variables `VFLIBCAP_PARAM_`*var*. For example, `VFLIBCAP_PARAM_TeX_DPI` is defined, its value becomes the value of the vflibcap variable `TeX_DPI`.

The value of an environment variable `VFLIBCAP_PARAM_`*var* is parsed as an S-expression, not as an string. Thus, if you want to specify a string `ABC 123`, the value of an environment variablue must be `\"abc 123\"`. (Without double quotation, it will be a sequence of two strings. Only the first one is effective and the second one is ignored.)

6.9 VFlib defaults

To specify global behavior of VFlib, (virtual) font class `VFlib` is defined.

The following capability are defined.

`implicit-font-classes` (optional)
: — A list of implicit font classes. Font classes listed by this capability is candidates for implicit font searching.

  example: `(implicit-font-classes "bdf" "pcf" "gf")`

`extension-hints` (optional)
: — A list of paris of font name postfix and corresponding font class name. This is hint information to find font class from a font name in case of searching an implicit font. If an implicit font name matches with a postfix given by this capability, specified font class is invoked to search an implicit font. This is effective to reduce time to search an implicit font.

  example: `(extension-hints (".pcf" pcf) (".bdf" bdf) ("gf" gf))`

`variable-values` (optional)
: — A list of pairs of a name of vflibcap variable and its default value.

  example: `(variable-values ("TeX_DPI" "300") ("TeX_KPATHSEA_MODE" "cx") (v ("p1" "v1")`

`uncompression-programs` (optional)
: — A list of pairs of file name extension and corresponding uncompression program. This is used for reading compressed font files. An uncompression program must output uncompressed data to standard output.

  This capability is just defines relations of an extension and an uncompression program. A list of supported compressed types of a font class is given in a font class default description of each font class.

  example: `(uncompression-programs (".Z" "zcat") (".gz" "gzip -cd")`

`code-conversion-files` (optional)
: — A list of file names for encoding conversion. Currently, TrueType font class uses this. See Chapter 8 [Code conversion system], page 73.

  example: `(code-conversion-files "iso8859-1_unicode.ccv".ccv")`

`use-kpathsea` (optional)
: — A flag whether kpathsea is used or not to search TEX font files. Value of this capability must be one of `"Yes"` or `"No"`.

  example: `(use-kpathsea "Yes")`

`kpathsea-mode` (optional)
: — A device mode name for kpathsea library.

  example: `(kpathsea-mode "cx")`

`kpathsea-dpi` (optional)
: — Device resolution (in dpi) of a device mode for kpathsea library.

  example: `(kpathsea-mode 300)`

`kpathsea-program-name` (optional)
: — An application program name for kpathsea library.

  example: `(kpathsea-mode "xgdvi")`

6.10 BDF font class

The BDF format is a bitmap font format encoded in human-readable, platform independent format for distributing X Window fonts.

This font class supports compressed font files and implicit fonts.

Font class name: `bdf`

Capabilities for font class default:

`font-directories` (optional)
: — A list of font directories for searching font files. Recursive searching of font files is support.

`compression-extensions` (optional)
: — A list of supported compression type for this font class. This font class supports only compression type given by this capability. When a font is searched, a file followed by a compression extension is searched if given font file is not found. (Note that `uncompression-programs` capability of VFlib class default description gives a uncompression programs.)

  example: `(compression-extensions ".gz" ".Z")`

`dpi` (optional)
— Defualt device resolution. Default horizontal and vertical resolutions will be the same value.

example: `(dpi 300)`

`dpi-x` (optional)
— Default horizontal device resolution.

example: `(dpi-x 300)`

`dpi-y` (optional)
— Default vertical device resolution.

example: `(dpi-y 300)`

`aspect-ratio` (optional)
— Aspect ratio of characters. If this value is 0.5 then width is half, and if 2 then width is doubled.

example: `(aspect-ratio 0.8)`

`properties` (optional)
— A list of pairs of a property name and its value. Property values given by this parameter is used by `VF_GetProp()`

example: `(properties ("PROP-1" "value-1") ("PROP-2" "value-2"))`

`variable-values` (optional)
— A list of pairs of a vflibcap variable name and its default value.

example: `(variable-values ("TeX_DPI" "300") ("TeX_KPATHSEA_MODE" "cx") ("TeX_KPATHSEA_PROGRAM" "/usr/X11R6/xldvi"))`

Capabilities for font definition:

`font-class` (essential)
— A font class name. This value must be `bdf`.

`font-directories` (optional)
— A list of font directories for searching font files. Recursive searching of font files is support. A font file is searched in the directories given by this capability.

If this capability is not given, the font directory specified by the class default is used to search fonts. If this capability is given, the font directory specified by the class default is not used to search fonts.

`font-file` (optional)
— A font file name string. If this capability is not specified, the font name is used as the font file name. Multiple font file names can be listed in this capability. The driver tries to open a font listed first. If it is impossible to open, then it tries to open the second font. This is repeated until a font is successfully opened. If all fonts are impossible to open, font open fails.

example: `(font-file "timI24.bdf" "timR24.bdf")`

`point-size` (optional)
— font size in points. If the size is different from the size defined in the BDF font file, the bitmap is enlarged or shrinked to yield the specified size. This capability has effect for the VFlib functions `VF_GetBitmap1()` and `VF_GetMetric1()`.

example: `(point-size 24.0)`

`pixel-size` (optional)
: — font size in pixels. If the size is different from the size defined in the BDF font file, the bitmap is enlarged or shrinked to yield the specified size. This capability has effect for the VFlib functions `VF_GetBitmap2()` and `VF_GetMetric2()`.

  example: `(pixel-size 24)`

`magnification` (optional)
: — magnification factor. The font is magnified by this factor.

  example: `(magnification 1.20)`

`character-set` (optional)
: — This is used for code point conversion. Value of this capability gives an external view of a character set of a font. Code conversion (ccv) is determined by this value and the following three capabilities.

`encoding` (optional)
: — This is used for code point conversion. Value of this capability gives an external view of an encoding of a font.

`font-character-set` (optional)
: — This is used for code point conversion. Value of this capability gives an internal view of a character set of a font. Therefore, this value must match the character set of the font file given by `font-file` capability.

`font-encoding` (optional)
: — This is used for code point conversion. Value of this capability gives an internal view of an encoding of a font. Therefore, this value must match the encoding of the font file given by `font-file` capability.

  The following example defines a font named `iso8859_5-font` with ISO-8859-5 encoding by using a KOI8-R encoded font file.

```
(define-font iso8859_5-font
  (font-class pcf)
  (character-set "ISO8859-5")   (encoding "ISO")
  (font-character-set "KOI8-R") (font-encoding "KOI8-R")
  (font-file "koi8x13.pcf"))
```

  Code conversion is done by a subsystem named CCV. See Chapter 8 [Code conversion system], page 73 for detail.

6.11 PCF font class

Font class name: `pcf`

Other specification is the same as BDF font class except font class name is `pcf`.

6.12 HBF font class

Font class name: `hbf`

Other specification is the same as BDF font class except font class name is `hbf`.

6.13 TrueType font class

TrueType is a vector font font format. This font class supports implicit fonts but does not support compressed font files. TrueType font driver uses FreeType library version 1.2 developed by David Turner, Robert Wilhelm, and Werner Lemberg. See `http://www.freetype.org/` for detail.

Font class name: `truetype`

Capabilities for font class default:

`font-directories` (optional)
: — A list of font directories. This driver supports font file search by kpathsea. To search a font file by kpathsea, use `TEXMF` for a directory name.

`point-size` (optional)
`pixel-size` (optional)
`dpi` (optional)
`dpi-x` (optional)
`dpi-y` (optional)
`aspect-ratio` (optional)
`hinting` (optional)
: — If the value of capability `on`, "hinting" information is used to render characters. This is effective when small characters are rendered. If the value is `off`, hinting is disabled. Hinting information is used by default.

Note that enabling hinting has effect when obtaining bitmaps. It has no effect when you obtain and rasterize outline data in VFlib format, since VFlib outline format does not supports hinting information.

`platform-id` (optional)
: — A TrueType font can have multiple character code - glyph mapping tables. A mapping table is selected by specifying a pair of platform ID (`Microsoft`, `Macintosh`, etc) and encoding ID (`Unicode`, `Shift-JIS`, etc). This capability is used to specify platform ID of a mapping table to be selected.

Value of this capability is one of strings below:

`apple` Apple platform

`macintosh`, `mac`
Macintosh platform

`ascii`, `iso`
ISO platform

`microsoft`, `windows`, `ms`
Microsoft platform

`any`, `?`, `*` Any platform

Default value for this capability is Microsoft platform.

example: `(platform-id "microsoft")`

`encoding-id` (optional)
: — Together with platform id, this capability is used to specify a mapping table.

When ISO platform is selected by the `encoding-id` capability, value of this `encoding-id` capability is one of strings below:

`ascii` ASCII encoding.

`iso10464` ISO 10464 encoding.

`iso8859-1`
ISO8859-1 encoding.

`any`, `?`, `*` Any encoding.

When Apple platform is selected by the `encoding-id` capability, value of this `encoding-id` capability is one of strings below:

`unicode1.1`
Unicode 1.1 encoding.

`unicode2.0`
Unicode 2.0 encoding.

`iso10464` ISO 10464 encoding.

`any`, `?`, `*` Any encoding

When Microsoft platform is selected by the `encoding-id` capability, value of this `encoding-id` capability is one of strings below:

`symbol`

`unicode` Unicode encoding.

`shift-jis`, `sjis`, `ms-kanji`
Shift JIS encoding.

`big5` Big5 encoding.

`rpc`

`wansung`

`johab`

`any`, `?`, `*` Any encoding

When Macintosh platform is selected by the `encoding-id` capability, value of this `encoding-id` capability is one of strings below:

`roman`

`japanese`

`traditional-chinese`
`korean`

`arabic`

`hebrew`

`greek`

`russian`

`any`, `?`, `*` Any encoding

example: `(encoding-id "any")`

`properties` (optional)
`variable-values` (optional)

Capabilities for font definition:

`font-class` (essential)
This value must be "truetype".

`font-directories` (optional)
— A list of font directories for searching font files. Recursive searching of font files is support. A font file is searched in the directories by this capability. If not found, then a font is searched in a directories given by the class default.

To search a font file by kpathsea, use `TEXMF` for a directory name.

`font-file` (optional)
`point-size` (optional)
`pixel-size` (optional)
`dpi` (optional)
`dpi-x` (optional)
`dpi-y` (optional)
`magnification` (optional)
`aspect-ratio` (optional)
`hinting` (optional)
`font-number` (optional)
`encoding-force` (optional)
— In case encoding id data is broken in a TrueType font, its value can be overridden by this capability.

example `(encoding-force "unicode")`

`character-set` (optional)
— Together with `encoding` capability, this capability is used to change "external view" of a font. A font would be a font of a character set given by this capability and encoding given by `encoding` capability. Conversion of font internal character set and encoding to an external view is determined by these capability. Conversion is done by by code conversion system, called CCV. Chapter 8 [Code conversion system], page 73 Code conversion files are specified in `code-conversion-files` in `VFlib` font class default. See Section 6.9 [VFlib defaults], page 36.

For example, a font of JIS X 0208 character set (a Japanese character set) in Shift-JIS encoding fonts can be accessed as a JIS encoding font.

`encoding` (optional)
— Together with `character-set` capability, this capability defines a external view of a font. Chapter 8 [Code conversion system], page 73

`properties` (optional)
`jisx0212-row47-empty-sjis` (optional)
: — This capability is used for an ad-hoc solution to handle JIS X 0212 fonts with non-standard encoding such that row 47 is empty and followed rows are shifted by one. (That is, Kanjis in row 48 of JIS X 0212 appeard in row 49 in such fonts.) JIS X 0212 fonts of Ricoh TrueTypeWorld ValueFont DX are such fonts. If `yes` is given to this capability, buggy encoding is virtually fixed.

  This capability can apply to other products of buggy encoded JIS X 0212 fonts whose internal encoding is Shift JIS. (Use `ftdump` utility of FreeType package to check internal encoding scheme of fonts.)

6.14 Type1 font class

Type1 is a vector font font format used by PostScript. This font class supports implicit fonts but does not support compressed font files. This Type1 font driver uses T1Lib library version 5.1 or later developed by Rainer Menzner. See `http://www.neuroinformatik.ruhr-uni-bochum.de/ini/PEOPLE/rmz/t1lib/t1lib.html` for detail.

Be careful, VFlib does not work with old T1Lib. Obtain and install T1Lib 5.1 or later.

Currently, this font driver supports only 8-bit encoded fonts, i.e., it does not support for fonts of Japanese Kanji characters.

The function `VF_GetOutline()` for Type1 font files is supported but the result is ugly. Since T1Lib does not have a function to obtain outline data of a character in Type1 font, this font driver creates an outline data from a bitmap (for compatibility). Thus, it is very ugly. If your application software requires outline data, you are recommended to use the same font in other font format, such as TrueType. Thus, the outline obtained `VF_GetOutline()` function for Type1 font should be used only when the same font in other font format is not available.

Font class name: `type1`

Capabilities for font class default:

`font-directories` (optional)
: — A list of directories of Type1 font files. This driver supports font file search by kpathsea. To search a font file by kpathsea, use `TEXMF` for a directory name.

`afm-directories` (optional)
: — A list of directories of AFM files. Each element of this capability should not be a directory for recursive search (ending by `//`). This is why AFM files are searched by inside of T1Lib, although Type1 font files are searched by a file search subsystem of VFlib.

`encoding-vector-directories` (optional)
: — A list of directories of encoding files. Each element of this capability should not be a directory for recursive search (ending by `//`). This is why encoding vector files are searched by inside of T1Lib.

  By default, directories `/usr/local/share/VFlib/`*x.y.z*`/t1lib/` and `/usr/local/share/VFlib/site/t1lib/` are registered. Optional directories can be installed by this capability.

  T1Lib adopts file format for encoding vector file. When we want to use encoding vector files supplied by `dvips`, we must convert them into T1Lib format. To automate this, you can use a Unix Shell script `mkt1enc.sh` which is in `/usr/local/share/VFlib/`*x.y.z*`/t1lib/` directory. Encoding vector files for `dvips` are converted into T1Lib format and they are also installed in this directory.

`point-size` (optional)
`pixel-size` (optional)
`dpi` (optional)
`dpi-x` (optional)
`dpi-y` (optional)
`aspect-ratio` (optional)
`log-level` (optional)
— Select logfile output type of T1Lib. The filename of a logfile is '`t1lib.log`'. If this capability is not given, the logfile is not created.

`error` Only error messages are written to the logfile.

`warning` Warning messages and error messages are written to the logfile.

`statistics`
Statistics messages and above written to the logfile.

`debug` Any messages useful for debugging and above are written to the logfile.

`none` Never use a logfile.

See users manual of T1Lib for detail.

`properties` (optional)
`variable-values` (optional)

Capabilities for font definition:

`font-class` (essential)
This value must be "type1".

`font-file` (optional)
A list of font file names. Font file is searched in the listed order until existing font files is found.

example: `(font-file "AvantGarde-Book" "a0100131.pfb")`

By this example, '`AvantGarde-Book`' is seached first. If it exists, it is opened. Otherwise, '`a0100131.pfb`' is seached next. If it exists, it is opened. If it does not exist either, font open fails.

`encoding-vector` (optional)
A file name for encoding vector. This file must be reside in a directoy listed by `encoding-vector-directories` capability for type1 font class default.

`point-size` (optional)
`pixel-size` (optional)
`dpi` (optional)
`dpi-x` (optional)
`dpi-y` (optional)
`magnification` (optional)
`aspect-ratio` (optional)
`slant-factor` (optional)
— Slant factor of a font. This value is $\tan(th)$, where th is slant angle of a font. Default value is 0, in case of th is 90 degree.

example: `(slant-factor 0.2)`

`properties` (optional)

6.15 Zeit font class

This font class supports "*Syotai Kurubu*" format; it is a vector font format for Japanese Kanji characters.

Several free Japanese fonts in this file format are available.

- Umeda vector fonts
  - `ftp://ftp.web.ad.jp/pub/TeX/akiu/fonts/umeda-vector/mincho.vf1.gz`
  - `ftp://ftp.web.ad.jp/pub/TeX/akiu/fonts/umeda-vector/mincho.vf2.gz`
  - `ftp://ftp.eos.hokudai.ac.jp/pub/TeX/fonts/umeda-vector-font/mincho.vf1.gz`
  - `ftp://ftp.eos.hokudai.ac.jp/pub/TeX/fonts/umeda-vector-font/mincho.vf2.gz`
- Watanabe vector fonts
  - `ftp://ftp.web.ad.jp/pub/TeX/akiu/fonts/watanabe-vector/mincho.vf1.gz`
  - `ftp://ftp.web.ad.jp/pub/TeX/akiu/fonts/watanabe-vector/mincho.vf2.gz`
  - `ftp://ftp.eos.hokudai.ac.jp/pub/TeX/fonts/watanabe-vector-font/mincho.vf1.gz`
  - `ftp://ftp.eos.hokudai.ac.jp/pub/TeX/fonts/watanabe-vector-font/mincho.vf2.gz`
- Wadalab fonts
  - `ftp://ftp.web.ad.jp/pub/TeX/akiu/fonts/wadalab-vector/`
  - `ftp://ftp.eos.hokudai.ac.jp/pub/TeX/fonts/wadalab-vector-font/`
  - See also `ftp://ftp.ipl.t.u-tokyo.ac.jp/Font/` for Free Japanese Kanji fonts in Type 1 formats.

This font class does not support compressed font files nor implicit fonts.

Font class name: `zeit`

Capabilities for font class default:

`font-directories` (optional)
`filename-extensions` (optional)
— Two file files form a font for this font class, e.g., `mincho.vf1` and `mincho.vf2`. Extension candidates without digit must be the value for this capability.

example: `(filename-extensions ".vf" ".VF")`

`point-size` (optional)
`pixel-size` (optional)
`dpi` (optional)
`dpi-x` (optional)
`dpi-y` (optional)
`aspect-ratio` (optional)
`writing-direction` (optional)

— Default writing direction. `Horizontal` or `vertical`; default is `Horizontal`. This is the same as `(vector-to-bbx-upper-left 0.0 0.86)` and `(vector-to-next-ref-point 1.0 0.0)` if value of this capability is "horizontal". Otherwise, it is the same as `(vector-to-bbx-upper-left -0.5 0.0)` and `(vector-to-next-ref-point 0.0 -1.0)`.

`vector-to-bbx-upper-left` (optional)

— Default value of a vector from the reference point of a character to upper-left bounding box. Right and up are positive directions.

example: `(vector-to-bbx-upper-left 0 0.86)`

`vector-to-next-ref-point` (optional)

— Default value of a vector from the reference point to next reference point.

example: `(vector-to-next-ref-point 1.0 0.0)`

`properties` (optional)
`variable-values` (optional)

Capabilities for font definition:

`font-class` (essential)

— A font class name.

`font-name` (optional)

— Font file name without extensions. Together with vale and extension given by default description, font file names are formed. For example, if `mincho` is given for the `font-name` capability and `(".vf")` is given for the `filename-extensions` capability, then font files `mincho.vf1` and `mincho.vf2` are used.

example: `(font-name "mincho")`

`point-size` (optional)
`pixel-size` (optional)
`dpi` (optional)
`dpi-x` (optional)
`dpi-y` (optional)
`magnification` (optional)
`aspect-ratio` (optional)
`writing-direction` (optional)
`vector-to-bbx-upper-left` (optional)

example: `(vector-to-bbx-upper-left 0 0.86)`

`vector-to-next-ref-point` (optional)

example: `(vector-to-next-ref-point 1.0 0.0)`

`properties` (optional)

6.16 JG font class

JG font format is a vector font format for Japanese character sets JIS X 0208. JG font driver is based on the work by Hideo Morishita.

Font class name: `jg`

Capabilities for font class default:

`filename-extensions` (optional)
: — Three files form a font for this font class, e.g., `mincho.fn0`, `mincho.fn1`, and `mincho.fn2`. Extension candidates without digit must be the value for this capability.

  example: `(filename-extensions ".fn" ".FN")`

(Other capabilities are the same as `zeit` font class.)

Capability for font definition:

Capabilities are the same as `zeit` font class.

6.17 eKanji font class

The eKanji font format is a bitmap font format for Kanji characters. The first character in an eKanji font file has code point 1, and the second character has code point 2. That is, characters in an eKanji font are numbered sequentially starting from 1. (This encoding scheme can be changed by setting some capabilities.)

eKanji font files are distributed at the following URL: `http://www.zinbun.kyoto-u.ac.jp/~ekanji/` The distribution package contains the following font files.

- Unicode (‘`ekan0010.d24`’)
- Kyoto University KangXi (‘`ekan0020.d24`’)
- Morohashi DaiKanwa (‘`ekan0030.d24`’)
- JIS X 0208 (‘`jisx9052.d24`’)

Font class name: `ekanji`

Capabilities for font class default:

`font-directories` (optional)
`font-dot-size` (optional)
: — Dot size of characters in the font file. Default value is 24.

  example: `(font-dot-size 24)`

`point-size` (optional)
`pixel-size` (optional)
`dpi` (optional)
`dpi-x` (optional)
`dpi-y` (optional)
`aspect-ratio` (optional)
`writing-direction` (optional)
: — Default writing direction. `Horizontal` or `vertical`; default is `Horizontal`. This is the same as `(vector-to-bbx-upper-left 0.0 0.92)` and `(vector-to-next-ref-point 1.0 0.0)` if value of this capability is "Horizontal". Otherwise, it is

the same as `(vector-to-bbx-upper-left -0.5 0.0)` and `(vector-to-next-ref-point 0.0 -1.0)`.

`vector-to-bbx-upper-left` (optional)
: — Default value of a vector from the reference point of a character to upper-left bounding box. Right and up are positive directions.

  example: `(vector-to-bbx-upper-left 0 0.90)`

`vector-to-next-ref-point` (optional)
: — Default value of a vector from the reference point to next reference point.

  example: `(vector-to-next-ref-point 1.0 0.0)`

`properties` (optional)
`variable-values` (optional)

Capability for font definition:

`font-class` (essential)
: — A font class name.

`font-name` (optional)
: — Font file name with extension.

`font-dot-size` (optional)
: — Dot size of characters in the font file.

`point-size` (optional)
`pixel-size` (optional)
`dpi` (optional)
`dpi-x` (optional)
`dpi-y` (optional)
`magnification` (optional)
`aspect-ratio` (optional)
`character-set` (optional)
`encoding` (optional)
`font-character-set` (optional)
`font-encoding` (optional)
: — Select code conversion. If you want to access a eKanji font by ISO-2022 (JIS) encoding scheme, define `eKanji` for `character-set`, `ISO2022` for `encoding`, `eKanji` for `font-character-set`, and `SEQUENTIAL2-1` for `encoding`. Then the first character in the eKanji font is accessed by code point 0x2121.

`mock-font-encoding` (optional)
: — By this capability, encoding of an eKanji font file is virtually changed. This capability requires an argument and parameter. There are three keywords for an argument:

  - `raw` No effect, i.e., font encoding is not changed.
  - `subblocks-94x94` B An eKanji font file is virtually divided by 94x94 sub blocks (blocks of 94x94 = 8836 characters) and selects B-th block for this font definition. This implies that an eKanji font file with this capability offers only 8836 (= 94x94) characters among all the characters of an eKanji font file. The first sub block is numbered zero. (A font with $B = 0$ for this capability selects the first sub block.)

    Characters with character code from $8836 * B + 1$ to $8836 * B + 8836$ in an eKanji font file is accessed by character code from 0x2121 to 0x7e7e.

- `subblocks-94x60` *B* This is similar to the case for `subblocks-94x60`. An eKanji font file is virtually divided by 94x60 sub blocks (blocks of 94x60 = 5640 characters) and selects *B*-th block for this font definition. The first sub block is numbered zero.

  Characters with character code from $5640 * B + 1$ to $5640 * B + 5640$ in an eKanji font file is accessed by character code from 0x3021 to 0x4e7e (first 30 rows) and from 0x5021 to 0x6e7e (another 30 rows). (This division scheme is the same as Mojikyo scheme.)
- `with-offset` *OFFS* An offset value *OFFS* is added to obtain character code of a character in an eKanji font file. When *OFFS* is `-0x4dff`, the first character in an eKanji font file is accessed by code number 0x4e00, since $0x4e00 + (-0x4dff) = 1$.

Theoretically, the same functionality shown above can be implemented within the CCV subsystem framework Chapter 8 [Code conversion system], page 73, the eKanji font driver defines this capability, since the eKanji font file format adopts curious character encoding scheme and it seems to be simpler and cleaner to define this capability.

`writing-direction` (optional)
`vector-to-bbx-upper-left` (optional)
: example: `(vector-to-bbx-upper-left 0 0.86)`

`vector-to-next-ref-point` (optional)
: example: `(vector-to-next-ref-point 1.0 0.0)`

`properties` (optional)

6.18 TEX default and TEX font mapping font class

This is a special font class to define common default values of TEX-related font classes.

This font class has a feature to map a requested font to another font. Thus, this class is also called "*TEX font mapping class*" or "*TEX font mapper*". TEX-related font classes include the followings: GF, PK, TFM, VF, ASCII Japanese TeX.

This font class supports only implicit fonts and explicit fonts is not supported. Therefore, the driver name of this font class must be given in `extension-hints` and/or `implicit-font-classes` capabilities in the `VFlib` default description if you want to use the font mapping feature of this class.

Mapped font is recursively (recursively) requested to open and any operations such as obtaining bitmaps on the requested font are applied on the mapped font.

Font class name: `TeX`

Capabilities for font class default:

`dpi` (optional)
: — Default device resolution for TEX-related fonts.

`tfm-directories` (optional)
: — A list of directories of TFM files. Directories listed by this capability is used for searching TFM files. If a directory name is `TEXMF`, kpathsea is invoked to search a file.

  example: `(tfm-directories "TEXMF" "/usr/local/share/font/tfm//")`

`tfm-filename-extensions` (optional)
— A list of extensions of filenames for TFM files. This is used to construct a TFM file name, e.g., "cmr10.tfm" for a font "cmr10".

example: `(tfm-filename-extensions "tfm")`

`font-mapping` (optional)
— Font mapping rules are described in this capability. When a font is requested to open (as an implicit font), the font name is mapped to another name and specified font driver is requested to open the mapped font.

Syntax of this capability is as follows:

```
(font-mapping
  ((DRIVER-NAME MAPPING-FORMAT FONT-OPEN-OPTIONS) ...
   FONT-NAME FONT-NAME ...)
  ((DRIVER-NAME MAPPING-FORMAT FONT-OPEN-OPTIONS) ...
   FONT-NAME FONT-NAME ...)
  ...)
```

Thus, value for this capability is a sequence of items `((`*`DRIVER-NAME MAPPING-FORMAT FONT-OPEN-OPTIONS`*`) ...` *`FONT-NAME FONT-NAME`* `...)`, and this forms a mapping rule. *FONT-NAME* is a font name and this rule applies if requested font name matches *FONT-NAME*. (*FONT-NAME* is a name without directory and extension parts. A font name requested to open is compared with *FONT-NAME* by deleting directory and extension parts.) If *FONT-NAME* contains `*` character, it matches to the rest of requested font name. For example, `cm*` matches `cmr10` and `cmbx10`. Thus, in case *FONT-NAME* is `*`, all fonts matches and thus all fonts applies the rule.

The font name is mapped according to the format *MAPPING-FORMAT* and mapped name is opened by calling a font driver *DRIVER-NAME*. `VF_OpenFont1()` and `VF_OpenFont2()` are not used. (As a special case, when *DRIVER-NAME* is `*`, `VF_OpenFont1()` or `VF_OpenFont2()` is used to open a mapped font.)

The syntax of *MAPPING-FORMAT* is similar to a format string of `printf()` function in C library, but conversion characters and semantics are different. Conversion specification is introduced by `%` character. Non-conversion characters are simply copied and conversion specifications are substituted for the following:

`%%` — `%` character

`%f` — the requested font name without extension and directory parts.

`%d` — font file resolution part in the extension of the requested font name

`%e` — file format part in the extension of the requested font name

For instance, let `/foo/bar/qwe.300pk` is the requested font name. Then `%f` is `qwe`, `%d` is `300`, and `%e` is `pk`. A conversion specification will be null string if corresponding substring does not exist.

In general, mapped fonts are opened with the same parameters (device resolution, magnification factors, point or pixel size) of the requested font. Such parameters can be changed by optional *FONT-OPEN-OPTIONS* part. Following descriptions can be used for *FONT-OPEN-OPTIONS*:

We can specify multiple `(`*`DRIVER-NAME MAPPING-FORMAT FONT-OPEN-OPTIONS`*`)` in a rule description. This is useful if we need to write multiple mapping rules for the same set of fonts.

A font is opened by the following way.

1. For each rule (from the first one to the last one), the requested font name is checked if the rule applies to the font. If the rule does not apply, check next rule.
2. For each (*DRIVER-NAME MAPPING-FORMAT FONT-OPEN-OPTIONS*) in the rule, the requested font is mapped and font open is attempted. If a mapped font is is successfully opened, it is used as a requested font. Otherwise, next mapping (*DRIVER-NAME MAPPING-FORMAT FONT-OPEN-OPTIONS*) is attempted. This is repeated for a mapped font is opened.
3. If mapped fonts are not opened, next rule is checked. This is repeated for all rules until a mapped font is opened.

example:

```
(font-mapping
 ((ascii-jtex-kanji "%f.jtex")
  "min*" "goth*" "tmin*" "tgoth*")
 ((type1 "%f.pfb" point-size-from-tfm (magnification-adjustment 1.0))
  *)
 ((pk "%f.%dpk") (gf "%f.%dgf") (tfm "%f.tfm")
  *))
```

For this example, suppose that `min10.300pk` is requested to open.

1. The first rule applies to the requested font since `min*` is in the font list. The `ascii-jtex-kanji` driver is invoked to open a mapped name `min10.jtex`. If it is opened, font open succeeds; `min10.jtex` is used as `min10.300pk` and font open finishes. If the font is not opened, continue to the next step.
2. The second rule applies to the requested font since `*` is given in the font list. The `type1` driver is invoked to open a mapped name `min10.pfb`. If it is opened, `min10.jtex` is used as `min10.300pk` and font open finishes. If the font is not opened, continue to the next step.
3. The second rule applies to the reqiested font since `*` is given in the font list.
   1. The `pk` driver is invoked to open a mapped name `min10.300pk`. If it is not opened, next mapping is attempted.
   2. The `gf` driver is invoked to open a mapped name `min10.300gf`. If it is not opened, next mapping is attempted.
   3. The `tfm` driver is invoked to open a mapped name `min10.tfm`.

If everything above fails, font open for `cmr10.300pk` fails.

- `point-size-from-tfm` (optional) — When a mapped font is opened (in mode 1, high resolution oriented mode), point size which is obtained from a TFM file is given. This is necessary when we use TrueType and/or Type1 fonts for mapped fonts. For example, "cmr10.ttf" and "cmr10.pfb" in the BaKoMa font set do not have point size information, since TrueType and Type1 format fonts cannot have information on point size.
- `(magnification-adjustment` *mag*`)` (optional) — Mapped fonts are opened with magnification factors magnified by *mag*. This can be used to adjust size of mapped fonts. But most of the case, this is not necessary.

As a restriction of fonts of this class, each font must have a TFM file.

`resolution-accuracy` (optional)
`resolution-corrections` (optional)

— According to arithmetic errors, DPI value for font files and computed value (= device resolution times magnification value) may be different. These two capabilities give correct resolution values for PK and GF fonts.

Syntax of these capabilities are as follows:

```
(resolution-accuracy ACCURACY)
(resolution-corrections
  (DEVICE-RESOLUTION FONT-RESOLION FONT-RESOLION ... )
  (DEVICE-RESOLUTION FONT-RESOLION FONT-RESOLION ... )
   ...)
```

DEVICE-RESOLUTION is the resolution of target device in DPI and *FONT-RESOLUTION* is a font resolution value.

To find a font file, font resolution is computed by device resolution times magnification factor. Then, this driver finds a list `(DEVICE-RESOLUTION FONT-RESOLUTION FONT-RESOLUTION ... )` such that *DEVICE-RESOLUTION* is the same as resolution of target device. (If there is no such list is the capability value, font file resolution is not corrected and computed value is used.)

For each *FONT-RESOLUTION* in the list, the driver checks if the computed font file resolution is in the range from *FONT-RESOLUTION* times (1-*ACCURACY*) to *FONT-RESOLUTION* times (1+*ACCURACY*). If the computed resolution is in the range, font file resolution is changed to *FONT-RESOLUTION*. Then, a font file is searched by the corrected font file resolution.

example:

```
(resolution-accuracy 0.02)
(resolution-corrections
 (300  ;; cx
  300  329  360  432  518  622  746  896 1075 1290  240 270)
 (600  ;; ljfour
  600  657  720  864 1037 1244 1493 1792 2150 2580  480 540))
```

Be careful not to map to the same name! Otherwise, font open will be an infinite loop. (VFlib restricts the depth of nested font open. Even if the font name is mapped to the same name, VFlib will detects an error, anyway.)

`properties` (optional)
`variable-values` (optional)

6.19 PK font class

PK fonts are bitmap fonts used by TeX system. PK font driver provides a logical view of PK fonts when a font is requested to open as an implicit font; if a given font filename is `cmr10.pk`, requested device resolution is 300 and magnification is 1.2, then PK font driver looks for a font file `cmr10.360pk`. Thus, font names (for font open) should not be the same as font filenames on filesystems.

To search a font file, the kpathsea library can be used. A special name `TEXMF` in a list of font directories (capability `font-directories`) is used to search a file by kpathsea.

To enable kpathsea, the value for `use-kpathsea` capability in `VFlib` class default must be `"Yes"`.

This font class supports compressed font files and implicit fonts.

Font class name: `pk`

Capabilities for font class default:

`font-directories` (optional)
: — A list of directory names for font files. This driver supports font file search by kpathsea. To search a font file by kpathsea, use `TEXMF` for a directory name.

`filename-extensions` (optional)
: — A list of extensions of filenames for PK fonts. This is used to construct a font file name "cmr10.300pk" from "cmr10.pk" for 300 dpi fonts.

  example: `(filename-extensions "pk")`

`make-missing-glyph` (optional)
: — Generate a PK font file from Metafont source file by running METAFONT on the fly, if a requested PK font file does not exist. example: `(make-missing-glyphs "yes")`

`point-size` (optional)
`pixel-size` (optional)
`dpi` (optional)
`properties` (optional)
`variable-values` (optional)

Capabilities for font definition:

`font-class` (essential)
: — A font class name.

`font-file` (optional)
: — a font filename. An extension (e.g., `.300pk`) can be omitted. In case of this capability is not given, font name is used as this capability value.

`point-size` (optional)
`pixel-size` (optional)
`dpi` (optional)
`properties` (optional)
`magnification` (optional)

6.20 GF font class

Font class name: `gf`

Other capabilities are the same as ones for `pk` font class.

6.21 TFM font class

TFM files contains only metric information for typesetting TeX documents. This font class provides fonts whose metrics are defined by TFM font files. Since TFM files do not have glyph, glyph of a font of this font class are (black or white) rectangles. Fonts of this font class can be used as substitutes of PK, GF, or VF files in case they are missing.

If the file is not found, it is searched by `font-directories` capability given in `TFM` font class default description. Note that the extension of font files (given by the `filename-extensions` capability) has no effect for searching by kpathsea library. The extension of font files must be ".tfm". See kpathsea manual for detail.

This font class supports compressed font files and implicit fonts.

Font class name: `tfm`

Capabilities for font class default:

`font-directories` (optional)
: — A list of directory names for font files. This driver supports font file search by kpathsea. To search a font file by kpathsea, use `TEXMF` for a directory name.

`filename-extensions` (optional)
`glyph-style` (optional)
: — Defines default glyph style: `empty` or `fill`. If `empty` is given, all glyph of a font are white rectangles. If `fill` is given, all glyph of a font are black rectangles.

`point-size` (optional)
`pixel-size` (optional)
`dpi` (optional)
`properties` (optional)
`variable-values` (optional)

Capabilities for font definition:

`font-class` (essential)
: — A font class name.

`font-file` (optional)
`glyph-style` (optional)
: — Defines glyph style: `empty` or `fill`. If `empty` is given, all glyph of a font are white rectangles. If `fill` is given, all glyph of a font are black rectangles.

`point-size` (optional)
`pixel-size` (optional)
`dpi` (optional)
`magnification` (optional)
`aspect-ratio` (optional)
`properties` (optional)

6.22 VF font class

This font class handles *Virtual Font* files. A virtual font consists of a font program and subfonts. A font program in a vf file is similar to DVI file formats. Glyph of a virtual font are constructed from a box instruction in a font program and glyph taken from subfonts. Therefore, this font class requires a font mapping rule to obtain glyph from subfonts. You can specify VFlib not to open subfonts of a virtual font in case of subfonts are unavailable.

Font class name: `vf`

Capabilities for font class default:

`font-directories` (optional)
: — A list of directory names for font files. This driver supports font file search by kpathsea library. To search a font file by kpathsea library, use `TEXMF` for a directory name.

`filename-extensions` (optional)
: — A extension string for virtual font files.

  example: `(filename-extensions "vf")`

`tfm-directories` (optional)
`tfm-filename-extensions` (optional)
`font-mapping` (optional)
: — A set of rules for mapping for subfonts to open as an VFlib fonts. This is the same as TeX font mapper, i.e., `font-mapping` capability for `TeX` font class default description.

  Be careful not to map to the same name! Otherwise, font open will be an infinite loop. (VFlib restricts the depth of nested font open. Even if the font name is mapped to the same name, VFlib will detects an error, anyway.)

`open-style` (optional)
: — This capability specifies how subfonts are opened.

  `none` — Boxes are used instead of glyph of subfonts. Subfonts are not opened.

  `try` — The virtual font driver tries to open each subfont. If subfonts are opened, glyph are taken from opened subfonts. If some subfonts are not opened, boxes are used instead of glyph of such subfonts. It is not an error even if all subfonts are not opened.

  `require` — The virtual font driver tries to open each subfont. It is an error if every subfont is not opened.

`glyph-style` (optional)
: — In case boxes are used instead of glyph of subfonts, this capability controls the looks of boxes.

  `empty` — Boxes are white, i.e., all pixels are value 0.

  `fill` — Boxes are black, i.e., all pixels are value 1.

`point-size` (optional)
`pixel-size` (optional)
`dpi` (optional)
`properties` (optional)
`variables` (optional)
`debug` (optional)

Capabilities for font definition:

`font-file` (optional)
: — File name of a virtual font.

`point-size` (optional)
`pixel-size` (optional)
`dpi` (optional)
`magnification` (optional)
`properties` (optional)

6.23 ASCII Japanese TEX Kanji font class

This is for *Kanji* fonts of Japanese TEX localized by ASCII Co. This font driver provides *Kanji* fonts that can be accessed as if they were PK fonts by using another VFlib font. Font metrics of *Kanji* characters defined by ASCII jTEX may not match that of a *Kanji* font to be used. This case happens when `jiskan24.pcf` font is used as a Japanese *Kanji* font for TEX. This font driver works as a *filter* that modifies font metrics of another font. Change of font metrics is defined by an external file called `adjustment file`. See sample distribution of adjustment files for their syntax. (Not documented now...)

Font metrics of a font of this class is defined by a TFM font; a vector to the next reference point is taken from a TFM file. A vector to upper left corner of a bitmap is taken from subfont. Then, font metrics is modified according to an adjustment file.

This driver supports vertical writing. In case fonts for vertical writing are not available, the driver rotates glyph of some characters, e,g, parenthesis, to yield (possible) correct glyph.

Font class name: `ascii-jtex`

Capabilities for font class default:

`implicit-font-mapping-suffix` (optional)
: — A suffix to map a font name for searching an implicit font. Suppose a font is requested to open. Then, font name is mapped in such a way that extension is deleted and the suffix given by this capability is added. Then, an vflibcap entry of the mapped name is searched. If such an entry exists, it is used for the implicit font and the font is opened as if it were an explicit font. (Note that a font of the mapped name must exist as an explicit font, not as an implicit font.)

  Suppose `min10.400pk` is requested to open as an implicit font and the value of this capability is `.jtex`. Then mapped name is `min10.jtex` and it is opened by this font driver internally.

  example: `(implicit-font-mapping-suffix ".jtex")`

`tfm-directories` (optional)
`tfm-filename-extensions` (optional)
`properties` (optional)
`variable-values` (optional)

Capabilities for font definition:

`font-class` (essential)
: — A font class name.

`kanji-font` (optional)
: — Font name for a subfont.

`kanji-font-point-size` (optional)
: — Default point size of subfont. If point size is not explicitly given when a font is opened, This value is used.

`kanji-font-pixel-size` (optional)
: — Default pixel size of a subfont. If pixel size is not explicitly given when a font is opened, This value is used.

`kanji-font-magnification` (optional)
: — magnification factor for subfont.

`tfm-file` (optional)
: — TFM file that defines font metrics of a font.

  example: `(tfm-file "min10.tfm")`

`metric-adjustment-file` (optional)
: — a file name for font metric adjustment file.

`properties` (optional)

6.24 Japanese comic font class

This font driver composes two Japanese *Kanji* fonts. According to code point (*Kana* or *kanji* character), one of two font is selected to obtain a bitmap or metric. This font class provides Japanese fonts that composes *Kana* and *Kanji* font.

Font class name: `japanese-comic`

Capabilities for font class default:

`point-size` (optional)
`pixel-size` (optional)
`dpi` (optional)
`properties` (optional)
`variable-values` (optional)

Capabilities for font definition:

`font-class` (essential)
: — A font class name.

`kanji-font` (optional)
: — A VFlib font name for Kanji characters. For code points for Kanji characters, this font is used to obtain bitmaps and metrics.

`kana-font` (optional)
: — A VFlib font name for Kana font (code point: 0x2121 ... 0x287f). For code points except Kanji characters, this font is used to obtain bitmaps and metrics.

`symbol-font` (optional)
: — A VFlib font name for symbol characters (code point: 0x2121 ... 0x227f). If this capability is not given, a font given by `kana-font` is used for symbol characters.

`alpha-numeric-font` (optional)
: — A VFlib font name for alphabet and numeric characters (code point: 0x2321 ... 0x237f). If this capability is not given, a font given by `kana-font` is used for alphabet and numeric characters.

`hirakana-font` (optional)
: — A VFlib font name for Hirakana characters (code point: 0x2421 ... 0x247f). If this capability is not given, a font given by `kana-font` is used for Hirakana characters.

`katakana-font` (optional)
: — A VFlib font name for Katakana characters (code point: 0x2521 ... 0x257f). If this capability is not given, a font given by `kana-font` is used for Katakana characters.

`greek-font` (optional)
— A VFlib font name for Greek characters (code point: 0x2621 ... 0x267f). If this capability is not given, a font given by `kana-font` is used for Greek characters.

`cyrillic-font` (optional)
— A VFlib font name for Cyrillic characters (code point: 0x2721 ... 0x277f). If this capability is not given, a font given by `kana-font` is used for Cyrillic characters.

`keisen-font` (optional)
— A VFlib font name for Keisen characters (code point: 0x2821 ... 0x287f). If this capability is not given, a font given by `kana-font` is used for Keisen characters.

`point-size` (optional)
`pixel-size` (optional)
`dpi` (optional)
`magnification` (optional)
`properties` (optional)

6.25 Try font class

A font of this font class has a list of *sub-fonts*. When a font of this class is requested open, the try font driver tries to open the sub-fonts one after another until one of them is successfully opened. If all sub-fonts in the list are not opened, the font is failed to be opened. All font operation of the font is applied to an opened sub-font.

Font class name: `try`

Capabilities for font class default:

`properties` (optional)
`variable-values` (optional)

Capabilities for font definition:

`font-class` (essential)
— A font class name. This must be `try`.

`font-list` (optional)
— A list of sub-fonts. These fonts are VFlib fonts, not a font file names.

`point-size` (optional)
`pixel-size` (optional)
`dpi` (optional)
`magnification` (optional)
`properties` (optional)

6.26 Mojikyo font mapping class

This font driver is specific to the Mojikyo font files, which is a huge collection (more than 80 thousand) of Kanji characters. (Access `http://www.mojikyo.gr.jp/` for detail.) The Mojikyo font is supplied by a set of font files, since the number of characters is too huge to contain in a single font file.

The Mojikyo font defines its own character encoding, staring from 1 and each character has its own character code. (Character codes are not the codes in a font file.) The Mojikyo font set is supplied by both TrueType and Type 1 formats. A single character space of the Mojikyo is divided into font file number and character code in a font file. This means that we must compute font file among many font files and code point in a font file to obtain a glyph of

Mojikyo characters. To avoid such complex procedure, this font driver provides a virtual single font.

Note that this font driver only delegates requested characters to other font driver (TrueType or Type 1). Therefore, TrueType and/or Type 1 font driver must be configured in VFlib and they must be propopery set up in a vflibcap file.

Font class name: `mojikyo-mapper`

Capabilities for font class default:

`properties` (optional)
`variable-values` (optional)

Capabilities for font definition:

`font-class` (essential)
: — A font class name. This must be `mojikyo-mapper`.

`division-scheme` (optional)
: Mapping scheme from the Mojikyo character space to real font files is different by real font file format (TrueType/Type1). This capability defines which mapping scheme is used.

  If `truetype` is given for this capability, underlaying font files are in TrueType format. If `type1` is given for this capability, underlaying font files are in Type 1 format. Default value for this capability is `truetype`.

  Aliases of division scheme names are defined as follows: `ttf` is an alias of `truetype`, and `pfb` is an alias of `type1`.

`subfont-name-format` (optional)
: This capability defines format of font file names. If `truetype` is selected for `division-scheme` capability, `Mojik%d.ttf` is assumed for this capability by default. `%d` in `Mojik%d.ttf` is substituted by font number, starting from 101. If `type1` is selected for `division-scheme` capability, `mo%dm%02d.pfb` is assumed for this capability by default. The first `%d` in `mo%dm%02d.pfb` is a major font number, starting from 101. The second `%d` in `mo%dm%02d.pfb` is a minor font number, starting from 6.

  In case you want to use a font name format other than described above, this capability should be defined. Note that format sting should contain exactly one `%d` if you select `truetype` division scheme, and exactly two `%d`s if you select `type1` division scheme.

`truetype-subfont-encoding` (optional)
: This capability has effect only when `truetype` division scheme is selected. This capability selects character encoding scheme of underlaying TrueType font files. If `unicode` is given to this capability, underlaying TrueType fonts are encoded in Unicode. If `iso-2022` (as aliases, `iso2022` or `jis` can be used) is given to this capability, underlaying TrueType fonts are encoded in ISO 2022 (JIS).

  Default value is `unicode`, which is the same as the the Mojikyo font files in TrueType format.

`properties` (optional)

6.27 Example vflibcap 1

This example vflibcap is for general use.

```
;; -----------------------------------------------------------------
;;   VFlib Default
;; -----------------------------------------------------------------
(define-default  VFlib
  ;; hint to find font class from font name for fast font open
  (extension-hints  (".bdf" bdf) (".pcf" pcf) (".hbf" hbf)
                    (".ttf" truetype) (".ttc" truetype)
                    (".pfa" type1) (".pfb" type1)
                    ("pk" TeX) ("gf" gf) ("tfm" tfm))
  ;; implicit font classes
  (implicit-font-classes  pcf bdf hbf truetype type1 zeit jg gf tfm)

  ;; uncompression programs
  (uncompression-programs  (".Z" "zcat") (".gz" "gzip -cd"))

  ;; a list of default values of variables
  ;; *Note* "variable-values" must come before variable uses
  (variable-values  (TeX_DPI               "300")
                    (TeX_USE_KPATHSEA      "Yes")
                    (TeX_KPATHSEA_MODE     "cx")
                    (TeX_KPATHSEA_PROGRAM  "/usr/local/bin/xldvi"))

  ;; kpathsea: enabled/disabled
  (use-kpathsea           $TeX_USE_KPATHSEA)
  ;; kpathsea mode (e.g., "cx")
  (kpathsea-mode          $TeX_KPATHSEA_MODE)
  ;; kpathsea program name (e.g., "/usr/local/bin/xdvi")
  (kpathsea-program-name  $TeX_KPATHSEA_PROGRAM)

  ;; encoding/charset conversion files
  (code-conversion-files
   "iso8859-1_unicode.ccv" "iso8859-2_unicode.ccv" "iso8859-3_unicode.ccv"
   "iso8859-4_unicode.ccv" "iso8859-5_unicode.ccv" "iso8859-6_unicode.ccv"
   "iso8859-7_unicode.ccv" "iso8859-8_unicode.ccv" "iso8859-9_unicode.ccv"
   "jisx0201_unicode.ccv" "jisx0208_unicode.ccv" "jisx0212_unicode.ccv"
   "ksc5601_unicode.ccv"
   "gb12345_unicode.ccv" "gb2312_unicode.ccv"
   "big5_unicode.ccv" "cns11643_unicode.ccv"
   "iso8859-5_koi8-r.ccv" "koi8-r_iso8859-5.ccv" "koi8-r_unicode.ccv"))

;; -----------------------------------------------------------------
;;   BDF Font Class Default
;; -----------------------------------------------------------------
(define-default  bdf
  ;; font directories
  (font-directories  "/usr/local/share/fonts/X11//")
  ;; extensions of compressed font files that this font class supports
  (compression-extensions ".gz" ".Z")
  ;; default values for fonts of this font class
  (variable-values    (VAR1 ("PROP1" "VAL1"))  ; just for debugging...
                      (VAR2 ("PROP2" "VAL2"))
                      (VARX ("PROPX" "VFlib-VALX")) )
```

```
  ;; properties for all fonts of this font class
  (properties ("FONT_CLASS" "BDF")) )

;; -----------------------------------------------------------------
;;   PCF Font Class Default
;; -----------------------------------------------------------------
(define-default  pcf
  ;; font directories
  (font-directories  "/usr/X11R6/lib/X11/fonts//"
                     "/usr/local/X11R6/lib/X11/fonts//"
                     "/usr/openwin/lib/X11/fonts//"
                     "/usr/X386/lib/X11/fonts//"
                     "/usr/XFree86/lib/X11/fonts//"
                     "/usr/X11/lib/X11/fonts//"
                     "/usr/local/lib/X11/fonts//"
                     "/usr/X11R5/lib/X11/fonts//"
                     "/usr/local/X11R5/lib/X11/fonts//"
                     "/usr/local/share/fonts/X11//")
  ;; extensions of compressed font files that this font class supports
  (compression-extensions ".gz" ".Z")
  ;; properties for all fonts of this font class
  (properties ("FONT_CLASS" "PCF"))  )

;; -----------------------------------------------------------------
;;   HBF Font Class Default
;; -----------------------------------------------------------------
(define-default  hbf
  ;; font directories
  (font-directories  "/usr/local/share/fonts/HBF//")
  ;; extensions of compressed font files that this font class supports
  (compression-extensions ".gz" ".Z")
  ;; properties for all fonts of this font class
  (properties ("FONT_CLASS" "HBF"))  )

;; -----------------------------------------------------------------
;; TrueType Font Class Default
;; -----------------------------------------------------------------
(define-default  truetype
  ;; font directories
  (font-directories  "TEXMF"  ; - a special name to search by ‘kpathsea’
                     "/usr/local/share/fonts/bakoma/ttf/"
                     "/usr/local/share/fonts/FontCity2//"
                     "/usr/local/share/fonts/DynaFont-Premium30/win95//"
                     "/usr/local/share/fonts/DynaFont-SpecialPack1/win95//"
                     "/usr/local/share/fonts/TrueTypeWorld-ValueFont141//"
                     "/usr/local/share/fonts/FontAsia//"
                     "/usr/local/share/fonts/FontGarden/ttf//"  )
  ;; debugging flags   ('*' selects all)
  (debug "")
  ;; properties for all fonts of this font class
  (properties ("FONT_CLASS" "TrueType"))  )
```

```
;; -----------------------------------------------------------------
;; Type1 Font Class Default
;; -----------------------------------------------------------------
(define-default type1
  ;; font (pfa, pfb) directories
  (font-directories "TEXMF"  ; - a special name to search by ‘kpathsea’
                   "/usr/local/share/fonts/bakoma/pfb/")
  ;; AFM directories
  (afm-directories "TEXMF"   ; - a special name to search by ‘kpathsea’
                  "/usr/local/share/fonts/bakoma/afm/")
  ;; T1Lib logfile output level: "none", "error", "warning", "stat", "debug"
  (log-level "none"))

;; -----------------------------------------------------------------
;;   Syotrai Club Font Class Default
;; -----------------------------------------------------------------
(define-default  zeit
  ;; filename extensions ("vf" for "mincho.vf{1,2}")
  (filename-extensions  ".vf")
  ;; font directories
  (font-directories   "/usr/local/share/fonts/Watanabe//"
                      "/usr/local/share/fonts/WadaLab//")
  ;; properties for all fonts of this font class
  (properties  ("FONT_CLASS" "ZEIT")
               ("CHARSET_REGISTRY" "jisx0208.1983")
               ("CHARSET_ENCODING" "0") ))

;; -----------------------------------------------------------------
;;   JG Font Class Default
;; -----------------------------------------------------------------
(define-default  jg
  ;; filename extensions ("fn" for "zkyo0by.fn{0,1,2}")
  (filename-extensions ".fn")
  ;; font directories
  (font-directories  "/usr/local/share/fonts/JG-Fonts//")
  ;; properties for all fonts of this font class
  (properties  ("FONT_CLASS" "JG")
               ("CHARSET_REGISTRY" "jisx0208.1983")
               ("CHARSET_ENCODING" "0")))

;; -----------------------------------------------------------------
;;   TeX-related Font Class Default and TeX Font Mapper
;; -----------------------------------------------------------------
(define-default  TeX
  ;; TFM file directories
  (tfm-directories  "TEXMF"
                   "/usr/local/share/fonts/bakoma/tfm/")
  ;; possible extensions of TFM files
  (tfm-filename-extensions  ".tfm")
  ;; font name mapping rules
  (font-mapping
   ((ascii-jtex-kanji "%f.jtex")  "min*" "goth*" "tmin*" "tgoth*")
```

```
   ((type1 "%f.pfb" point-size-from-tfm (magnification-adjustment 1.0))
     *)
   ((pk "%f.%dpk") (gf "%f.%dgf")  *)
   ((tfm "%f.%dtfm")  *))
  ;; accuracy of device resolutions, used with 'resolution-corrections'
  (resolution-accuracy 0.02)
  ;; font resolution values for each device resolutions.
  (resolution-corrections
   (240  ;; sparcptr
    240  263  288  312  346  415  498  597)
   (300  ;; cx
    300  329  360  432  518  622  746  896 1075 1290  240 270)
   (400  ;; sparcptr
    400  438  480  576  691  829  995 1194 1433 1720  320 360)
   (600  ;; ljfour
    600  657  720  864 1037 1244 1493 1792 2150 2580  480 540))
  ;; default device resolution
  (dpi $TeX_DPI))

;; -------------------------------------------------------------------
;;   TeX GF Font Class Default
;; -------------------------------------------------------------------
(define-default  gf
  ;; font directories
  (font-directories  "TEXMF"
                     "/usr/local/TeX/gf//")
  ;; properties for all fonts of this font class
  (properties  ("FONT_CLASS" "GF")) )

;; -------------------------------------------------------------------
;;   TeX PK Font Class Default
;; -------------------------------------------------------------------
(define-default  pk
  ;; font directories
  (font-directories  "TEXMF"
                     "/usr/local/TeX/pk//")
  ;; properties for all fonts of this font class
  (properties  ("FONT_CLASS" "PK")) )

;; -------------------------------------------------------------------
;;   TeX TFM Font Class Default
;; -------------------------------------------------------------------
(define-default  tfm
  ;; font directories
  (font-directories  "TEXMF"
                     "/usr/local/TeX/tfm//")
  ;; glyph style: "fill" (all pixels black) or "empty" (all pixels white)
  (glyph-style  "fill")
  ;; properties for all fonts of this font class
  (properties  ("FONT_CLASS" "TFM")) )

;; -------------------------------------------------------------------
```

```
;;   ASCII-jTeX Kanji Font Class Default
;; -----------------------------------------------------------------
(define-default  ascii-jtex-kanji
  ;; TFM file directories
  (tfm-directories  "TEXMF")
  ;; possible extensions of TFM files
  (tfm-filename-extensions ".tfm")
  ;; Suffix for name mapping
  ;; (e.g., implicit font "min10.300pk" is mapped to "min10.jtex")
  (implicit-font-mapping-suffix ".jtex"))

;; -----------------------------------------------------------------
;;   Japanese Comic Font Class Default
;; -----------------------------------------------------------------
(define-default  japanese-comic
  ;; debugging flags   ('*' selects all)
  (debug "f")
  ;; properties for all fonts of this font class
  (properties  ("FONT_CLASS" "JAPANESE-COMIC")
               ("CHARSET_REGISTRY" "jisx0208.1983")
               ("CHARSET_ENCODING" "0")))


;; -----------------------------------------------------------------
;;  sample font definitions for Japanese TeX

(define-font jtex-min    (font-class pcf) (font-file "jiskan24.pcf"))
(define-font jtex-goth   (font-class pcf) (font-file "jiskan24.pcf"))
(define-font jtex-tmin   (font-class pcf) (font-file "jiskan24.pcf"))
(define-font jtex-tgoth  (font-class pcf) (font-file "jiskan24.pcf"))

;; Definitions for "min10" fonts. These fonts are used by
;; 'name mapping' feature of ascii-jtex-kanji driver.
;; (e.g., "min10.400pk" is mapped to "min10.jtex")
(define-macro min-common
  (font-class ascii-jtex-kanji) (kanji-font jtex-min)
  (kanji-font-magnification 0.85) (metric-adjustment-file "jiskan24.adj"))
(define-macro goth-common
  (font-class ascii-jtex-kanji) (kanji-font jtex-goth)
  (kanji-font-magnification 0.85) (metric-adjustment-file "jiskan24.adj"))
(define-macro tmin-common
  (font-class ascii-jtex-kanji) (kanji-font jtex-tmin)
  (kanji-font-magnification 0.85)
  (metric-adjustment-file "jiskan24v.adj"))
(define-macro tgoth-common
  (font-class ascii-jtex-kanji) (kanji-font jtex-tgoth)
  (kanji-font-magnification 0.85)
  (metric-adjustment-file "jiskan24v.adj"))
(define-font min5.jtex     (kanji-font-point-size  5)  min-common)
(define-font min6.jtex     (kanji-font-point-size  6)  min-common)
(define-font min7.jtex     (kanji-font-point-size  7)  min-common)
(define-font min8.jtex     (kanji-font-point-size  8)  min-common)
```

```
(define-font min9.jtex     (kanji-font-point-size  9)  min-common)
(define-font min10.jtex    (kanji-font-point-size 10)  min-common)
(define-font goth5.jtex    (kanji-font-point-size  5)  goth-common)
(define-font goth6.jtex    (kanji-font-point-size  6)  goth-common)
(define-font goth7.jtex    (kanji-font-point-size  7)  goth-common)
(define-font goth8.jtex    (kanji-font-point-size  8)  goth-common)
(define-font goth9.jtex    (kanji-font-point-size  9)  goth-common)
(define-font goth10.jtex   (kanji-font-point-size 10)  goth-common)
(define-font tmin5.jtex    (kanji-font-point-size  5)  tmin-common)
(define-font tmin6.jtex    (kanji-font-point-size  6)  tmin-common)
(define-font tmin7.jtex    (kanji-font-point-size  7)  tmin-common)
(define-font tmin8.jtex    (kanji-font-point-size  8)  tmin-common)
(define-font tmin9.jtex    (kanji-font-point-size  9)  tmin-common)
(define-font tmin10.jtex   (kanji-font-point-size 10)  tmin-common)
(define-font tgoth5.jtex   (kanji-font-point-size  5)  tgoth-common)
(define-font tgoth6.jtex   (kanji-font-point-size  6)  tgoth-common)
(define-font tgoth7.jtex   (kanji-font-point-size  7)  tgoth-common)
(define-font tgoth8.jtex   (kanji-font-point-size  8)  tgoth-common)
(define-font tgoth9.jtex   (kanji-font-point-size  9)  tgoth-common)
(define-font tgoth10.jtex  (kanji-font-point-size 10)  tgoth-common)

;; -----------------------------------------------------------------
;; EOF
```

6.28 Example vflibcap 2

This vflibcap file is an example for TeX DVI drivers. This vflibcap file provides a set of fonts of the form `NAME.DVI`pk and `NAME`.pk. For example, `cmr10.300pk` and `cmr10.pk`.

For Japanese Kanji character fonts `min5` ... `min10`, `goth5` ... `goth10`, `tmin5` .. `tmin10`, `tgoth5` ... `tgoth10`, X Window PCF format font `jiskan24.pcf` is used via `ascii-jtex-kanji` font driver.

Other fonts are solved in PK and GF format fonts. If a font is not available in these formats, a TFM font is used to produce a "black" box. (TFM files are metrics files and do not conatin glyph. But TFM driver in VFlib produces a "box" glyph as it ware font files.)

Parameters of device resolution and magnification factor for the function `VF_OpenFont1()` determines the font size and font metrics to be opened.

To use VFlib with this vflibcap file, I recommend to open font by `VF_OpenFont1(name.pk, dpi, dpi, -1, mag, mag)`, where `name.pk` is a font name (e.g., `cmr10.pk`), *dpi* is the device resolution in dpi (e.g., `300`), and *mag* is the magnification factor (e.g., `1.2` for magstep 1 fonts).

```
;; -----------------------------------------------------------------
;;   VFlib Default
;; -----------------------------------------------------------------
(define-default VFlib
  (extension-hints  ("pk" TeX))
  (implicit-font-classes)
  (uncompression-programs  (".Z" "zcat") (".gz" "gzip -cd"))
  (variable-values  (TeX_USE_KPATHSEA      "Yes")
                    (TeX_DPI               "300")  ;; or "600"
                    (TeX_KPATHSEA_MODE     "cx")   ;; or "ljfour"
                    (TeX_KPATHSEA_PROGRAM  "/usr/local/bin/xldvi"))
```

```
  (use-kpathsea           $TeX_USE_KPATHSEA)
  (kpathsea-mode          $TeX_KPATHSEA_MODE)
  (kpathsea-program-name  $TeX_KPATHSEA_PROGRAM)
  (code-conversion-files
   "iso8859-1_unicode.ccv" "iso8859-2_unicode.ccv" "iso8859-3_unicode.ccv"
   "iso8859-4_unicode.ccv" "iso8859-5_unicode.ccv" "iso8859-6_unicode.ccv"
   "iso8859-7_unicode.ccv" "iso8859-8_unicode.ccv" "iso8859-9_unicode.ccv"
   "jisx0201_unicode.ccv" "jisx0208_unicode.ccv" "jisx0212_unicode.ccv"
   "ksc5601_unicode.ccv"
   "gb12345_unicode.ccv" "gb2312_unicode.ccv"
   "big5_unicode.ccv" "cns11643_unicode.ccv"
   "iso8859-5_koi8-r.ccv" "koi8-r_iso8859-5.ccv" "koi8-r_unicode.ccv"))

;; -------------------------------------------------------------------
;;   TeX-related Font Class Default and TeX Font Mapper
;; -------------------------------------------------------------------
(define-default  TeX
  (tfm-directories  "TEXMF")
  (tfm-filename-extensions  ".tfm")
  (font-mapping
   ((ascii-jtex-kanji "%f.jtex")  "min*" "goth*" "tmin*" "tgoth*")
   ((pk "%f.%dpk") (gf "%f.%dgf")  *)
   ((tfm "%f.%dtfm") *))
  (resolution-accuracy 0.02)
  (resolution-corrections
   (240  ;; sparcptr
    240  263  288  312  346  415  498  597)
   (300  ;; cx
    300  329  360  432  518  622  746  896 1075 1290  240 270)
   (400  ;; sparcptr
    400  438  480  576  691  829  995 1194 1433 1720  320 360)
   (600  ;; ljfour
    600  657  720  864 1037 1244 1493 1792 2150 2580  480 540))
  (dpi $TeX_DPI))

;; -------------------------------------------------------------------
;;   GF Font Class Default
;; -------------------------------------------------------------------
(define-default gf
  (font-directories  "TEXMF"))

;; -------------------------------------------------------------------
;;   PK Font Class Default
;; -------------------------------------------------------------------
(define-default pk
  (font-directories  "TEXMF"))

;; -------------------------------------------------------------------
;;   VF Font Class Default
;; -------------------------------------------------------------------
(define-default vf
  (font-directories  "TEXMF")
```

```
  (font-mapping
   ((type1 "%f.pfb" point-size-from-tfm)  *) )
  (open-style "try")     ;; "none", "try", or "require"
  (glyph-style "fill"))  ;; "fill", or "empty"

;; -------------------------------------------------------------------
;;   TFM Font Class Default
;; -------------------------------------------------------------------
(define-default tfm
  (glyph-style  "fill"))

;; -------------------------------------------------------------------
;;   ASCII-JTeX Kanji fonts
;; -------------------------------------------------------------------
(define-default  ascii-jtex-kanji
  (tfm-directories  "TEXMF")
  (implicit-font-mapping-suffix ".jtex"))

;; -------------------------------------------------------------------
;; Type1 Font Class Default
;; -------------------------------------------------------------------
(define-default type1
  (font-directories  "TEXMF")
  (afm-directories   "TEXMF")
  (log-level "none")
  (dpi $TeX_DPI))

;; -------------------------------------------------------------------
;; TrueType Font Class Default
;; -------------------------------------------------------------------
(define-default truetype
  (font-directories  "TEXMF")
  (platform-id "microsoft")
  (dpi $TeX_DPI))

;; -------------------------------------------------------------------
;;   PCF Font Class Default
;; -------------------------------------------------------------------
(define-default  pcf
  (font-directories  "/usr/X11R6/lib/X11/fonts//"
                     "/usr/local/X11R6/lib/X11/fonts//"
                     "/usr/openwin/lib/X11/fonts//"
                     "/usr/X386/lib/X11/fonts//"
                     "/usr/XFree86/lib/X11/fonts//"
                     "/usr/X11/lib/X11/fonts//"
                     "/usr/local/lib/X11/fonts//"
                     "/usr/X11R5/lib/X11/fonts//"
                     "/usr/local/X11R5/lib/X11/fonts//"
                     "/usr/local/share/fonts/X11//")
  (compression-extensions ".gz" ".Z")
  (dpi $TeX_DPI))
```

```
;; -----------------------------------------------------------------
;;  Japanese Kanji fonts using standard X11 PCF fonts
(define-font jtex-min    (font-class pcf) (font-file "jiskan24.pcf"))
(define-font jtex-goth   (font-class pcf) (font-file "jiskan24.pcf"))
(define-font jtex-tmin   (font-class pcf) (font-file "jiskan24.pcf"))
(define-font jtex-tgoth  (font-class pcf) (font-file "jiskan24.pcf"))

(define-macro min-common
  (font-class ascii-jtex-kanji) (kanji-font jtex-min)
  (kanji-font-magnification 0.85) (metric-adjustment-file "jiskan24.adj"))
(define-macro goth-common
  (font-class ascii-jtex-kanji) (kanji-font jtex-goth)
  (kanji-font-magnification 0.85) (metric-adjustment-file "jiskan24.adj"))
(define-macro tmin-common
  (font-class ascii-jtex-kanji) (kanji-font jtex-tmin)
  (kanji-font-magnification 0.85) (metric-adjustment-file "jiskan24v.adj"))
(define-macro tgoth-common
  (font-class ascii-jtex-kanji) (kanji-font jtex-tgoth)
  (kanji-font-magnification 0.85) (metric-adjustment-file "jiskan24v.adj"))

(define-font min5.jtex     (kanji-font-point-size  5)  min-common)
(define-font min6.jtex     (kanji-font-point-size  6)  min-common)
(define-font min7.jtex     (kanji-font-point-size  7)  min-common)
(define-font min8.jtex     (kanji-font-point-size  8)  min-common)
(define-font min9.jtex     (kanji-font-point-size  9)  min-common)
(define-font min10.jtex    (kanji-font-point-size 10)  min-common)
(define-font goth5.jtex    (kanji-font-point-size  5)  goth-common)
(define-font goth6.jtex    (kanji-font-point-size  6)  goth-common)
(define-font goth7.jtex    (kanji-font-point-size  7)  goth-common)
(define-font goth8.jtex    (kanji-font-point-size  8)  goth-common)
(define-font goth9.jtex    (kanji-font-point-size  9)  goth-common)
(define-font goth10.jtex   (kanji-font-point-size 10)  goth-common)
(define-font tmin5.jtex    (kanji-font-point-size  5)  tmin-common)
(define-font tmin6.jtex    (kanji-font-point-size  6)  tmin-common)
(define-font tmin7.jtex    (kanji-font-point-size  7)  tmin-common)
(define-font tmin8.jtex    (kanji-font-point-size  8)  tmin-common)
(define-font tmin9.jtex    (kanji-font-point-size  9)  tmin-common)
(define-font tmin10.jtex   (kanji-font-point-size 10)  tmin-common)
(define-font tgoth5.jtex   (kanji-font-point-size  5)  tgoth-common)
(define-font tgoth6.jtex   (kanji-font-point-size  6)  tgoth-common)
(define-font tgoth7.jtex   (kanji-font-point-size  7)  tgoth-common)
(define-font tgoth8.jtex   (kanji-font-point-size  8)  tgoth-common)
(define-font tgoth9.jtex   (kanji-font-point-size  9)  tgoth-common)
(define-font tgoth10.jtex  (kanji-font-point-size 10)  tgoth-common)
;; -----------------------------------------------------------------
;; EOF
```

6.29 Example vflibcap 3

This is an example for TEX DVI drivers. This vflibcap desgnates VFlib to use PK files. For missing PK files, black "boxes" by TFM fonts are used as substitutes of glyphs of PK files

```
;; -------------------------------------------------------------------
;;   VFlib Default
;; -------------------------------------------------------------------
(define-default VFlib
  (extension-hints  ("pk" TeX) ("gf" TeX))
  (implicit-font-classes)
  (uncompression-programs  (".Z" "zcat") (".gz" "gzip -cd"))
  (variable-values  (TeX_USE_KPATHSEA      "Yes")
                    (TeX_DPI               "300")  ;; or "600"
                    (TeX_KPATHSEA_MODE     "cx")   ;; or "ljfour"
                    (TeX_KPATHSEA_PROGRAM  "/usr/local/bin/xldvi"))
  (use-kpathsea           $TeX_USE_KPATHSEA)
  (kpathsea-mode          $TeX_KPATHSEA_MODE)
  (kpathsea-program-name  $TeX_KPATHSEA_PROGRAM))

;; -------------------------------------------------------------------
;;   TeX-related Font Class Default and TeX Font Mapper
;; -------------------------------------------------------------------
(define-default  TeX
  (tfm-directories  "TEXMF"
                    "/usr/local/lib/jtex/fonts"
                    "/usr/local/lib/tex/fonts")
  (tfm-filename-extensions  ".tfm")
  (font-mapping
   ((pk "%f.%dpk") *)
   ((tfm "%f.%dtfm") *))
  (resolution-accuracy 0.02)
  (resolution-corrections
   (240  ;; sparcptr
    240  263  288  312  346  415  498  597)
   (300  ;; cx
    300  329  360  432  518  622  746  896 1075 1290  240 270)
   (400  ;; sparcptr
    400  438  480  576  691  829  995 1194 1433 1720  320 360)
   (600  ;; ljfour
    600  657  720  864 1037 1244 1493 1792 2150 2580  480 540))
  (dpi $TeX_DPI))

;; -------------------------------------------------------------------
;;   PK Font Class Default
;; -------------------------------------------------------------------
(define-default pk
  (font-directories "TEXMF"))

;; -------------------------------------------------------------------
;;   TFM Font Class Default
;; -------------------------------------------------------------------
```

```
(define-default tfm
  (glyph-style  "fill"))

;; -----------------------------------------------------------------
;EOF
```

7 Debugging a vflibcap

There is no utility programs that checks syntax of a vflibcap file. But VFlib checks syntax of vflibcap file when a font driver is initialized or a font is opened.

VFlib prints a message to inform a user if syntax is illegal, undefined capability is used (this may be a typographical error), essential capability is missing, an undefined macro is used, or forms of capability values are illegal.

The following Unix environment variables are used to print debugging messages.

`VFLIB_DEBUG_FONT_OPEN`
: — If this environment variable is defined, the processes of font opens are printed.

`VFLIB_DEBUG_FONT_SEARCH`
: — If this variable is defined, the processes of font opens are printed

`VFLIB_DEBUG_VFLIBCAP`
: — If this variable is defined, the process of reading of vflibcap file is printed.

`VFLIB_DEBUG_PARAMETERS`
: — If this variable is defined, VFlib prints how parameters (variables) in vflibcap file are substituted.

`VFLIB_DEBUG_CCV`
: — If this variable is defined, the process of reading CCV files is printed.

`VFLIB_DEBUG_CCV_MAPPING`
: — If this variable is defined, encoding conversions by CCV are printed.

8 Code conversion system

Code conversion system (CCV) is used to convert from a character set and an encoding to another. For example, a font of Unicode character set and Unicode encoding can be accessed as ISO 8859-2 character set of ISO encoding by encoding conversion. TrueType font class makes use of this feature to hide invisible internal font encoding scheme and provides desired external view to users.

Conversion rule is given by one of the following two methods

- Internal functions in VFlib (written in C) These functions are hardcoded and new conversions rules cannot be added without modifying source code.
- External files, called CCV files. A list of CCV files to be used is specified in `code-conversion-files` capability of `VFlib` default.

8.1 How CCV works

Each conversion rule has the following information.

- EXTERNAL charset name
- EXTERNAL encoding name
- INTERNAL charset name
- INTERNAL encoding name
- other info such as format and size of conversion table...

On invocation of VFlib, these information is read from each CCV files. (CCV files are not fully loaded at initialization of VFlib; VFlib just checks relation of conversion. Conversion tables, which can be large, are loaded on demand.) In addition, when VFlib is initialized, internal CCV functions are installed and these information is given for each conversion function.

"EXTERNAL" means external view (i.e., user side encoding) and "INTERNAL" means internal view (i.e., font encoding). Users can define arbitrary charset and encoding names, except that some font driver may predefined names for internal use. (TrueType font driver uses some predefined names, such as "unicode".)

CCV system has a conversion table searching mechanism. Table is searched by source charset/encoding names and destination charset/encoding names. If there is a CCV file listed in `code_conversion_files` capability of `VFlib` defaults entry in vflibcap that matches charset and encoding name, the CCV file is dynamically loaded and used for code conversion.

For example, a CCV file `iso8859-1_unicode.ccv` has the following charset/encoding names:

- EXTERNAL charset name: `ISO8859-1`
- EXTERNAL charset encoding: `ISO`
- INTERNAL charset name: `UNICODE`
- INTERNAL charset encoding: `UNICODE`

By this CCV file, a unicode font can be viewed as a ISO encoding of ISO 8859-1 charset. (It is very important to note that names are just symbols and not have any meaning; in the above example, conversion to ISO encoding is implemented by conversion table body in CCV file.)

In the current implementation, BDF, PCF, HBF, and TrueType font drivers use CCV system. In the following, how TrueType font driver uses CCV is described.

Each TrueType font has information about charset name and encoding name of the font. When a font entry is defined in vflibcap file and `encoding` and `character-set` capability is defined, say, E and C respectively. According to internal charset and encoding information of TrueType font, the driver searches a CCV table, when the VFlib font is opened, that matches the following conversion relation.

- EXTERNAL charset name: *E*
- EXTERNAL charset encoding: *C*
- INTERNAL charset name: possibly, `UNICODE` (from font file info)
- INTERNAL charset encoding: possibly, `UNICODE` (from font file info)

If not found, conversion is impossible. If found, a CCV file found is used. After a font is opened, CCV table is used for converting code points for VFlib operation such as `VF_GetBitmap1()`.

8.2 The internal (hardcoded) CCV functions

VFlib has several hardcoded CCV functions. Followings CCV functions are implemented.

- from ISO-2022 (`ISO2022`) to Shift JIS (`SJIS`)
- from Shift JIS (`SJIS`) to ISO 2022 (`ISO2022`)
- from EUC (`EUC`) to ISO 2022 (`ISO2022`)
- from Row-Cell (`Row-Cell`) to ISO 2022 (`ISO2022`)
- from ISO-2022 (`ISO2022`) to Row-Cell (`Row-Cell`)
- from ISO-2022 (`ISO2022`) to WanSung (`WanSung`)
- from Row-Cell (`Row-Cell`) to WanSung (`WanSung`)
- from ISO-2022 (`ISO2022`) to Sequential Numbering (`Sequential2-0` and `Sequential2-1`)

  By these encoding schemes, characters are numbered sequentially starting from 0 (`Sequential2-0`) or 1 (`Sequential2-1`). That is, `Sequential2-0` encoding is an encoding such that code of the first character is 0, code of the second is 1, ..., and code of the i-th character is (i-1). `Sequential2-1` encoding is an encoding such that code of the first character is 1, code of the second is 2, ..., and code of the i-th is (i). External code point must be encoded two-byte, i.e., 0x2121...7e7e. These values are converted to 0...8835 or 1...8836.

Encoding name `JIS` is defined as an alias of `ISO2022`. Encoding name `Ku-Ten` is defined as an alias of `Row-Cell`. Note that these aliases are defined only for hardcorded CCV functions.

All of these are implemented simple arithmetic and large conversion tables are not necessary in memory.

8.3 The syntax of CCV files

The syntax of CCV files is lisp-like notation, similar to vflibcap files. The CCV file defines its own directive set, explained below. A code conversion table is divided in several sub-tables to reduce the file size (and memory size when the file is loaded into memory). The sub-tables are called 'blocks'.

`(charset-external-name` *`from-cs-name`*`)`
`(charset-external-encoding` *`from-cs-enc`*`)`
`(charset-internal-name` *`to-cs-name`*`)`
`(charset-internal-encoding` *`to-cs-enc`*`)`

: — These four directives describes character set and encoding information of conversion.

`(table-type` *`type`*`)`

: *type* must be one of the following:

    `array`

```
              random-arrays
(c1-min c1min)
(c1-max c1max)
(c2-min c2min)
(c2-max c2max)
(block-size size)
(nblocks nblocks)
(block block code0 code1 ...)
```

Let *c* be a code point of a character to be converted by this CCV file.

It is converted as follows. Let *c1* be *c/size* and *c2* be *c* modulo *size*. The block number *b* that should be referred to is *c1-c1min*. The position *i* in the block *b* is *c2-c2min*. Thus, the value of *i*-th entry of a block numbered *b* is converted code point.

c1max, *c2max* and *nblocks* are used internally to determine the necessary memory area to load the table.

Theoretically, `array` is enough for the value for `table-type` directive. But in case that there are many blocks that do not have conversion entries. This is happen in the case of CNS11643 character set (a Hanji character set in Taiwan). To reduce the table size, some of blocks can be omitted by giving `random-array` for `table-type` directive.

In case of `array`, lookup for code conversion is implemented by indexing an entire array, which is very fast. In case of `random-array`, lookup for code conversion takes time to find a corresponding sub-table (block), since the table is not linear.

8.4 Example of a CCV file 1

The following example is a CCV file that virtually provides a ISO 8859-1 character set font using a Unicode font. That is, a Unicode font can be used as if it were a ISO 8859-1 font by the CCV file. This file is distributed with VFlib and installed by the name `iso8859-1_unicode.ccv`.

This file is a table indexed by code points of ISO 8859-1; contents of table entries are Unicode code points.

```
; Conversion table: ISO8859-1 ==> UNICODE
(charset-external-name ISO8859-1)
(charset-external-encoding ISO)
(charset-internal-name UNICODE)
(charset-internal-encoding UNICODE)
(table-type array)
; Code point C is converted to C' by the following formula:
;   C' = Table[(c1 - c1min)*M + (c2 - c2min)],
;   where c1 = C/B and c2 = C%B, and M = c2max - c2min + 1.
;   B is a block size given by the 'block-size:' parameter.
(c1-min 0x0)
(c1-max 0x0)
(c2-min 0x20)
(c2-max 0xff)
(block-size 256)
(nblocks 1)
; 0x0020 ... 0x00ff
(block 0
    0x0020 0x0021 0x0022 0x0023 0x0024 0x0025 0x0026 0x0027
    0x0028 0x0029 0x002a 0x002b 0x002c 0x002d 0x002e 0x002f
```

```
0x0030 0x0031 0x0032 0x0033 0x0034 0x0035 0x0036 0x0037
0x0038 0x0039 0x003a 0x003b 0x003c 0x003d 0x003e 0x003f
0x0040 0x0041 0x0042 0x0043 0x0044 0x0045 0x0046 0x0047
0x0048 0x0049 0x004a 0x004b 0x004c 0x004d 0x004e 0x004f
0x0050 0x0051 0x0052 0x0053 0x0054 0x0055 0x0056 0x0057
0x0058 0x0059 0x005a 0x005b 0x005c 0x005d 0x005e 0x005f
0x0060 0x0061 0x0062 0x0063 0x0064 0x0065 0x0066 0x0067
0x0068 0x0069 0x006a 0x006b 0x006c 0x006d 0x006e 0x006f
0x0070 0x0071 0x0072 0x0073 0x0074 0x0075 0x0076 0x0077
0x0078 0x0079 0x007a 0x007b 0x007c 0x007d 0x007e -1
-1     -1     -1     -1     -1     -1     -1     -1
-1     -1     -1     -1     -1     -1     -1     -1
-1     -1     -1     -1     -1     -1     -1     -1
-1     -1     -1     -1     -1     -1     -1     -1
0x00a0 0x00a1 0x00a2 0x00a3 0x00a4 0x00a5 0x00a6 0x00a7
0x00a8 0x00a9 0x00aa 0x00ab 0x00ac 0x00ad 0x00ae 0x00af
0x00b0 0x00b1 0x00b2 0x00b3 0x00b4 0x00b5 0x00b6 0x00b7
0x00b8 0x00b9 0x00ba 0x00bb 0x00bc 0x00bd 0x00be 0x00bf
0x00c0 0x00c1 0x00c2 0x00c3 0x00c4 0x00c5 0x00c6 0x00c7
0x00c8 0x00c9 0x00ca 0x00cb 0x00cc 0x00cd 0x00ce 0x00cf
0x00d0 0x00d1 0x00d2 0x00d3 0x00d4 0x00d5 0x00d6 0x00d7
0x00d8 0x00d9 0x00da 0x00db 0x00dc 0x00dd 0x00de 0x00df
0x00e0 0x00e1 0x00e2 0x00e3 0x00e4 0x00e5 0x00e6 0x00e7
0x00e8 0x00e9 0x00ea 0x00eb 0x00ec 0x00ed 0x00ee 0x00ef
0x00f0 0x00f1 0x00f2 0x00f3 0x00f4 0x00f5 0x00f6 0x00f7
0x00f8 0x00f9 0x00fa 0x00fb 0x00fc 0x00fd 0x00fe 0x00ff )
```

8.5 Example of a CCV file 2

The following example is a CCV file that virtually provides a CNS 11643 Plane 1 character set font using a Unicode font. This file is distributed with VFlib and installed by the name `cns11643-1_unicode.ccv`.

This file is an example of CCV files that have `random-arrays` for `table-type` directive.

```
; Conversion table: CNS11643-1 ==> UNICODE
(charset-external-name CNS11643-1)
(charset-external-encoding ISO2022)
(charset-internal-name UNICODE)
(charset-internal-encoding UNICODE)
(table-type random-arrays)
; Code point C is converted to C' by the following formula:
;   C' = Table[(c1 - c1min)*M + (c2 - c2min)],
;   where c1 = C/B and c2 = C%B, and M = c2max - c2min + 1.
;   B is a block size given by the 'block-size:' parameter.
(c1-min 0x121)
(c1-max 0xe67)
(c2-min 0x21)
(c2-max 0x7e)
(block-size 256)
(nblocks 218)
; 0x12121 ... 0x1217e
(block 0
```

```
    0x3000 0xff0c 0x3001 0x3002 0xff0e 0x30fb 0xff1b 0xff1a
    0xff1f 0xff01 0xfe30 0x2026 0x2025 0xfe50 0xfe51 0xfe52
    0x00b7 0xfe54 0xfe55 0xfe56 0xfe57 0xfe31 0x2014 0xfe32
    0x2013 -1     -1     -1     -1     0xff08 0xff09 0xfe35
    0xfe36 0xff5b 0xff5d 0xfe37 0xfe38 0x3014 0x3015 0xfe39
    0xfe3a 0x3010 0x3011 0xfe3b 0xfe3c 0x300a 0x300b 0xfe3d
    0xfe3e 0x3008 0x3009 0xfe3f 0xfe40 0x300c 0x300d 0xfe41
    0xfe42 0x300e 0x300f 0xfe43 0xfe44 0xfe59 0xfe5a 0xfe5b
    0xfe5c 0xfe5d 0xfe5e 0x2018 0x2019 0x201c 0x201d 0x301d
    0x301e 0x2032 0x2035 0xff03 0xff06 0xff0a 0x203b 0x00a7
    0x3003 0x25cb 0x25cf 0x25b3 0x25b2 0x25ce 0x2606 0x2605
    0x25c7 0x25c6 0x25a1 0x25a0 0x25bd 0x25bc )
; 0x12221 ... 0x1227e
(block 1
    0x32a3 0x2105 0x203e -1     0xff3f -1     0xfe49 0xfe4a
    0xfe4d 0xfe4e 0xfe4b 0xfe4c 0xfe5f 0xfe60 0xfe61 0xff0b
    0xff0d 0x00d7 0x00f7 0x00b1 0x221a 0xff1c 0xff1e 0xff1d
    0x2266 0x2267 0x2260 0x221e 0x2252 0x2261 0xfe62 0xfe63
    0xfe64 0xfe66 0xfe65 0x223c 0x2229 0x222a 0x22a5 0x2220
    0x221f 0x22bf 0x33d2 0x33d1 0x222b 0x222e 0x2235 0x2234
    0x2640 0x2642 0x2641 0x2609 0x2191 0x2193 0x2192 0x2190
    0x2196 0x2197 0x2199 0x2198 0x2016 0xff5c 0xff0f 0xff3c
    0x2215 0xfe68 0xff04 0xffe5 0x3012 0xffe0 0xffe1 0xff05
    0xff20 0x2103 0x2109 0xfe69 0xfe6a 0xfe6b 0x33d5 0x339c
    0x339d 0x339e 0x33ce 0x33a1 0x338e 0x338f 0x33c4 0x00b0
    0x5159 0x515b 0x515e 0x515d 0x5161 0x5163 )
; 0x12321 ... 0x1237e
(block 2
    0x55e7 0x74e9 0x7cce 0x2581 0x2582 0x2583 0x2584 0x2585
    0x2586 0x2587 0x2588 0x258f 0x258e 0x258d 0x258c 0x258b
    0x258a 0x2589 0x253c 0x2534 0x252c 0x2524 0x251c 0x2594
    0x2500 0x2502 0x2595 0x250c 0x2510 0x2514 0x2518 0x256d
    0x256e 0x2570 0x256f 0x2550 0x255e 0x256a 0x2561 0x25e2
    0x25e3 0x25e5 0x25e4 0x2571 0x2572 0x2573 -1     -1
    -1     -1     -1     -1     -1     -1     -1     -1
    -1     -1     -1     -1     -1     -1     -1     -1
    -1     -1     -1     -1     -1     -1     -1     -1
    -1     -1     -1     -1     -1     -1     -1     -1
    -1     -1     -1     -1     -1     -1     -1     -1
    -1     -1     -1     -1     -1     -1     )

... it's very long, snip, snip, snip ...

; 0xe6621 ... 0xe667e
(block 3397
    0x7bd0 0x7c2f 0x7c32 0x7c42 0x7c4e 0x7c68 0x7ca9 0x7ced
    0x7dd0 0x7e07 0x7dd3 0x7e64 0x7f40 -1     0x8041 0x8063
    0x80bb 0x6711 0x6725 0x8248 0x8310 0x8362 0x8312 0x8421
    0x841e 0x84e2 0x84de 0x84e1 0x8573 0x85d4 0x85f5 0x8637
    0x8645 0x8672 0x874a 0x87a9 0x87a5 0x87f5 0x8834 0x8850
    0x8887 0x8954 0x8984 0x8b03 0x8c52 0x8cd8 0x8d0c 0x8d18
    0x8db0 0x8ebc 0x8ed5 0x8faa 0x909c -1     0x915c 0x922b
```

```
    0x9221 0x9273 0x92f4 0x92f5 0x933f 0x9342 0x9386 0x93be
    0x93bc 0x93bd 0x93f1 0x93f2 0x93ef 0x9422 0x9423 0x9424
    0x9467 0x9466 0x9597 0x95ce 0x95e7 0x973b 0x974d 0x98e4
    0x9942 0x9b1d 0x9b98 -1     0x9d49 0x6449 0x5e71 0x5e85
    0x61d3 0x990e 0x8002 0x781e -1     -1     )
; 0xe6721 ... 0xe677e
(block 3398
    0x5528 0x5572 0x55ba 0x55f0 0x55ee 0x56b8 0x56b9 0x56c4
    0x8053 0x92b0 -1     -1     -1     -1     -1     -1
    -1     -1     -1     -1     -1     -1     -1     -1
    -1     -1     -1     -1     -1     -1     -1     -1
    -1     -1     -1     -1     -1     -1     -1     -1
    -1     -1     -1     -1     -1     -1     -1     -1
    -1     -1     -1     -1     -1     -1     -1     -1
    -1     -1     -1     -1     -1     -1     -1     -1
    -1     -1     -1     -1     -1     -1     -1     -1
    -1     -1     -1     -1     -1     -1     -1     -1
    -1     -1     -1     -1     -1     -1     -1     -1
    -1     -1     -1     -1     -1     -1     )
```

9 Utility programs

9.1 vflmkcaptex

`vflmkcaptex` is a utility program to generate vflibcap file for TEX DVI driver software automatically. With simple command line arugments, a vflibcap that uses PK, GF, Virtual Font, Type 1 fonts with complex TEX font mapping rules.

Usage: `vflmkcaptex` [*OPTIONS...*] [*CLASS...*]

Usage: `vflmkcaptex` [*SHORTCUT*] [*OPTIONS...*]

'*CLASS...*' is a list of font class names to support by vflibcap file to be generated. '*OPTIONS...*' is option list to customize default settings. '*SHORTCUT*' is a shortcut name to typical options and class name list.

Shortcut:

`minimum` This is the same as command line option `pk`. Use PK fonts only.

`simple` This is the same as command line option `-g pk tfm`. Use PK fonts. If PK font file is missing, it is generated on-the-fly. If font cannot be created, black square is displayed instead of character glyph (as long as corresponding TFM file exists).

`standard` This is the same as command line option `-t1 -g`. Use default class list `type1 vf pk tfm`. Missing PK font is created on-the-fly.

`simple-ja`
This is the same as command line option `-g pk tfm -jtex -jisx0212 -jpcf`. (Japanese support for `simple` shortcut.)

`standard-ja`
This is the same as command line option `-t1 -g -jtex -jisx0212 -jpcf`. (Japanese support for `standard` shortcut.)

Font class list:

`gf` Enables to use GF font files. For searching font files, kpathsea is used. That is, font files are searched under TEX '`texmf`' directory (typically, '`/usr/local/share/texmf`').

`pk` Enables to use PK font files. For searching font files, kpathsea is used.

`vf` Enables to use Virtual Font files. For searching font files, kpathsea is used.

`tfm` Enables to use TFM files. This option enables to display black square instead of glyph. The size of square obeys font metric of each character. This is useful when glyph file (e.g., PK, Type1) is missing.

`type1` Enables to use Type 1 fonts. (Currently, it supports Roman fonts. CJK fonts are not supported.) For searching font files, kpathsea is used.

By this option, font definitions for PostScript fonts used in TEX DVI files are generated by reading '`psfonts.map`' of `dvips`. Each PostScript font listed in '`psfonts.map`' is checked if it exists. (For PostScript fonts, this program automatically searchs Adobe Acrobat 3 and 4 font directories.) If a PostScript font in question is not found, `Ghostscript` font definition file '`Fontmap`' is checked to substitute the font by a font in `Ghostscript` font directory.

This feature is very useful for displaying and printing TEX DVI files with PostScript fonts. So, I recommned obtain Adobe Acrobat 3 and 4 for Type 1 PostScript fonts. (Linux version are freely available.)

When a font is requested to open, the font is tried to open by font classes by the order in the command line.

So, by a '`vflibcap`' generated by the following example, font in PK format is rearched first. If a font in PK format is not found, font in Type 1 format is searched next.

```
vflmkcaptex pk type1
```

Therefore, the order of font classes decides the priority of font file formats to search.

Options:

`--help` Print a list of command line options and exit.

`--version`
: Print version number of this program and exit.

`-p` *PROG* Application program name. This is used by kpathsea for font file search. Default is `xgdvi`, which is a DVI previewer in the TeX-Guy package.

`-n` *MODE* Device mode name for font file search, used by kpathsea. Default is `cx`

`-r` *DPI* Device resolution in DPI. Default is 300.
: If this option is not given, `vflmkcaptex` reads '`mode.mf`', which is a device mode definition file for METAFONT, and obtains revice resolution automatically.

`-g` Configure '`vflibcap`' to generate non-existing PK files on-the-fly.

`-pk` When '*CLASS...*' is not given, default font class set is assumed by default. For such case, generate a '`vflibcap`' to search PK font file before searching Type 1 font by this option.

`-t1` When '*CLASS...*' is not given, generate a '`vflibcap`' to search Type 1 font file before searching PK font file by this option.

Options for Japanese TeX support:

`-jtex` Generate font definitions for JIS X0208 character set used by Japanese TeX. By default, a '`vflibcap`' to be generated uses Japanese Kanji character in PCF format (in X11 font directory). See also '`-jpcf`', '`-jekanji`' and '`-jttf`' options.

`-jisx0212`
: Generate font definitions for JIS X0208 character set used by Japanese TeX. Note that generated font names are not standard. It is used for private use of the author.

`-jpcf` Switch to use PCF fonts for Japanese Kanji characters. (This is the default.)

`-jekanji` Switch to use eKanji fonts for Japanese Kanji characters. See Section 6.17 [eKanji font class], page 47 for detail about eKanji fonts.

`-jttf` Switch to use TrueType fonts for Japanese Kanji characters.

`-jpfd` *DIR*
: Add a PCF font directory. `vflmkcaptex` checks typical X11 PCF font directories and existing directories are added to PCF font directory list. This option should be used when you want to add optional (and not automatically detected) PCF font directory. This option can be used multiple times.

`-jefd` *DIR*
: Add a eKanji font directory. This option can be used multiple times.

`-jtfd` *DIR*
: Add a TrueType font directory. This option can be used multiple times.

`-jtdb` *FILE*

By this option, an external definition database file *FILE* is read for generating definitions of non-standard TeX Japanese fonts using Japanese TrueType font files. Each line in *FILE* is a pair of (1) font name used in TeX and (2) TrueType font file name. Following is an example:

```
dfailpaa dcail5.ttc
dfainpaa dcai5.ttc
dfaispaa dcais5.ttc
dfbrrsaa dfbrr7.ttc
dfbrrzaa dfbrrc.ttc
dfbrspaa dfbrs5.ttc
dfbrsvaa dfbrs9.ttc
dfbrszaa dfbrsc.ttc
```

See files in a directory '`ascii-jtex/`' for detail.

`vflmkcaptex` is a Unix Shell script. It uses following programs to generate a '`vflibcap`' file. Descriptions the followng programs are ommited since most of users never use them directly. For details, invoke each program with '`--help`' option to see how to use it.

`vflmkvfl` A generator for VFlib defaults. (See Section 6.9 [VFlib defaults], page 36.)

`vflmktex` A generator for TeX mapping class. (See Section 6.18 [TeX default and TeX font mapping font class], page 49.)

`vflmkpk` A generator for PK class. (See Section 6.19 [PK font class], page 52.)

`vflmkgf` A generator for GF class. (See Section 6.20 [GF font class], page 53.)

`vflmkvf` A generator for Virtual Font class. (See Section 6.22 [VF font class], page 54.)

`vflmktfm` A generator for TFM class. (See Section 6.21 [TFM font class], page 54.)

`vflmkt1` A generator for Type 1 class. (See Section 6.14 [Type1 font class], page 43.)

`vflmkekan`

A generator for eKanji class. (See Section 6.17 [eKanji font class], page 47.)

`vflmkajt` A generator for ASCII Japanese TeX class. (See Section 6.23 [ASCII Japanese TeX Kanji font class], page 56.)

9.2 vflpp

`vflpp` prettyprints (i.e., grinds) a vflibcap file. It eliminate all comment strings and unnecessary space and newline characters.

Usage: `vflpp` [*vflibcap-file*]

A program `vflpp` prettyprints a file *vflibcap* to standard output. If no argument is given, `vflpp` reads from standard input.

9.3 vflmkfdb

Usage: `vflmkfdb` *font-directory* [...]

A program `vflmkfdb` makes a font file hint database (FDB for short) in a font directories given in the command line argument.

It is used in a font file search module in VFlib. In case there are many font files in many directories, search a font file consumes much time to traverse directory hierarchy. FDB file contains pairs of file name and path name to the file in a single file. By reading FDB file, a font file can be found without traversing directories.

For each *font-directory*, a FDB file named '`VFlib.fdb`' is created in the directory.

9.4 vfldrvs

Usage: `vfldrvs`

A program `vfldrvs` prints a list of pre-installed font drivers in VFlib.

10 Sample programs

10.1 vflserver

`vflserver` is a font server that provides the functionality of VFlib via network.

`vflserver` can be invokes from command line or via network.

10.1.1 Using vflserver from command line

Usage: `vflserver` [`-v` *vflibcap*] [`-s` *shrink*] [*cmd-file ...*]

`vflserver` receives a command, executes it, and return a result. This is repeated until connection is closed or quit command is executed. `vflserver` reads a sequence of command from standard input if *cmd-file* option is not given.

Options:

`-v` *vflibcap*
: A file name of vflibcap to be used. If this option is not given, default vflibcap file is used. (Possibly, default vflibcap is '`/usr/local/share/VFlib/3.6.14/vflibcap`'.)

`-s` *shrink*
: `vflserver` has a feature to print obtained bitmaps in ASCII-art style for debugging purpose. When this feature is enabled, bitmaps are shrinked by this factor. This is effective when obtained bitmaps are huge.

cmd-file ...
: A sequence of commands can be read from files. Commands in files *cmd-file ...* are executed in given order. After executing all files, `vflserver` reads a sequence of commands from standard input. Thus, quit command may be explicitly given in *cmd-file*. This option is effective in the process of font driver development to do the same commands many times.

10.1.2 Using vflserver via network

Before using `vflserver` via network, it must be installed to be invoked by `inetd`. You must be a root to do the following procedures.

First, edit '`/etc/services`':

- Network service name: vflserver
- Well known port: 4681
- Protocol: tcp

Add the following line to '`/etc/inetd.conf`'.

```
vflserver stream tcp nowait nobody /usr/local/bin/vflserver vflserver
```

If you need to explicitly specify a vflibcap file to be used, you must give '`-v`' option as follows:

```
vflserver stream tcp nowait nobady /usr/local/bin/vflserver vflserver -v /foo/vflibcap
```

To force inetd to re-read '`inetd.conf`', send a HUP signal to inetd.

We finished installing vflserver to use via network. Now, use `telnet` to check if `vflserver` is correctly installed to network service. The following an example interaction.

```
% telnet localhost vflserver
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
```

```
; This is a font server VFLSERVER Version 2.0 Fri Mar 13 11:58:42 JST 1998
  ...

; Type ‘HELP’ for description of the protocol.

(100 "vflserver ready.")

open1 timR14.pcf
(100 0 "timR14.pcf")
debug bitmap on
(100 "Ascii-art bitmap on.")
bitmap1 0 0x67
(100 8 13 0 9 9 0
 "3eccc4c4cc78407c7f83c1e27c"
"
 89012345678901
 +------------+
9|            |9
0| ..@@@@@.   |0
1| @@..@@..   |1
2| @@...@..   |2
3| @@...@..   |3
4| @@..@@..   |4
5| .@@@@...   |5
6| .@......   |6
7| .@@@@@..   |7
8| .@@@@@@@   |8
9| +.....@@ o |9
0| @@.....@   |0
1| @@@...@.   |1
2| .@@@@@..   |2
3|            |3
 +------------+
 89012345678901
")
quit
(100 "Happy Hacking")
Connection closed by foreign host.
```

10.1.3 The protocol of vflserver

10.1.3.1 Introduction

The VFLSERVER Protocol is a communication protocol between a server which offers font service and a client which uses fonts.

The character set assumed by this protocol is ASCII character set. A line is a sequence of character terminated by a newline character and communication between a server and a client is line-oriented.

10.1.3.2 Reply Format of a Server

Each request to a server by a client takes a form of a line. The following are examples of client's requests.

```
OPEN1 timR24.pcf
DEBUG BITMAP ON
BITMAP1 1 33
```

A reply by a server to a client is an S-expression, (lisp-like notation). The following are examples of server's response.

```
(100 0 "timR14.pcf")
(100 "Ascii-art bitmap on.")
(100 8 13 0 9 9 0 "3eccc4c4cc78407c7f83c1e27c")
```

The first number of the response of each reply by a server are formed by decimal digits and these three digits indicates the status of an execution of client's request. Thus, this three digits is a status code.

The first digit is one of '1', '2', ..., '5'. If this digit is '1', it there is no error at all. If it is '5', there are some errors to achieve a request. According to the degree of fatalness, the digit is decided; It is '1' if no error is detected and is '5' if some fatal errors are detected and it is impossible to continue to execute a server. If it is not '5', a client can receive some result.

10.1.3.3 The Protocol

The following defines commands and their arguments by a client, and corresponding responses by a server. Command name is case-insensitive, but arguments are case-sensitive. In the description of command format, arguments enclosed by [ ] can be omitted, while arguments that are not enclosed by [ ] are essential arguments and cannot be omitted.

`OPEN1` *font_name* [*point_size* [*mag_x mag_y* [*dpi_x dpi_y*]]]

Open a font in mode 1 (high resolution device oriented mode). This corresponds to `VF_OpenFont1()` function. If it succeeds opening the font, a font identifier is returned. After a font is opened, any request for a font is specified by font identifier (not font name).

Response:

When the command is successful: `(` *`status fontid message`* `)`

fontid is a font id represented by non-negative integer in decimal.

When the command failed: `(` *`status message`* `)`

status indicates that an error occurred.

`OPEN2` *font_name* [*pixel_size* [*mag_x mag_y*]]

Open a font in mode 2 (low resolution device oriented mode). This corresponds to `VF_OpenFont2()` function. If it succeeds opening the font, a font identifier is returned. After a font is opened, any request for a font is specified by font identifier (not font name).

Response: Response is the same as one for `OPEN1` command.

`CLOSE` *font_id*

Closed a font.

Response: `(` *`status message`* `)`

`BITMAP1` *font_id code_point* [*mag_x mag_y*]

Obtain a bitmap. *font_id* is a font id. This command corresponds to `VF_GetBitmap1()` function of VFlib.

Response:

When the command is successful: `( status width height offx offy mvx mvy bitmap )`

width and *height* is a size of bitmap in pixels. *bitmap* is encoded as a sequence of hexadecimal number. Eight pixels are encoded to two hexadecimal number and the weight of the i-th ($0 <= i < 8$) pixel from the leftmost pixel is `0x80 >> i`. *bitmap* is a sequence of encoded of rasters; the first raster begins from the upper left corner to upper right corner. Then, it is followed by next raster (one pixel down from the first raster). One raster is $(width + 7)/8$ bytes, and *bitmap* contains an encoded bitmap of $((width + 7)/8) * height$ bytes. Thus, the length of *bitmap* is $2 * ((width + 7)/8) * height$.

When the command failed: `( status message )`

status indicates that an error occurred.

`BITMAP2` *font_id code_point* [*pixel_size* [*mag_x mag_y*]]

Obtain a bitmap. This command corresponds to `VF_GetBitmap2()` function. Response is the same as `BITMAP1` command.

`METRIC1` *font_id code_point* [*point_size* [*mag_x mag_y*]]

Obtain a metric information of a font. This command corresponds to `VF_GetMetric1()` function.

Response:

When the command is successful: `( status width height offx offy mvx mvy )`

Each element of the response is the same as return values for `BITMAP1` command except for their units are points.

When the command failed: `( status message )`

status indicates that an error occurred.

`METRIC2` *font_id code_point* [*pixel_size* [*mag_x mag_y*]]

Obtain a metric information of a font. This command corresponds to `VF_GetMetric2()` function.

Response: Same as METRIC1 command except for units are points.

`FONTBBX1` *font_id* [*mag_x mag_y*]

Obtain font bounding information of a given font *font_id* opened in mode 1. The argument *mag_x mag_y* are magnification factor to be scaled. This command corresponds to `VF_GetFontBoundingBox1()` function.

Response:

When the command is successful: `( status width height xoff yoff )`

width and *height* are width and height of bounding box, respectively. *xoff* and *yoff* are the largest horizontal and vertical displacement of lower left corner of bounding box from reference points. Note that these values does not guarantee the minimality; they only guarantee that all characters can be contained in a box described by them. Units of return values are point.

When the command failed: `( status message )`

status indicates that an error occurred.

`FONTBBX2` *font_id* [*mag_x mag_y*]

Arguments and return values are the same except *font_id* must be in mode 2 and units of return values are pixel.

`PROPERTY` *font_id property_name*
: Obtain a property named *property_name* of a font *font_id*.

  Response:

  When the command is successful: `(` ***`status value`*** `)`
  When the command failed: `(` ***`status message`*** `)`
  status indicates that an error occurred.

`MINIMIZE-BBX` [*flag*]
: Select a mode whether a bitmap sent from a server should be minimized or not. If *flag* is `ON`, bounding boxes of bitmaps sent from a server is minimized not to contain white pixels as possible. If *flag* is `OFF`, bitmaps sent from a server is not guaranteed to be minimized bounding boxes. If *flag* is not given, current mode is returned. Initial mode is `OFF`.

  Response: The current mode is returned even if the operation fails or succeeds.

  When the command is successful: `(` ***`status-code mode`*** `)`
  Mode is one of `ON` or `OFF`.

  When the command failed: `(` ***`status-code mode`*** `)`
  Mode is one of `ON` or `OFF`.

`QUIT` Finish interaction between a server and a client. This operation always succeeds.

  Response: `(` ***`status message`*** `)`

10.2 vfltest

`vfltest` displays glyph of a given font and characters by ASCII-art form on a character terminal. It does not requires X Window System, but the font of the terminal must be a fixed-width font, since bitmaps are printed by ASCII-art form.

Usage: `vfltest` [*OPTIONS...*] *FONT_NAME CHAR_LIST*

Options:

`-mode1` A font is opened in mode 1 (high resolution device oriented mode).

`-mode2` A font is opened in mode 2 (low resolution device oriented mode).

`-ol` Bitmaps are obtained by `VF_GetOutline()` and then `VF_OutlineToBitmap()`. This option is effective only when a font is opened in mode 1.

`-v` *vflibcap*
: A file name of vflibcap to be used. If this option is not given, default vflibcap file is used. (Possibly, default vflibcap is '`/usr/local/share/VFlib/3.6.14/vflibcap`'.)

`-p` *point* or `-p` *pixel*
: Specify point or pixel size of characters. If this option is not given, size of characters are original size of a font.

`-d` *dpi* Give a device resolution in DPI. This option is effective only when a font is opened in mode 1.

`-m` *mag* Specify vertical and horizontal magnification factor. If this option is not given, magnification factor is 1.0.

`-mx` *mag_h*
: Specify horizontal magnification factor. If this option is not given, horizontal magnification factor is 1.0.

`-my` *mag_v*
: Specify vertical magnification factor. If this option is not given, vertical magnification factor is 1.0.

`--help` Print command line arguments and key operations on a window.

A list of character is a sequence of the following forms.

code A character is given by character code. Decimal (e.g., '`34`') and Hexa-decimal (e.g., '`0x67`') numbers can be used.

from `-` *to* This form specifies characters by a range of character code, from *from* to *to* (e.g., '`0x20 - 0x7e`'). Space characters are necessary before and after minus sign (`-`).

`=`*charlist* This form specifies characters by a list of 1-byte characters, e.g., '`=abcdefg`'.

10.3 vflx11

`vflx11` displays glyph of a given font in a window. It requires X11R5 or X11R6.

Usage: `vflx11` [*OPTIONS...*] *FONT_NAME*

Options:

`-mode1` A font is opened in mode 1 (high resolution device oriented mode).

`-ol` Bitmaps are obtained by `VF_GetOutline()` and then `VF_OutlineToBitmap()`. This is effective when a font is opened in mode 1.

`-mode2` A font is opened in mode 2 (low resolution device oriented mode).

`-v` *vflibcap*
: A file name of vflibcap to be used. If this option is not given, default vflibcap file is used. (Possibly, default vflibcap is '`/usr/local/share/VFlib/3.6.14/vflibcap`'.)

`-p` *point* or `-p` *pixel*
: Specify point or pixel size of characters. If this option is not given, size of characters are original size of a font.

`-m` *mag* Specify vertical and horizontal magnification factor. If this option is not given, magnification factor is 1.0.

`-mx` *mag_h*
: Specify horizontal magnification factor. If this option is not given, horizontal magnification factor is 1.0.

`-my` *mag_v*
: Specify vertical magnification factor. If this option is not given, vertical magnification factor is 1.0.

`--help` Print command line arguments and key operations on a window.

Following operations are defined on a `vflx11` window.

⟨q⟩ Finish `vflx11`

⟨b⟩ Go to previous page.

⟨SPC⟩ Go to next page.

⟨[⟩ Go to previous 4 page.

⟨]⟩ Go to next 4 page.

⟨{⟩ Go to previous 16 page.

⟨}⟩ Go to next 16 page.

⟨+⟩ Enlarge the window.

⟨-⟩ Shrink the window.

⟨<⟩ Go to the first page.

⟨>⟩ Go to the last page.

⟨m⟩ Mark the current page.

⟨g⟩ Goto the marked page.

10.4 vfldisol

`vfldisol` displays 'disassembled lists' of vector data of a given font and code points.

Usage: `vfldisol` [*OPTIONS...*] *FONT_NAME CODE ...*

Options:

`-v` *VFLIBCAP*
: A file name of vflibcap to be used. If this option is not given, default vflibcap file is used. (Possibly, default vflibcap is '`/usr/local/share/VFlib/3.6.14/vflibcap`'.)

`-d` *DPI* Resolution of device in dpi. If this option is not given, default resolution of a font is used.

`-p` *POINT* Point size of characters. If this option is not given, default point size of a font is used.

`-x` Print hexadecimal dump of outline data instead of disassembled list.

10.5 ctext2pgm

`ctext2pgm` creates an image file in PGM or PBM format from a multilingual text file encoded by compound text format. It also supports various text encodings such as Chinese, Japanese, Korean EUCs and Shift-JIS. PGM and PBM formats are portable formats, image files can be easily converted to another image format such as GIF, TIFF.

It supports various character sets and left-to-right and right-to-left directionalities.

ISO 8859-1,2,3,4,5,7,8,9
: — Latin character sets, including Hebrew which is written from right to left.

Mule Arabic
: — An Arabic script, written from right to left. This character set is used by the multilingual editor Mule.

JIS X 0201, JIS X 0208, JIS X 0212
: — Japanese character sets.

GB 2312 — A Chinese character set.

CNS 11641-1, CNS 11641-2
: — Chinese character sets.

KSC 5601 — A Hangle character set.

10.5.1 Running ctext2pgm

Usage: `ctext2pgm` [*OPTIONS...*] [*file*]
— `ctext2pgm` reads *file* (if not given, reads standard input) and prints image file to standard output.

Example:

```
% ctext2pgm -pgm -ctext -16 -times DOC-10.txt  > IMAGE.pgm
```

(Never forget to redirect the output.)

Options for VFlib:

`-v` *`f`* — a vflibcap file to be used by `ctext2pgm`. Default value is '`vflibcap-ctext2pgm`'.

Options for input encoding and script:

`-ctext` — Assume that an encoding of input file. (This is the default input encoding.) Default writing directionality is set to left-to-right.

By this encoding, multiple character set can be used in an input text by escape sequences. Mixture of scripts of left-to-right and right-to-left directionalities is also supported.

`-iso-8859-1` or `-latin-1`
— Assume that input file is encoded by iso-8859-1. Escape sequence is not allowed in input file. Default writing directionality is set to left-to-right.

`-iso-8859-2` or `-latin-2`
— Assume that input file is encoded by iso-8859-2. Escape sequence is not allowed in input file. Default writing directionality is set to left-to-right.

`-iso-8859-3` or `-latin-3`
— Assume that input file is encoded by iso-8859-3. Escape sequence is not allowed in input file. Default writing directionality is set to left-to-right.

`-iso-8859-4` or `-latin-4`
— Assume that input file is encoded by iso-8859-4. Escape sequence is not allowed in input file. Default writing directionality is set to left-to-right.

`-iso-8859-5`, `-cyrillic` or `-russian`
— Assume that input file is encoded by iso-8859-5. Escape sequence is not allowed in input file. Default writing directionality is set to left-to-right.

`-iso-8859-7` or `-greek`
— Assume that input file is encoded by iso-8859-7. Escape sequence is not allowed in input file. Default writing directionality is set to left-to-right.

`-iso-8859-8` or `-hebrew`
— Assume that input file is encoded by iso-8859-7. Escape sequence is not allowed in input file. Default writing directionality is set to right-to-left.

`-iso-8859-9` or `-latin-5`
— Assume that input file is encoded by iso-8859-9. Escape sequence is not allowed in input file. Default writing directionality is set to left-to-right.

`-euc-jp` or `-euc-jp1`
— Assume that input file is encoded by Japanese EUC. Default writing directionality is set to left-to-right.

JIS X 0201 Roman character set is used for code set 0, JIS X 0208 is used for code set 1, JIS X 0201 Kana is used for code set 2, and JIS X 0212 is used for code set 3.

`-euc-jp2` — Same as `-euc-jp1` except ASCII character set is used for code set 0.

`-euc-kr` — Assume that input file is encoded by Korean EUC. Default writing directionality is set to left-to-right.

ASCII character set is used for code set 0 and KSC 5601 is used for code set 1.

`-euc-cn` or `-euc-gb`
— Assume that input file is encoded by Chinese EUC by simplified Hanzi. Default writing directionality is set to left-to-right.

ASCII character set is used for code set 0, and GB 2312 is used for code set 1.

`-euc-cns` — Assume that input file is encoded by Chinese EUC by traditional Hanzi. Default writing directionality is set to left-to-right.

ASCII character set is used for code set 0, CNS 11643-1 is used for code set 1, and CNS 11643-2 is used for code set 3.

`-sjis` — Assume that input file is encoded by Shift-JIS. Escape sequence is not allowed in input file. Default writing directionality is set to left-to-right.

ASCII character set is used for code set 0, JIS X 0208 is used for code set 1, and

Options for directionality:

`-l2r` — Select left-to-right directionality for typesetting.

`-r2l` — Select right-to-left directionality for typesetting.

Options for font selection:

`-fixed`, `-times`, `-helv` or `-cour`
— Select a font family: Fixed, Times, Helvetia, or Courier, respectively. Default font family is Times.

`-bold` or `-italic`
— Select a font face: bold or italic (or oblique), respectively. Default face is normal.

`-14`, `-16`, `-18` or `-24`
— Select a font set of 14-, 16-, 18-, or 24-dot, respectively. Default font size if 16.

`-scale` *n* — Select a scalable font set and scales the font to *n* dot.

`-m` *m* — Specify vertical and horizontal magnification factors. Default value is 1.

`-mx` *m* — Specify horizontal magnification factor.

`-my` *m* — Specify vertical magnification factor.

`-font-list`
— Print all installed character sets and font names. Then exit the program.

Options for typesetting:

`-b` *s* — Specify factor of baseline skip. Baseline of a text is moved this value times dot-size of a selected font set. Default value is 1.2.

`-center-line`
— Each line is centered. Output image is vertically and horizontally centered.

`-flush-left`
— Each line is flushed left. This is default mode if writing directionality is left-to-right. Output image is flushed left.

`-left-line`
— Each line is flushed left, but image is not flushed left.

`-flush-right`
: — Each line is flushed to right. This is the default mode if writing directionality is right-to-left. (Note that options '`-flush-right`' and '`-r2l`' are different — consider an English text including Arabic words in the same line.) Output image is flushed right.

`-right-line`
: — Each line is flushed right, but image is not flushed right.

It is important to notice that the difference of '`-flush-left`' and '`-left-line`' options (and '`-flush-right`' and '`-right-line`' options). By '`-flush-left`' option, input text is typeset to flush each line left and typeset result is placed in the left of an output image. By '`-left-line`' option, input text is typeset to flush each line left and does not specify how to put the typeset result in an output image. The difference appears when the horizontal size of output image is explicitly given by '`-pw`' option.

Options for output:

`-pgm` or `-pgm-raw`
: — Select binary PGM format for image output.

`-pgm-ascii`
: — Select ascii PGM format for image output. This is the default output mode.

`-pbm` or `-pbm-ascii`
: — Select ascii PBM format for output an image.

`-ascii-art` or `-ascii-art-v`
: — An image is printed as an ASCII art. (Vertical mode) Baseline is vertical; thus this mode is similar to the `banner` command on Unix.

`-ascii-art-h`
: — An image is printed as an ASCII art. (Horizontal mode) Baseline is horizontal.

`-eps` — Select EPS format for image output. By default, 16-dot font is printed by 12-point in EPS file. To change the point size, use the `-eps-ptsize` option described below.

`-eps-ptsize` *pt*
: — Select point size of characters for EPS output. If this option is given, point size of each character is scaled to *pt* point regardless dot size of fonts.

`-none` — An image is not shipped out.

`-r` — Reverse the black and white of output image. (This option does not have effect when EPS is selected for image output format.)

`-s` *n* — Shrink factor for anti-aliased output. *n* by *n* pixels are shrinked together and forms one pixel in an output image. This option has effect when output format is PGM and EPS. Default value is 1.

`-pw` *w* — Specify width of output image (in pixels). If this option is not given, the width of output image is the smallest width to contain the glyph of all characters.

`-ph` — Specify height of output image (in pixels). If this option is not given, the height of output image is the smallest height to contain the glyph of all characters.

`-g` — Specify horizontal and vertical margins of output image (in pixels). Default margin is zero pixel.

`-gx` — Specify horizontal margin of output image (in pixels). Default margin is zero pixel.

`-gy` — Specify vertical margin of output image (in pixels). Default margin is zero pixel.

`-center-image`
: — An image of typeset text is horizontaly and vertically centered.

`-h-center-image`
: — An image of typeset text is horizontaly centered.

`-v-center-image`
: — An image of typeset text is vertically centered.

`-left-image`
: — An image of typeset text is flushed left.

`-right-image`
: — An image of typeset text is flushed right.

`-top-image`
: — An image of typeset text is flushed top.

`-bottom-image`
: — An image of typeset text is flushed bottom.

10.5.2 Making input files for ctext2pgm

Any text editor can be used to prepare input files for `ctext2pgm`. Input files are plain texts. If you want to create an image containing multiple character sets, save the files by *compound text* encoding. If you want to make images of Arabic text, use the `Mule` editor. (`Mule` is an extension of `GNU Emacs` for multilingual text processing.) For making images of Arabic script, `ctext2pgm` only supports a text created by Mule, ISO-8859-6 is not supported.

Unlike TEX and HTML, newlines of input files are *not* ignored and a newline character in input text breaks line. Thus, input text is typeset like 'verbatim' environment of LaTEX or '<PRE> ... </PRE>' tag of HTML.

10.5.3 Commands in input text

Several commands can be embedded in text files such as font switch. Command sequence starts by a backslash (\) followed by one character which represents command name. If you want to display a backslash character itself, use double backslashes \\.

Following commands are defined:

`\f`, `\t`, `\h`, `\c`
: — Change of font families. Current font family is changed to fixed, times, Helvetia, courier, respectively.

`\d` — Current font family is changed to the default font family. The default font family can be specified by a command line option.

`\N`, `\B`, `\I` — Change of font faces. Current font face is changed to normal, bold, italic, respectively.

`\D` — Current font face is changed to the default font face. The default font face can be specified by a command line option.

`\.` — Same as `\d` followed by `\D`.

`\(` — Black and white of glyph of following characters are reversed. This command is recommended *only* for fixed-width fonts. (For the reason of current implementation, resulting bitmap is ugly for proportional fonts and you cannot read the text in an image.)

Nesting of `\(` has no effect.

`\)` — End of reversing black and white.

`\\` — Print backslash itself.

10.5.4 Trouble shooting

In case you failed to obtain a desired image output, the following command line options for debugging may be useful. (Debugging messages are printed to standard output. The '`-none`' option is useful to suppress printing binary image to your terminal. Otherwise, debugging message and image file are printed together on your terminal!)

`-ds` — Print the state transition of the parser for compound text.

`-dr2l` — Print the state transition of bi-directionality handling.

`-df` — Print font name to be opened.

`-dbc` — Print each character glyph in ascii-art form.

`-dbl` — Print each line image by in ascii-art form.

`-dbp` — Print entire page image in ascii-art form.

`-dall` — Selects all debugging options above.

11 Difference between VFlib version 3.6 and 2

VFlib version 3.6 and version 2 are quite different and you should forget about VFlib version 2.

VFlib 2 was designed only for Japanese Kanji fonts
: VFlib 3.6 can handle fonts for multilingual text printing.

Font metric is introduced in VFlib 3.6
: VFlib 2 does not concepts on font metrics since it assumes all characters are the same metrics. Font metrics is introduced in VFlib 3.6 and proportional fonts can be used.

Syntax of vflibcap file
: Syntax of vflibcap file is quite different. VFlib 2 adopted termcap-like notation, but now VFlib 3.6 adopts lisp-like notation.

Arguments and return values of function are changed
: In VFlib 2, bitmaps of characters are written in a frame buffer which is given by argument. But in VFlib 3.6, a bitmap object is returned.

Acknowledgments

Since I released VFlib version 1, so many people helped me to improve VFlib. I am grateful for all of them. Special gratitude is due to Satoru Tomura, Ken'ichi Handa, Werner Lemberg, and Ichiro Matsuda.

Concept index

Data type index

S

V

Function index

Program index

Short Contents

Table of Contents