Abstract

This is the preliminary documentation for the pilot release of **GeX**. **GeX** is the most rapidly evolving part of VTEX; more detailed documentation and more **GeX** power are forthcoming.

Chapter 1

Why GeX?

Starting with version 6.2, VTeX's PDF backend includes an integrated PostScript processor. This allows easy one-pass handling of

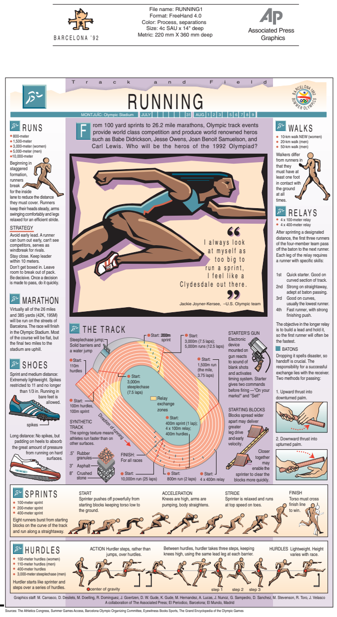
- Encapsulated PostScript files (`.eps`).
- Direct support for PSTricks and PSFrag.
- Other kinds of inline PostScript code, including feed-back of information the TeX processor.

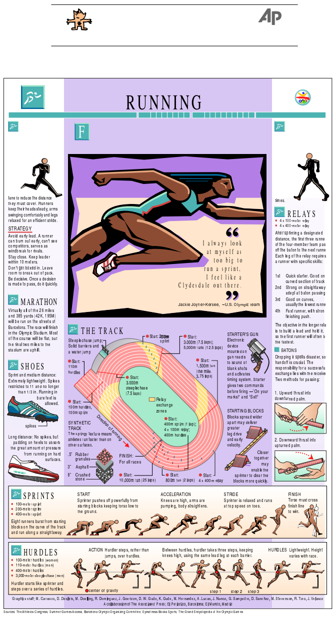
GeX shortcuts the lengthy model of

$$\begin{tikzcd}
 & & \texttt{.eps} \arrow[d, Rightarrow] & \\
\texttt{.tex} \arrow[r, Rightarrow] & \texttt{.dvi} \arrow[r, Rightarrow] & \texttt{.ps} \arrow[r, Rightarrow] & \texttt{.pdf} \\
 & & \text{inline}\texttt{.ps} \arrow[u, Rightarrow] &
\end{tikzcd}$$

While `.eps` inclusion has been previously supported in VTeX via GhostScript calls, **GeX** offers much better performance and output quality. Inline PostScript inclusion, on the other hand, is a totally new feature: the FastDraw support in the previous versions offered only a very a limited subset of PostScript.

The advantage of **GeX** over GhostScript should be apparent from the following picture





Here we include the same `.eps` file with both **GeX** and GhostScript (a new `\special{GS+}`/`\special{GS-}` was added to VTeX just for the sake of this comparison). Besides being cleaner, the picture on the left (**GeX** output) takes less space and takes less time to compile.

This is the TeX code used to place the pictures:

```
\hbox to \textwidth{
  ||\includegraphics[width=5cm]{olympics.eps} \hss
    \special{GS+}%
    \includegraphics[width=5cm]{olympics.eps}%
    \special{GS-}%
  ||%
  }
```

The `.eps` inclusion is likely to be the main *initial* application of **GeX**. However, in our view it is the inline PostScript which would lead to new and interesting applications.

1.1 Why call it GeX?

The **GeX** name [pronounced `g-e-k-s`] stands for Graphics EXtensions.

While the current extensions are generally compatible with the PostScript language, **GeX** is intended to be a TeX-resident extension, not an Acrobat clone. Even in the current implementation there are facilities for communication between TeX and PostScript; these facilities are likely to be further developed. While it is our intention to stay PostScript-compatible to a degree needed for

`.eps` and inline PostScript support we envision further enhancing **GeX** with features that are decisively non-PostScript.

Chapter 2

Using GeX

To enable **GeX**, use the
`-ox`
switch on the VTEX command line. **GeX** initialization will take place only when **GeX** is explicitly called for the first time. Notice that **GeX** works only in the **PDF** backend mode; in other modes the switch will be ignored.

Note: The `-ox2` switch will initialize **GeX** in Level II mode. Not all Level II operators are currently supported; so the use of this switch is somewhat experimental. See Language level.

VTEX 6.43 adds an integer register `\gexmode`; it will return `0` if GeX is not running and `1` otherwise. While you can assign values to it

```
\gexmode=1999
```

such assignment has no effect on VTEX/GeX operations.

2.1 Troubleshooting

GeX initialization would fail if **GeX** cannot locate the base fonts.

GeX requires the base-13 fonts to be available. Make sure that the `aliasing.pst` file lists them and that the `.tfm` files for them are in `tfm.lib`

The base-13 fonts are the 4 variants of Times, Helvetica, and Courier each, plus the Symbol font.

Chapter 3

Syntax

3.1 Supported PostScript operators

GeX currently supports a large subset of PostScript, including most of Level I implementation and some Level II operators.

Since not the full PostScript operator set is supported, it is possible (and even easy) to write a valid PostScript code which will be rejected by **GeX**; on the other hand, the supported subset includes all the "practically" used operators, so **GeX** would handle correctly most `.eps` that do *appear in nature*. Testing of **GeX** on a large random set of `.eps` files downloaded from Internet shows that **GeX** correctly handles more than 99% of them.

The supported instruction set at this time is:

- `=`
- `abs`
- `add`
- `aload`
- `anchorsearch`
- `and`
- `arc`
- `arcn`
- `arct`
- `arcto`
- `array`
- `ashow`
- `astore`
- `atan`
- `awidthshow`
- `begin`
- `bind`
- `bitshift`
- `cachestatus`, dummy
- `ceiling`
- `charpath`
- `clear`
- `cleartomark`
- `clip`
- `clippath`
- `closefile`
- `closepath`
- `colorimage`
- `concat`
- `concatmatrix`
- `copy`
- `cos`
- `count`
- `countdictstack`
- `counttomark`
- `currentcmykcolor`
- `currentcolorscreen`, simplified
- `currentcolortransfer`
- `currentdash`
- `currentdict`
- `currentfile`
- `currentflat`
- `currentfont`
- `currentglobal`
- `currentgray`

`currenthalftone`
`currenthsbcolor`
`currentlinecap`
`currentlinejoin`
`currentlinewidth`
`currentmatrix`
`currentmiterlimit`
`currentoverprint`, dummy
`currentpoint`
`currentrgbcolor`
`currentscreen`, simplified
`currenttransfer`
`curveto`
`cvi`
`cvlit`
`cvn`
`cvr`
`cvrs`, added in 6.3
`cvs`
`cvx`
`def`
`defaultmatrix`
`definefont`
`defineresource`
`dict`
`div`
`dtransform`
`dup`
`eexec`
`end`
`eoclip`
`eofill`
`eq`
`errordict`
`exch`
`exec`
`executeonly`
`exit`
`exp`
`fill`
`findfont`
`findresource`
`flattenpath`
`floor`
`for`
`forall`
`ge`
`get`
`getinterval`
`grestore`
`grestoreall`
`gsave`
`gt`
`identmatrix`
`idiv`
`idtransform`
`if`
`ifelse`
`image`
`imagemask`
`index`
`initclip`
`invertmatrix`
`itransform`
`known`
`kshow`
`le`
`length`
`lineto`
`ln`
`load`
`log`
`loop`
`lt`
`makefont`
`makepattern`, level2
`mark`
`matrix`
`maxlength`
`mod`
`moveto`
`mul`
`ne`
`neg`
`newpath`
`noaccess`
`not`
`or`
`pathbbox`
`pathforall`
`pop`
`print`
`pstack`
`put`
`putinterval`
`quit`
`rand`
`rcurveto`
`read`
`readhexstring`
`readline`
`readonly`
`readstring`
`rectclip`
`rectfill`
`rectstroke`
`repeat`
`resourcestatus`
`restore`
`reversepath`
`rlineto`

`rmoveto`
`roll`
`rotate`
`round`
`rrand`
`run`
`save`
`scale`
`scalefont`
`search`
`selectfont`
`setcachedevice`
`setcachelimit`
`setcharwidth`
`setcmykcolor`
`setcolor`
`setcolorspace`
`setcolortransfer`
`setdash`
`setflat`
`setfont`
`setglobal`
`setgray`
`sethalftone`
`sethsbcolor`
`setlinecap`
`setlinejoin`
`setlinewidth`
`setmatrix`
`setmiterlimit`
`setoverprint`, dummy
`setpagedevice`
`setpattern`, level2
`setrgbcolor`
`setscreen`
`setstrokeadjust`, dummy
`settransfer`, simplified
`show`
`showpage`
`sin`
`sqrt`
`srand`
`status`
`statusdict`
`stop`
`stopped`
`store`
`string`
`stringwidth`
`stroke`
`strokepath`
`sub`
`systemdict`
`token`
`transform`
`translate`
`truncate`
`type`
`userdict`
`usertime`
`version`
`vmstatus`
`where`
`widthshow`
`xcheck`
`xor`
`xshow`
`xyshow`
`yshow`

3.2 Supported additional operators

The following additional operators are understood by **GeX**.

`.autofontload` — if the integer argument is non-zero, **GeX** will query the `aliasing.pst` file when the `findfont` operator cannot resolve a font name. The default is *not to load* fonts implicitly and substitute Helvetica.

`.currentdigits` — returns the number of emitted fractional digits. Default is 2.

`.loadfont` — load a Type 1 font into the interpreter. The argument should be a string containing a font name; only fonts listed in `aliasing.pst` can be loaded.

`.setdigits` — sets the number of emitted fractional digits in the generated PDF output to an integer argument.

`.setlanguagelevel` — make the interpreter behave as PostScript level 1 or 2, see Language level.

`.settexfont` — used internally; similar to `selectfont` except that it takes a TEX font number rather than a font name as an argument.

`.produceimage` — internal use only (6.3).

`.extend` — see Extending GeXX, 6.3.

`.enabletransfer` — see transfer handling, (6.3).

`.tkwrite` — TEX-**GeX** exchange, see TEX-**GeX** interface via .tk* operators

`.tkread` — TEX-**GeX** exchange, see TEX-**GeX** interface via .tk* operators

`.tklength` — TEX-**GeX** exchange, see TEX-**GeX** interface via .tk* operators

`.loadimage` — see image handling, (6.3).

`.loadimagemask` — see image handling, (6.3).

`.loadcolorimage` — see image handling, (6.3).

`.readimage` — see image handling, (6.4).

`.readrgbimage` — see image handling, (6.4).

3.3 TEX-GeX interface

The **GeX** engine is invoked from VTEX with the `\special`'s:

- `\special{ps: ...}` is used to pass a file to **GeX**.
- `\special{pS: ...}` is used to pass an immediate string to **GeX**.

Note that if you are going to use **GeX** primarily for inclusion of ready-made `.eps` files, you should use a high-level package like `graphics` rather than VTEX's `\special`'s.

In **GeX** mode, VTEX allows to precede a `\special` with the `\immediate` command. `\immediate \special`'s are passed to **GeX** right away, while TEX is still doing formatting.

In version 6.2 there was no way to reuse the `.pdf` code that could be generated during the execution of an `\immediate \special`; however, the **GeX** kernel could be used for computations and the results of such computation could be passed back to TEX. One practical example of this is the letterspacing implementation. In version 6.3 the code generated in *immediate* mode can be reused, see Re-using pdf code.

Versions 6.3+ of VTEX support two similar interfaces.

The first interface has been available since version 6.2. It allows to write to a designated TEX `\toks` register via standard PostScript output operators. This interface is no longer recommended and might be phased out.

To report the results from **GeX** to TEX, set the `\psconsole` integer parameter to a number between 0 and 255:

```
\psconsole=100
```

This instructs **GeX** to write the output to the `\toks###` register rather then to the console (`###` is the number passed via `\psconsole`. By default, `psconsole` is set to `-1` and **GeX** output goes to the `.log` file; when `psconsole` is in the 0–255 range, the output goes to the corresponding `\toks` register.)

The output is always *appended* to `\toks`; multiple outputs from **GeX** get concatenated within the `\toks` register. The only ways to clear the `\toks` register are via a TEX assignment

```
  \toks100={}
```

or by restoring the register on a TEX group closing.

Notice further that you should use the PostScript `print` operator to place information into a `\the\toks` register. This is because the other output operators (`=`, `pstack`) append newline characters and the resulting contents will be difficult to parse in TEX. (Also, the actual format of the output of these operators may change in the future versions of **GeX**.) Since `print` allows only string arguments, you may need to use a `cvs` to convert other PostScript types to strings (see an example below).

If you intend to retrieve information from **GeX** with `\immediate\special`, make sure to flush **GeX** output once at the beginning:

```
   \immediate\special{pS: }
```

The reason for this is that **GeX** is initialized when a first `\special{pS:...}` is encountered; during its first-time initialization it produces some output which normally goes to the `.log` file and you do not want to be appended to the `toks` variable. Supplying an *empty* call to **GeX** forces it to be initialized and the initialization output to be written to the `.log` file.

Here is a small example of using **GeX** inline:

```
\documentclass{article}
\begin{document}

\immediate\special{pS: \the\year\space srand}
  % This also will initialize GeX

\def\rand{%
  \immediate\special{pS: }%
  \psconsole 100
  \immediate\special{pS: rand 10 string cvs print}%
  The number is = \the\toks100.
  \psconsole -1
  \toks100={}
  }

\rand\par\rand\par\rand\par
\end{document}
```

[You may want to cut out the text of this example and try it yourself.]

3.4 TEX-GeX interface via .tk* operators

The new interface available in version 6.3 consists of three additional PostScript operators:

- `.tkread` to read contents of a TEX `\toks` register.
- `.tkwrite` to write to a TEX `\toks` register.
- `.tklength` to find out the length of a TEX `\toks` register.

The syntax of these operators is as follows:

- **`integer string .tkread` → `integer string`**

  where the `integer` parameter should be in the range 0 through 255 and designate a TEX token register; the `string` parameter is the receiving string. In the output, the integer value is the new length of the string; the string contains the contents of the `\toks` register.

- **`integer .tklength` → `integer`**

  where the `integer` parameter should be in the range 0 through 255 and designate a TEX token register; the output integer is the lentgh of the contents of the TEX `\toks` register.

- **`boolean integer string .tkwrite` →**

  where the `boolean` argument defermines if the data should be appended to the `\toks` contents (`true`) or overwrite it (`false`); the `integer` parameter should be in the range 0 through 255 and designate a TEX token register; the contents of the `string` parameter will be placed into the specified `\toks` register.

  Note: `.tkwrite` writes the output in “global” mode (6.43+).

Note: During `.tkread` a `rangeerror` may occur if the `\toks` register contains more characters than can be placed into the receiving string; one can use the `.tklength` operator to find out how big the receiving string should be before allocating it.

Note: Control sequences tokens withing TEX token strings are converted into spaces during `.tkread`; they are counted as single characters in `.tklength`.

Note: Token strings produced by `.tkwrite` contain only tokens with TEX `\catcode` 11 (other).

Note: The tokens produced by `.tkwrite` are always *appended* to the `\toks` register.

Note: These three operators are also available in `GeXX` as the `tkread()`, `tkwrite()` and `tklength()` methods.

3.5 Re-using pdf code

Version 6.3 of VTEX allows to re-use the pdf code that is generated by the `\immediate` form of the `\special{pS:...}` operator.

The base logic is the following:

During the `\immediate` output, the pdf code is written to a temporary stream. Any `\immediate\special{pS:...}` operator opens such a stream (unless it is already open by another operator); the currently opened stream, if exists, is destroyed at the moment of `shipout`. Thus, by default, everything written to immediate streams is lost.

To preserve the contents of an `immediate` stream, use the `\special{ice}` command. This command closes the immediate stream; the new TEX count register `\pdflaststream` can be used to retrieve the handle to the just closed stream. The `\special{!stream \the\pdflaststream}` command can be used to reinsert the frozen stream into the TEX machinery; it will get emitted during the normal shipout.

Notice that the `\special{ice}` command must be issued in the `\immediate` mode (otherwise, there will be no stream to freeze by the time it gets processed); on the other hand, `\special{!stream ...}` must be deterred till the `\shipout`.

If you generate pdf code during the `\immediate` mode, you should realize that the positioning of your code will not be known until the time of the `\shipout`. Thus, the PostScript `currentpoint` is not really defined. The way to overcome this problem is to initialize it to $(0, 0)$ by executing `0 0 moveto` at the beginning of the `\immediate` stream.

The data inserted in the output during the `\special{!stream...}` processing is offsetted by the `currenpoint` as computed during the `\shipout`.

3.6 image handling

Six GeX extention operators:

- `.loadimage`

- `.loadimagemask`
- `.loadcolorimage`
- `.produceimage`
- `.readimage`
- `.readrgbimage`

are designed to enhance the PostScript imaging model and provide a possibility for writing user extentions. Normally, PostScript interpreters handle the image sampling via these operators:

- `image`
- `imagemask`
- `colorimage`

(you can find the detailed meaning and syntax of these operators explained in the PostScript Language Manual.) GeX internally separates the task of obtaining image samples from the task of emitting the image data (generating the Pdf code in the **Pdf** backend) and provides partial operators for both tasks. Specifically,

- `image` ≡ `.loadimage .produceimage`
- `imagemask` ≡ `.loadimagemask .produceimage`
- `colorimage` ≡ `.colorloadimage .produceimage`

with the first operator in each case responsible to collecting the samples, while the second operator (always, `.produceimage`) does the emission. While, for example `image` is implemented as a primitive operator in GeX, it is fully equivalent to a macro defined as

```
/image {.loadimage .produceimage} def
```

The `.loadimage`, `.loadimagemask` and `.loadcolorimage` operators leave a built image on the GeX operand stack; `.produceimage` retrieves it.

This imaging models allows to enhance the image processing sequence with additional operators, inserted between the loading and producing stages. If, for example, `.togray` is an operator that converts color images to grayscale, we can redefine

```
/image {.loadimage .togray .produceimage} def
```

and have the images come up in grayscale. Notice that since the image handle on the operand stack *is not a PostScript object*, operators like `.togray` have to be implemented as usehyGeXX extentions, rather than PostScript macros.

The `transbit` extention library (to appear in 6.5) will provide a set of such imaging filters; impatient users can find sample `transbit` outputs for the `toBright` and `toContrast` filters at

```
ftp.micropress-inc.com/image
```

Filtering the image processing as described above would work only for the image data within `.eps` files; the majority of the images you may want to transform come, however, in bitmapped format. We, therefore, have added some more extentions:

- `.readimage` reads the image data from a bitmapped file in the format identical to the one produced by `.loadimage` and suitable for filters or `.produceimage`.
- `.readrgbimage` is a similar operator which also forces the data to the RGB color model.

For example,

```
(mypic.gif) .readimage .produceimage
```

will load the image from a GIF file and insert it to the output stream.

The supported formats currently are BMP, PCX, GIF, TIFF, JPEG, PNG, and TARGA. Notice that inclusion of bitmapped images via GeX requires the `-ox` switch (unlike the default inclusion via the `\special{G...}` model.)

Note: High-level support for these operators in `graphicx` is not provided in this version of VTEX but will be forthcoming later.

Note: GeXX image API interface details will be published later.

3.7 transfer handling

A problem which arises with some `.eps` images is the use of the `settransfer` PostScript and related operators. The problem is that these operators are used for both device-dependant and device-independant color manupulations. The first usage is more common and is essentially for minor color adjustments. In such situations the best strategy for producing device-independant `.pdf` files is to disregard the transfer altogether. This is the default behaviour of GeX (and of the Acrobat Distiller).

However, in some (fortunately, rare) `.eps` files the same operators are used to effect major device-independant adjustments. An example of such an adjustment would be to invert a black-and-white picture; this can be done with the

```
{ 1 exch sub } settransfer
```

PostScript code snippet. Disregarding this would produce an inverted image. Thus, both Acrobat Distiller and GeX allow to change this behaviour. In the case of Distiller, the override is a global Job option which will apply to all parts of a document; GeX allows to override only the handling of an individual image. This is accomplished with the extention

int `.enabletransfer`

operator. With a 0 argument, `.enabletransfer` disables processing of `transfer` code; a non-zero argument enables `transfer`processing.

Here is an example of a small `.eps` file that uses transfer code (thanks, CdA):

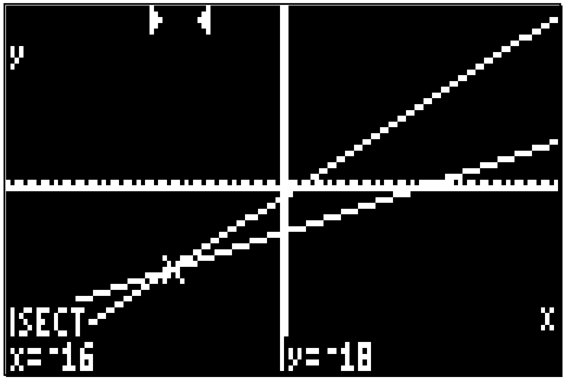
```
%!PS-Adobe-2.0 EPSF-2.0
%%BoundingBox: 20.0 20.0 172.0 122.0
%%Creator: WLINK85.EXE
%%Title:TEST.EPS
%%CreationDate:12/26/98111:082
%%EndComments

% EPS file to print LCD of TI-85 Graphics Calculator

save

   % Define constants ------------------------------------------------

   % Width & height of printed LCD. Scale these as necessary.
   /wid  150 def            % Actual screen width  = 150 (52.8  mm)
   /ht   100 def            % Actual screen height = 100 (35.15 mm)

   /picstr 1 string def     % Blank string used by ScreenImage procedure.

   % Define procedures -----------------------------------------------
   /inch {72 mul} def

   % Set position for lower-left corner of printout
   /setposn { 21 21 translate} def

   % Draw a border around the image.
   /drawborder
       { gsave
            setposn
            [1 0 0 1 -0.5  0.5] concat
            newpath
               0         0               moveto
               0         ht              rlineto
               wid       0               rlineto
               0         ht 0 exch sub   rlineto
               closepath
               0.48 setlinewidth   % 2/300 inch
            stroke
         grestore
       } def

   /concatprocs
```

```
     { /proc2 exch cvlit def
       /proc1 exch cvlit def

       /newproc proc1 length proc2 length add
        array def
       newproc 0 proc1 putinterval
       newproc proc1 length proc2 putinterval
       newproc cvx
     } def

  /ScreenImage                      % Image is  128 pixels per row, 64 rows.
     {  128 64 1 [ 128 0 0 -64 0 64]
         { currentfile picstr readhexstring pop }
         image
     } def

  % ============================ Main Program ============================

  gsave
     setposn
     wid ht scale
     % Set currenttransfer to print bit map in black on white.
     {1 exch sub} currenttransfer     % Use {dup} and {1 exch sub} as inverses.
     concatprocs
     settransfer

     ScreenImage

00 00 00 00 40 02 00 01 80 00 00 00 00 00 00 00
00 00 00 00 60 06 00 01 80 00 00 00 00 00 00 00
00 00 00 00 70 0e 00 01 80 00 00 00 00 00 00 06
00 00 00 00 60 06 00 01 80 00 00 00 00 00 00 18
00 00 00 00 40 02 00 01 80 00 00 00 00 00 00 60
00 00 00 00 00 00 00 01 80 00 00 00 00 00 03 80
00 00 00 00 00 00 00 01 80 00 00 00 00 00 0c 00
50 00 00 00 00 00 00 01 80 00 00 00 00 00 30 00
50 00 00 00 00 00 00 01 80 00 00 00 00 00 c0 00
20 00 00 00 00 00 00 01 80 00 00 00 00 03 00 00
40 00 00 00 00 00 00 01 80 00 00 00 00 0c 00 00
00 00 00 00 00 00 00 01 80 00 00 00 00 30 00 00
00 00 00 00 00 00 00 01 80 00 00 00 00 c0 00 00
00 00 00 00 00 00 00 01 80 00 00 00 03 00 00 00
00 00 00 00 00 00 00 01 80 00 00 00 0c 00 00 00
00 00 00 00 00 00 00 01 80 00 00 00 30 00 00 00
00 00 00 00 00 00 00 01 80 00 00 00 c0 00 00 00
00 00 00 00 00 00 00 01 80 00 00 03 00 00 00 00
```

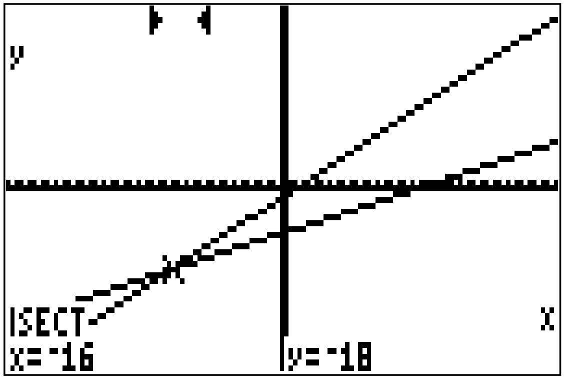
```
00 00 00 00 00 00 00 01 80 00 00 0c 00 00 00 00
00 00 00 00 00 00 00 01 80 00 00 30 00 00 00 00
00 00 00 00 00 00 00 01 80 00 00 c0 00 00 00 00
00 00 00 00 00 00 00 01 80 00 03 00 00 00 00 00
00 00 00 00 00 00 00 01 80 00 0c 00 00 00 00 00
00 00 00 00 00 00 00 01 80 00 30 00 00 00 00 06
00 00 00 00 00 00 00 01 80 00 c0 00 00 00 00 78
00 00 00 00 00 00 00 01 80 03 00 00 00 00 07 80
00 00 00 00 00 00 00 01 80 0c 00 00 00 00 78 00
00 00 00 00 00 00 00 01 80 30 00 00 00 07 80 00
00 00 00 00 00 00 00 01 80 c0 00 00 00 78 00 00
00 00 00 00 00 00 00 01 83 00 00 00 07 80 00 00
b6 d6 da da db 5b 6b 6d ed ad b5 b5 fe b6 d6 da
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff fe
00 00 00 00 00 00 00 01 c0 00 00 78 00 00 00 00
00 00 00 00 00 00 00 03 80 00 07 80 00 00 00 00
00 00 00 00 00 00 00 0d 80 00 78 00 00 00 00 00
00 00 00 00 00 00 00 31 80 07 80 00 00 00 00 00
00 00 00 00 00 00 01 c1 80 78 00 00 00 00 00 00
00 00 00 00 00 00 06 01 87 80 00 00 00 00 00 00
00 00 00 00 00 00 18 01 f8 00 00 00 00 00 00 00
00 00 00 00 00 00 60 0f 80 00 00 00 00 00 00 00
00 00 00 00 00 01 80 f1 80 00 00 00 00 00 00 00
00 00 00 00 00 06 0f 01 80 00 00 00 00 00 00 00
00 00 00 00 00 18 f0 01 80 00 00 00 00 00 00 00
00 00 00 00 08 ef 00 01 80 00 00 00 00 00 00 00
00 00 00 00 05 f0 00 01 80 00 00 00 00 00 00 00
00 00 00 00 0f 00 00 01 80 00 00 00 00 00 00 00
00 00 00 00 fd 00 00 01 80 00 00 00 00 00 00 00
00 00 00 0f 68 80 00 01 80 00 00 00 00 00 00 00
00 00 00 f1 80 00 00 01 80 00 00 00 00 00 00 00
00 00 0f 06 00 00 00 01 80 00 00 00 00 00 00 00
00 00 f0 18 00 00 00 01 80 00 00 00 00 00 00 00
00 00 00 60 00 00 00 01 80 00 00 00 00 00 00 00
4d cd c1 80 00 00 00 01 80 00 00 00 00 00 00 14
51 10 86 00 00 00 00 01 80 00 00 00 00 00 00 08
49 90 98 00 00 00 00 01 80 00 00 00 00 00 00 08
45 10 80 00 00 00 00 01 80 00 00 00 00 00 00 14
59 cc 80 00 00 00 00 01 80 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00
00 02 30 00 00 00 00 01 00 02 70 00 00 00 00 00
57 36 40 00 00 00 00 01 57 36 50 00 00 00 00 00
20 02 70 00 00 00 00 01 50 02 70 00 00 00 00 00
27 02 50 00 00 00 00 01 27 02 50 00 00 00 00 00
50 07 70 00 00 00 00 01 40 07 70 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
   grestore
   drawborder

restore
```

After `0 .enabletransfer` (and by default), this image will produce



while `1 .enabletransfer` will result in



Notice that the `.enabletransfer` setting obeys PostScript `gsave`/`grestore`; we do not provide any way for a PostScript program to read the setting.

3.8 Language level

GeX can function as either Level 1 or Level 2 PostScript interpreter. While PostScript Level 2 is a superset of Level 1, EPS files are not necesserily processed the same way (and produce the same image) in Level 1 and 2 modes. This happens because EPS files are programs, which can (and often do) check for the existance or value of some PostScript operator (often, `languagelevel`) and then execute totally different code depending on the results.

Thus, while in most cases the user might prefer to initialize the GeX interpreter in Level 2 mode, there may be cases when an individual drawing should be processed in Level 1 PostScript.

To initialize VTEX/GeX in Level 1 mode, use the `-ox` switch; to initialize VTEX/GeX in Level 2 mode, use the `-ox2` switch. To switch GeX mode, use the

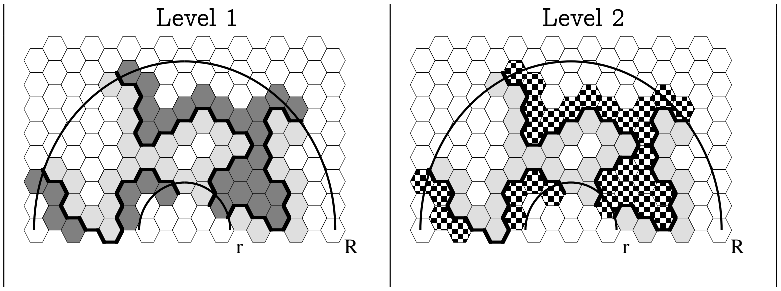
```
\special{pS: N .setlanguagelevel}
```

command (`N` is either 1 or 2).

The execution of `.setlanguagelevel` switches between the two versions of `systemdict` that exist in GeX; it also affects PostScript parsing and memory management rules (for example, the `<<` operator exists only Level 2). Switching between levels should be done at the outside-most level of the execution.

Note that the distinction between the Level 1 and Level 2 PostScripts is not well defined. In some Level 1 PostScript interpreters, `languagelevel` is not defined, in some others, it is defined and returns 1 (GeX Level 1: not defined). The `colorimage` operator has been defined in some versions of Level 1 PostScript but not all (GeX: defined).

Here is an example of a drawing that comes up differently in Level 1 and Level 2:



Note: This example uses the `paths.eps` sample file.

Chapter 4

GeXX

4.1 Extending GeXX

The major new feature added in VTEX 6.3 is GeXX. The second "X" stands for eXtensible. With GeXX you can supplement the existing set of the PostScript operators with new constructs, implemented in C.

This feature is applicable to both VTEX/Windows and VTEX/Linux implementations. The terminology in this document is tailored to Windows users; Linux users should have no problem with it as long as they translate a few platform-dependant terms; for example, DLL's under Linux are just Shared Objects (.SO).

To enrich **GeX** with new operators, you should

- Implement them within a C-language DLL, see Writing an implementation DLL.
- Call the `.extend` operator to load the new language extensions, see Loading an extension DLL
- Provide additional TEX and/or PostScript code for easy access to the new operators.

VTEX 6.3 distribution includes an example of such an extension for drawing the PieCharts.

Note: To be visible to the VTEX compiler, the extension DLL should be placed into the `VTEX\BIN\GEX` subdirectory.

4.2 Writing an implementation DLL

In the current version, extention DLL's must be implemented with Borland C compiler (version 5.0 or newer).

An extention `.DLL` must export the following three functions:

- `extern "C" WINAPI int Version()`

  This function must return the version of the interface as defined in `gexi.h`. If the version returned by the extention DLL does not match the version needed for the VTEX compiler, the DLL is not loaded.

- `extern "C" WINAPI int Count()`

  which returns the number of extensions implemented (≥ 1).

- `extern "C" WINAPI int Names(int i, char **s, void **p)`

  which returns the PostScript names and memory addresses of the new extensions. This function will be called with the index argument `i` and should fill up the parameter `s` with the name of the `i`'s operator implemented and the parameter `p` with the address of the operator implementation. Notice that `i` is counted from 0 through $\mathrm{Count}() - 1$.

Here is a sample implementation of these two functions:

```
extern "C" WINAPI int Names(int i, char **s, void **p)
{
  if (i == 0)
  {
     *s = "bluestretch";
     *p = (void*)bluestretch;
  }
  else
  {
     *s = NULL;
     *p = NULL;
  }
  return ((*s) != NULL);
};

extern "C" WINAPI int Count()
{
  return 1;
}
```

The code above adds new PostScript operator `bluestretch`. Notice that `.extend` will define it in the current top PostScript dictionary.

4.3 Writing an extention operator

An extension operator should be declared as an `int` function; its solo argument is the **GeX** interface structure, `GEXI`. The function should return 0 in case of success, and a non-zero value otherwise (triggering a PostScript `internal` error).

It is recommended not to remove arguments from the PostScript stack until you are sure that the extention operator will return success.

`void bluestretch(GEXI *g)`

The interface structure allows a C-language extention to interract with the PostScript environment via C-versions of the PostScript operators.

Below is a sample implementation of a simple extention operator, `bluestretch`:

```
#define MAXLEN 19
#define RESOLUTION 256

int bluestretch(GEXI *g)
{
  double width = 0;
  double height = 0;
  char Buf[MAXLEN+1];

  int len;
  g->tkread(99, Buf, MAXLEN, &len);
  if (sscanf(Buf, "%lf%lf", &height, &width) != 2)
    return;
  double step = width/(double)RESOLUTION;

  double x, y;
  g->currentpoint(&x, &y);   // undefined in the \immediate mode
  g->gsave();

  g->newpath();
  for (int i = 0; i < RESOLUTION; i++)
  {
    g->moveto(x, y);
    g->lineto(x+step, y);
    g->lineto(x+step, y+height);
    g->lineto(x, y+height);
    g->setrgbcolor(0.0, (double)i/(double)RESOLUTION, 1.0);
    g->fill();
    x += step;
  }
  g->grestore();
return 0;
}
```

Notice that `bluestretch` takes the actual parameters from TeX rather than PostScript via a hardwired `\toks99` register. [A proper PostScript extention should take the arguments from the PostScript stack.]

The `GEXI` interface functions are listed below. Note that the list of the interface functions is likely to grow in future versions; always look at `gexi.h` for current information.

```
//---------------- GEXI.H listing --------------
// Copyright (C) 1999 by MicroPress, Inc.
// All Rights Reserved
// By Michael Vulis
#define GEXI_VERSION 630

struct GEXI {
        long size;
        long version;
// GeX/PostScript calls
        int (*print)(char*);
        int (*tkwrite)(int n,char*,int append);
        int (*tkread)(int n,char*,int maxlen,int *newlen);
        int (*tklength)(int n,int *newlen);
        int (*currentpoint)(double*,double*);
        int (*moveto)(double,double);
        int (*lineto)(double,double);
        int (*rlineto)(double,double);
        int (*curveto)(double,double,double,double,double,double);
        int (*rcurveto)(double,double,double,double,double,double);
        int (*arc)(double,double,double,double,double);
        int (*arcn)(double,double,double,double,double);
        int (*arct)(double,double,double,double,double);
        int (*newpath)();
        int (*setlinewidth)(double);
        int (*currentlinewidth)(double*);
        int (*setlinecap)(int);
        int (*currentlinecap)(int*);
        int (*setlinejoin)(int);
        int (*currentlinejoin)(int*);
        int (*setmiterlimit)(double);
        int (*currentmiterlimit)(double*);
        int (*closepath)();
        int (*stroke)();
        int (*fill)();
        int (*eofill)();
        int (*clip)();
        int (*eoclip)();
        int (*gsave)();
        int (*grestore)();
        int (*setgray)(double);
        int (*currentgray)(double*);
        int (*setrgbcolor)(double,double,double);
        int (*currentrgbcolor)(double*,double*,double*);
        int (*setcmykcolor)(double,double,double,double);
        int (*currentcmykcolor)(double*,double*,double*,double*);
// Operand Stack interface
        int (*CountOstack)(int *);
        int (*GetOstackInteger)(int,int*);
        int (*GetOstackNumeric)(int,double*);
        int (*GetOstackReal)(int,double*);
        int (*GetOstackBoolean)(int,int*);
        int (*GetOstackString)(int,int*,unsigned char**);
        int (*OstackPop)(int);
// Private part
        } gexi_;
```

All the functions listed above return zero if the call is successful and non-zero error code if the call fails. A call would fail in exactly the same cases when a corresponding PostScript operator would fail.

Majority of the methods provided by `GEXI` correspond one-to-one to either PostScript operators with the same names, or **GeX** extention operators (`.tkread`, for example). The only exception to this at this time are the methods that deal with the PostScript operand stack; these methods are used to retrieve (and, later, pop) the arguments provided on the operand stack.

Note: The `GEXI` structure is likely to change in the future. The changes to the structure can be tracked by checking the version number; this will correspond to the VTEX version. An attempt to execute an extension with a wrong version of the `.DLL` is likely to crash the system; the version control is implemented to prevent this from happening.

4.4 PS VM access

Very limited access to PostScript run-time stack is provided in this version. Use the `CountOstack()` function to find out the number of elements on the PostScript stack. You can retrieve individual elements only if they are of PostScript type `integer`, `real`, `boolean` or `string`; the `GetOstackInteger()`, `GetOstackReal()`, `GetOstackBoolean()`, and `GetOstackString()` functions are provided to retrieve elements of these types. The `GetOstackNumeric()` can retrieve either `integer` or `real` value.

The first argument of these functions is an index to the operand stack; the top elements resides at $CountOstack() - 1$. The other argument(s) are for recieving the value; in the case of `GetOstackString()` both the string length and the pointer to the string are returned.

These functions will return an error if called for a wrong type.

Dereference the string value pointer only for reading; modifying a string from-within an extention might lead to unpredictable results.

A more sophisticated interface may be provided in a future version.

4.5 Loading an extension DLL

To load an extension `.DLL`, execute the `.extend` operator.

The syntax is:

`string .extend` → `integer`

where the `string` argument contains the name of the `.dll` file with the language extentions, the returned `integer` is the number of extention operators loaded.

The `.extend` operator will fail with an error if

- the specified DLL cannot be found.
- the specified DLL does not export all three required functions (`Count()`, `Version()`, `Names()`).
- the version returned by the DLL does not match the version of the VTEX compiler.

In all three cases, `.extend` will cause a PostScript `fileerror`.

4.6 PieChart

PieChart is a more sophisticated example of using GeXX.

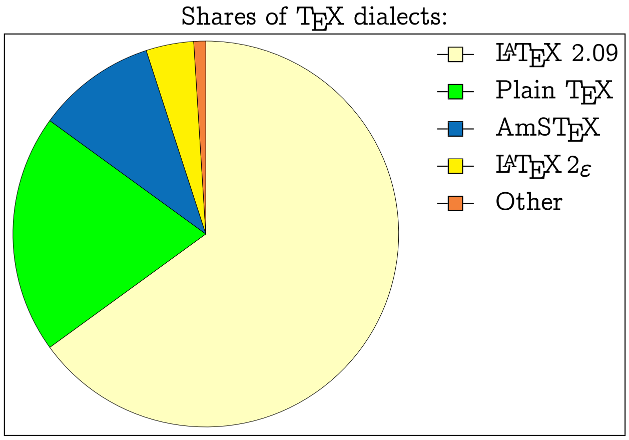
PieChart implements MS Word-like PieChart in TEX. The implementation consists of

- `PieChart.DLL`, the extension library.
- `PieChart.Sty`, a LATEX2E style for using PieChart.

Here is a sample code for using PieChart:

```
%% Define some colors
\definecolor{lightyell}{rgb}{1,1,0.75}
\definecolor{peach}{cmyk}{0,0.50,0.70,0}
\definecolor{orange}{cmyk}{0,0.61,0.87,0}
\definecolor{navyblue}{cmyk}{0.94,0.54,0,0}

\begin{center}
Shares of \TeX\ dialects:\par
\fbox{\begin{PieChart}[rt]{2in}
\PieSlice{lightyell}{65}{\LaTeXe}
\PieSlice{green}{20}{Plain \TeX}
\PieSlice{navyblue}{10}{AmS\TeX}
\PieSlice{yellow}{4}{\LaTeX\ 2.09}
\PieSlice{orange}{1}{Other}
\end{PieChart}}
\end{center}
%%
```

and a sample PieChart produced by this extension:



4.7 PieChart source

We provide full source of the PieChart plugin. The source has been divided into three parts:

- `gexstd.h` implements the required portion of the API; and hides the platform-dependant parts. As long as the extension `.dll` implements a single PostScript extension, you should be able to reuse this file in your own plugins. **We strongly recommend doing exactly this since this will make your plugin useful on both the Windows and the Linux platforms.**
- `piechart.cpp` is the PieChart-related portion of the source. The only API-required activity is to include (at the beginning) `gexstd.h` and execute (at the end) the the `stdgexapi` macro to declare the new extension.
- `piechart.sty` is the TeX interface for calling PieChart.

Here is the `gexstd.h` source. Linux users should notice the use of `#ifdef linux` in the source; the Linux makefile defines it.

```
// Copyright (C) 1999 by MicroPress, Inc.
// All Rights Reserved
// By Michael Vulis and Alex Kostin
```

```
#ifdef __WIN32__
#include <windows.h>
#else
#include <ctype.h>
#endif
#include <stdio.h>
#include <math.h>
#pragma hdrstop

#ifdef linux
  #define DLLFUNC(type) type
#elif defined(__BORLANDC__)
  #define DLLFUNC(type) type __export WINAPI
  #define DLLENTRY DllEntryPoint
#else // _MSC_VER_
  #define DLLFUNC(type) __declspec(dllexport) type WINAPI
  #define DLLENTRY DllMain
#endif

#define CFUNC extern "C"

#ifndef ushort
  #define ushort unsigned short
#endif
#ifndef byte
  #define byte unsigned char
#endif
#ifndef TRUE
  #define TRUE 1
  #define FALSE 0
#endif
#ifndef M_PI
  const double M_PI = 2.0*acos(0.0);
#endif

#include "gexi.h"

#define stdgexapi(psname,cppname) \
CFUNC DLLFUNC(int) Names(int i, char** s, void** p) \
{ \
  if (i == 0) \
  { \
    *s = psname; \
    *p = (void*)cppname; \
  } \
  else \
  { \
    *s = NULL; \
    *p = NULL; \
  } \
  return ((*s) != NULL); \
}; \
\
CFUNC DLLFUNC(int) Version() { \
  return GEXI_VERSION; \
} \
\
CFUNC DLLFUNC(int) Count() \
{ \
  return 1; \
} \

#ifdef __WIN32__
CFUNC BOOL WINAPI DLLENTRY(HINSTANCE /* hinst */,
  DWORD /* wDataSeg */, LPVOID /* cmdline */)
{
  return TRUE;
}
#endif
// End of GeXstd.h source
```

Here is the actual implementation source for PieChart:

```
// Copyright (C) 1999 by MicroPress, Inc.
// All Rights Reserved
// By Alex Kostin

#include "gexstd.h"

#define NOCMYK -128

typedef struct
        {
          double red;
          double green;
          double blue;
          double black;
          double percent;
        } TSliceData;

//const double pt2bp = 7200.0/7227.0;
```

```
const double pt2bp = 1.0;


int isnumber(char c)
{
  return isdigit(c) || c == '.' || c == '-' || c == '+' ? TRUE : FALSE;
}


int GetSliceData(char** StrPtr, TSliceData* slicedata)
{
  if (!*StrPtr)
    return FALSE;

  while (isspace(**StrPtr))
    (*StrPtr)++;

  if (**StrPtr != '(')
    return FALSE;
  else
    (*StrPtr)++;

  char* CurStr = *StrPtr;
  while (**StrPtr != ')')
    if (!**StrPtr)
      return FALSE;
    else
      (*StrPtr)++;

  *(*StrPtr)++ = '\0';

  int ReadComp = sscanf(CurStr, "%lf%lf%lf%lf", &slicedata->red,
    &slicedata->green, &slicedata->blue, &slicedata->black);

  if (ReadComp == 3)
    slicedata->black = NOCMYK;
  else
    if (ReadComp != 4)
      return FALSE;

  while (isspace(**StrPtr))
    (*StrPtr)++;

  if (**StrPtr != ':')
    return FALSE;
  else
```

```
    (*StrPtr)++;

  if (sscanf(*StrPtr, "%lf", &slicedata->percent) != 1)
    return FALSE;

  while (isnumber(**StrPtr))
    (*StrPtr)++;

  return TRUE;
}


int GetTokenRegisterNumber(GEXI* GeX, int *Value)
{
  int StackIndex, result;
  GeX->CountOstack(&StackIndex);
  if (StackIndex<=0)
    return -1;
  StackIndex--;
  result=GeX->GetOstackInteger(StackIndex,Value);
  if (result)                // Things went OK.
     GeX->OstackPop(1);
  return result;
}


int piechart(GEXI* GeX)
{
  int TokenRegisterNumber;
  int result;
  result=GetTokenRegisterNumber(GeX, &TokenRegisterNumber);
  if (result)                // Things failed.
     return result;

  int Length;
  GeX->tklength(TokenRegisterNumber, &Length);
  unsigned char* Buf = new unsigned char[Length+1];
  GeX->tkread(TokenRegisterNumber, Buf, Length, &Length);

  TSliceData SliceData;
  char* StrPtr = (char*)Buf;
  double PieRadius;

  if (sscanf(StrPtr, "%lf", &PieRadius) != 1)
  {
    delete [] Buf;
    return -1;
  }
  PieRadius *= pt2bp/2.0;
  while (isspace(*StrPtr++));
  while (isnumber(*StrPtr++));

  double X0, Y0;
  GeX->currentpoint(&X0, &Y0);    //Not defined in immed mode
  X0 += PieRadius;
  Y0 += PieRadius;

  double CurAngle = 90.0;

  GeX->gsave();
  while (GetSliceData(&StrPtr, &SliceData))
  {
    GeX->newpath();
    GeX->moveto(X0, Y0);
    double SliceAngle = 3.6*SliceData.percent;
    double StartingAngle = M_PI*(CurAngle-SliceAngle)/180.0;
    GeX->lineto(X0+PieRadius*cos(StartingAngle),
      Y0+PieRadius*sin(StartingAngle));
    GeX->arc(X0, Y0, PieRadius, CurAngle-SliceAngle, CurAngle);
    CurAngle -= SliceAngle;
    GeX->closepath();

    GeX->gsave();
    if (SliceData.black == NOCMYK)
      GeX->setrgbcolor(SliceData.red, SliceData.green, SliceData.blue);
    else
      GeX->setcmykcolor(SliceData.red, SliceData.green, SliceData.blue,
        SliceData.black);
    GeX->fill();
    GeX->grestore();

    GeX->setrgbcolor(0, 0, 0);
    GeX->stroke();
  }
  GeX->grestore();

  delete [] Buf;
  return 0;
}

stdgexapi("piechart",piechart)
```

```
// End of PieChart.CPP source
```

The following `makefile` can be used with `gcc` under Linux:

```
TARGET_NAME := ../piechart
TARGET_TYPE := so
CXX = g++
CXXINCLUDES := -I../common/
CXXFLAGS := -g -fpic -Dlinux $(CXXINCLUDES)

LD := ld

SRCS := piechart.cpp
OBJS := $(SRCS:.cpp=.o)

.PHONY: all
all:    $(TARGET_NAME).$(TARGET_TYPE)

$(TARGET_NAME).so:      $(OBJS)
        $(LD) -G -o $@ $^

%.o:    %.cpp
        $(CXX) $(CXXFLAGS) -c $<

.PHONY: clean
clean:
        rm *.o *.so;
```

while the one below is suitable for BC 4.5 or BC 5.0 under Windows:

```
.AUTODEPEND
#
# Borland C++ path
#
BCPATH = C:\BC5

#
# Borland C++ tools
#
BCC32   = Bcc32 +BccW32.cfg
TLINK32 = TLink32

#
# Options
#
OPTIONS = -u -I$(BCPATH)\INCLUDE -D_RTLDLL;_BIDSDLL;
LINKOPTIONS = -Tpd -ap -c -L$(BCPATH)\LIB
```

```
#
# Dependency List
#
Dep_piechart = piechart.dll

piechart : BccW32.cfg $(Dep_piechart)
  echo Success

Dep_piechartddll = \
   piechart.obj

piechart.dll : $(Dep_piechartddll)
  $(TLINK32) @&&|
 /v -Tpd -ap -c -L$(BCPATH)\LIB -x +
$(BCPATH)\LIB\c0d32.obj+
piechart.obj
$<,$*
$(BCPATH)\LIB\bidsfi.lib+
$(BCPATH)\LIB\import32.lib+
$(BCPATH)\LIB\cw32i.lib

|
piechart.obj :  piechart.cpp
  $(BCC32) -c @&&|
  $(OPTIONS) -o$@ piechart.cpp
|

# Compiler configuration file
BccW32.cfg : piechart.mak
   Copy &&|
-w
-R
-v
-WM-
-vi
-H-
-O-d
-O
-Ob
-Oe
-Og
-Oi
-Ol
-Om
-Ot
```

```
-Op
-Ov
-k-
-Z
-WCD
-u
| $@
```

The final part of the source is the **`piechart.sty`** style; you can find it in the `VTEX\L2E` subdirectory.

Chapter 5

Imaging plugins

5.1 General

GeX imaging plugins are used to adjust images during their inclusion into the pdf document. Two sample imaging plugins are included in VTEX6.4:

- TransBit: bright/contrast/colorspace adjustment
- Degrade: image downsampling

Supplied imaging plugins are partially supported by the `graphicx` package.

The examples in this chapter use a single image file, `macaw.jpg`. Normally, this image can be included with the `graphicx` command:

```
\includegraphics[scale=0.4]{macaw.jpg}
```

which will result in

Note: This (default) image loading is implemented via the `\special{G...}` command and does not require GeX.

We can also load the same image via GeX with

```
\includegraphics[scale=0.4,viagex]{macaw.jpg}
```

which is essentially equivalent to

```
\hbox to 2in{\hss
\special{pS: (macaw.jpg) .readimage .produceimage}\hss}
```

While there are subtle differences between these two procedures, here they will produce identical images.

The image plugins work only in the GeX model. The general idea is to insert an image plugin action between the `.readimage` and the `.produceimage` operators, for example with

```
\hbox to 2in{\hss
\special{pS: (macaw.jpg) .readimage
  (toBright 10) transbit
.produceimage}\hss}
```

Since the image plugin functionality is not implemented in the core GeX, a plugin should be loaded with the `.extend` operator. However, if you are using the `graphicx` package, the supplied plugins will be loaded on demand.

Image plugins are a brand new feature of VTEX; errors are possible and both the documentation and the specific syntax are subject to revisions.

5.2 TransBit

The **Transbit** plug-in uses the GeXX image interface to implement several image manupulation functions, including adjustment to the brightness and the contrast of an image, and replace the color space.

The functionality of **Transbit** is supported by `graphicx` macro package.

The `transbit` plugin can be loaded by

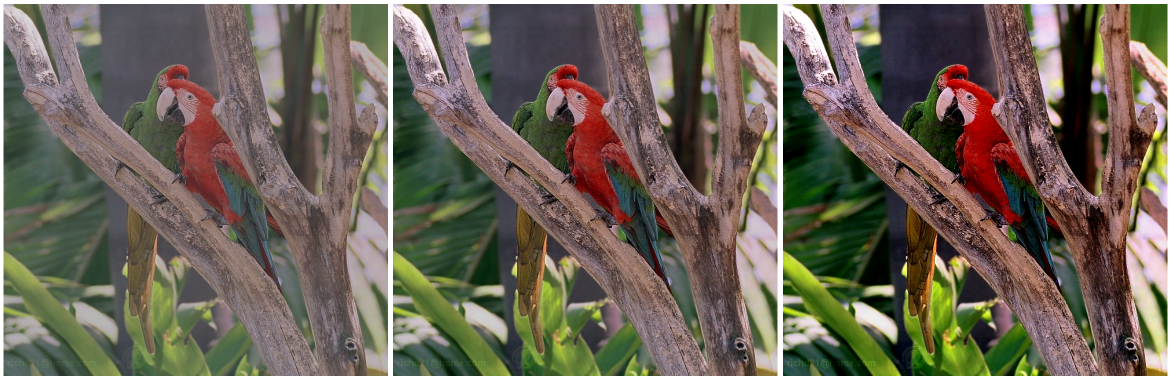
```
\special{pS: (transbit) .extend }
```

The first examples modifies the contrast the image. The three images below were produced with these commands:

```
\includegraphics[width=2in,viagex,contrast=-0.3]{macaw.jpg}
\includegraphics[width=2in,viagex,contrast=0]{macaw.jpg}
\includegraphics[width=2in,viagex,contrast=+0.3]{macaw.jpg}
```

which, on the low level, expand into

```
\special{pS: gsave currentpoint translate 0.2 0.2 scale (macaw.jpg)
.readimage (toContrast -30) transbit
.produceimage grestore}
\special{pS: gsave currentpoint translate 0.2 0.2 scale (macaw.jpg)
.readimage (toContrast   0) transbit
.produceimage grestore}
\special{pS: gsave currentpoint translate 0.2 0.2 scale (macaw.jpg)
.readimage (toContrast +30) transbit
.produceimage grestore}
```



In a similar fashion,

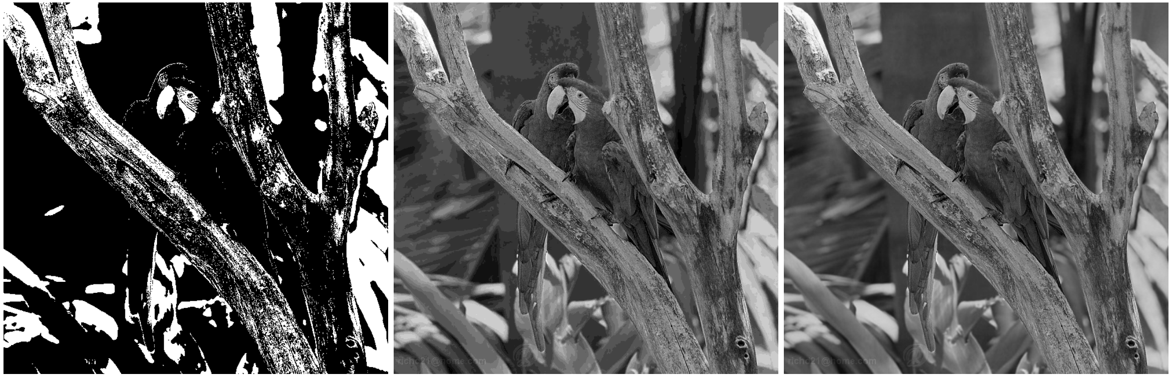
```
\includegraphics[width=2in,viagex,brightness=-0.3]{macaw.jpg}
\includegraphics[width=2in,viagex,brightness=0]{macaw.jpg}
\includegraphics[width=2in,viagex,brightness=+0.3]{macaw.jpg}
```

will invoke **Transbit** to produce

The final functionality of **Transbit** is the Color Space conversion. Usually, you would want it to decrease the size of the image and to prepare the document for printing on a non-color device. For example,

```
\includegraphics[width=2in,viagex,colorspace=bw]{macaw.jpg}
\includegraphics[width=2in,viagex,colorspace=grayscale 16]{macaw.jpg}
\includegraphics[width=2in,viagex,colorspace=grayscale 256]{macaw.jpg}
```

will yield



5.3 Degrade

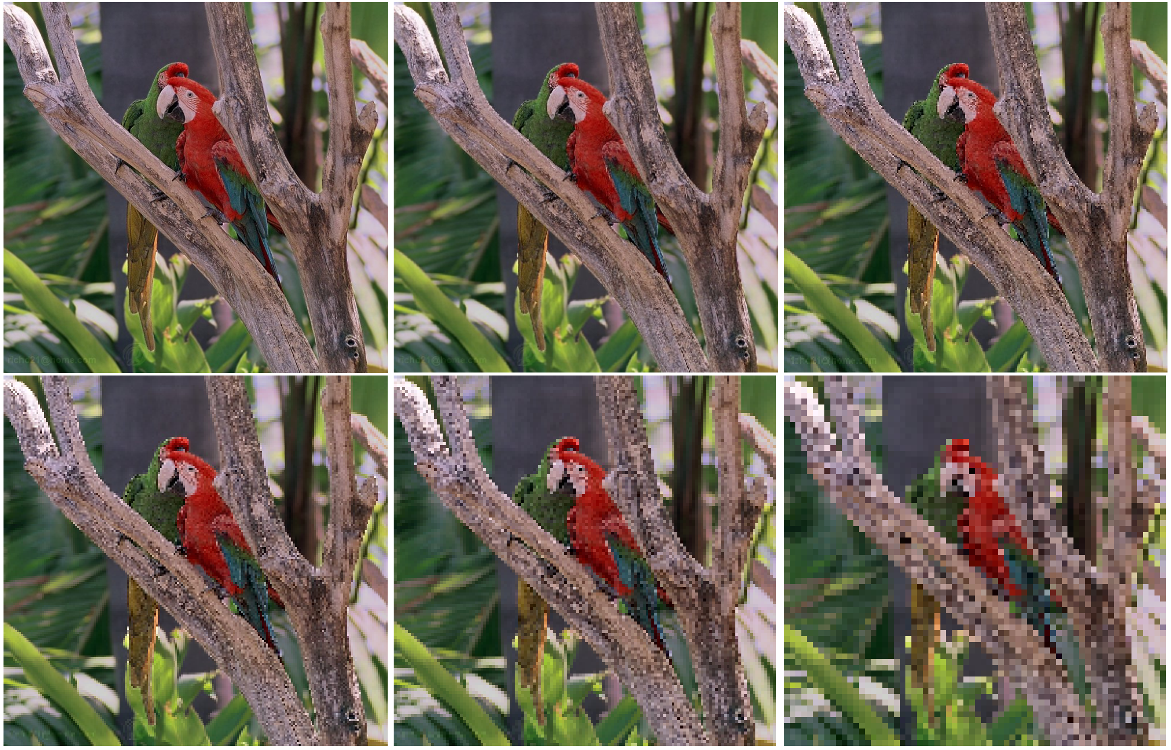
The **Degrade** plugin is used to downsample (reduce the number of pixels) images. The only purpose of using this plugin is to reduce the size of the `.pdf` file; the price you are paying is the reduced quality of the image.

Degrade is supported by `vtex.def` via the `degrade=` key; the affiliated value is a number in the range 0 through 1 with 0 effectively destroying the image (removing all pixels) and 1 leaving it intact. For example:

```
\includegraphics[width=2in,viagex]{macaw.jpg}
\includegraphics[width=2in,viagex,degrade=0.6]{macaw.jpg}
\includegraphics[width=2in,viagex,degrade=0.4]{macaw.jpg}
```

```
\includegraphics[width=2in,viagex,degrade=0.3]{macaw.jpg}
\includegraphics[width=2in,viagex,degrade=0.2]{macaw.jpg}
\includegraphics[width=2in,viagex,degrade=0.1]{macaw.jpg}
```

yields



Notice how the (uncompressed) sizes of the six embedded images decrease:

Coeff.	Data Size
1.0	1276872
0.6	458496
0.4	203520
0.3	114624
0.2	50688
0.1	12672

Typically, it is the large images that should be considered for degrading. Which coeffient will produce the best results can be determined only experimentally. It might be a good idea to print documents that contain downsampled images to see if the loss of quality is tolerable.

Chapter 6

Bugs

Being a pilot implementation with source of about 15000 lines of code, **GeX** undoubtly has many bugs. About a hundred of them were fixed since the happy moment in July when we thought it more-or-less works (and all fall apart on testing of huge set of *real-life* `.eps`'s from all kinds of sources.

In the aggravation of fixing what-we-thought was a working program, we discovered that the bugs came in three flavors:

- Our bugs
- Peculiarities (often undocumented) of PostScript language
- Bugs (or problems) in Adobe Software.

Bugs of our implementation (important for us for sentimental reasons) are not worth discussing here; but some of the other bugs are definitly worthwhile mentioning.

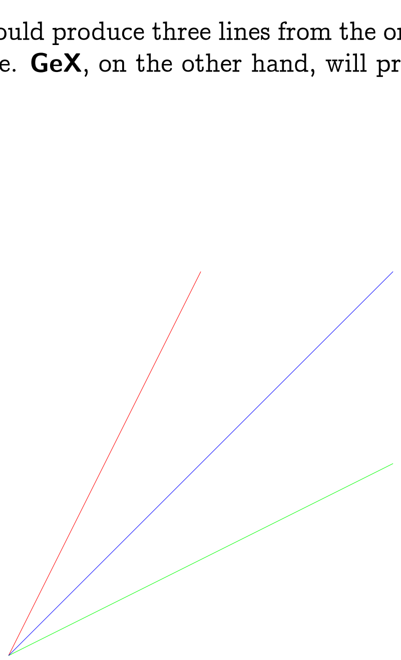
6.1 Degenerate matrices

Near-degenerate matrix transforms cause a serious problem with the Acrobat's 16-bit computational limit. It can be shown that the problem is not solvable correctly in general; and Adobe Acrobat Distiller would fail on degenerate transforms.

The example file

```
% lwid.ps
0 0 moveto
gsave 100 200 lineto 2 3 scale 1 0 0 setrgbcolor stroke grestore
gsave 200 100 lineto 0.5 0.3 scale 0 1 0 setrgbcolor stroke grestore
gsave 200 200 lineto 0 0 1 setrgbcolor
   [0.186718 -0.565306 0.873838 -2.64563 0 0] setmatrix
   stroke grestore
showpage
```

should produce three lines from the origin. Distiller, however, will miss the blue line. **GeX**, on the other hand, will produce correct output:

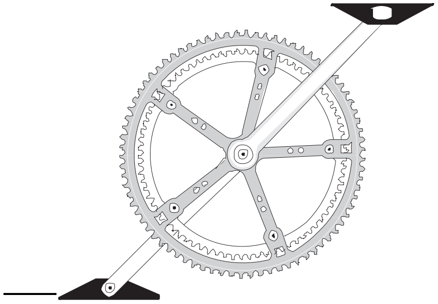


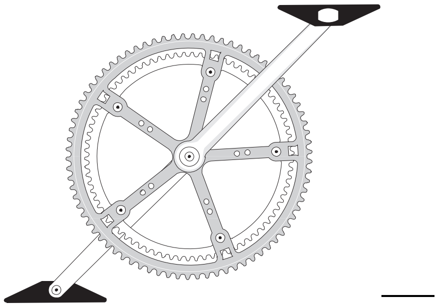
Near-degenerate matrices are not a perverted abberation: they tend to be generated by some common software, especially the CorelDraw. The particular set of numbers in the source above came from a Corel example.

While **GeX** does the work correctly in all cases, some distortion in the line widths is possible and is not avoidable.

6.2 Level 1 strokeadjust

Some graphics programs (Freehand is one) output Level I PS code which fits the coordinates to an integer grid. This code, if executed literally, will produce rather disasterous results with **GeX**(left):





The nature of the problem is a bug (or *feature*) in the Freehand adjustment code which does not bother to check for the device matrix and assumes that it corresponds to the output pixel resolution of 300 dpi or higher (which would imply a device matrix `[4 0 0 4 .. ..]`). However, the **GeX** device matrix is chosen to be an identity, to avoid extra rounding by TEX's ⇔ **GeX**'s coordinate translation. This causes extremely hoarse coordinate rounding.

To avoid this problem, set the `innerscale` option to at least 4. The example above, for example, was produced with

```
\hbox to \textwidth{
  ||\includegraphics[width=5cm]{gears.eps} \hss
  \includegraphics[innerscale,width=5cm]{gears.eps}
  ||%
  }
```

Note: The `innerscale` option is ignored by VTEX in modes other than PDF; it is not also a VTEX-only option.

Notice that the Level II `strokeadjust` operator does not cause a problem since it is totally ignored by **GeX**.

6.3 Font name collision bug

There seems to be a bug in many versions of Acrobat which results in (different) fonts with names starting with `|------...` being treated as a single font.

To avoid this problem, we replace such names with `|xxxxxx....` This, however, would cause a collision if you actually have a font with such name installed.

6.4 Encoding bug

Under Windows, the Acrobat seems to ignore the `/StandardEncoding` specification and uses the `WinAnsiEncoding` instead. This may lead to incorrect character substitution for some codes in the 2nd half of the ASCII set.

To overcome this problem, we always include the encoding vector, even if the font is not reencoded.

6.5 Acrobat4: Missing fonts

Acrobat4 introduced a fresh problem: missing Times and Helvetica fonts.

Adobe apparently was looking for a way out of bugs caused by the absense of Windows-standard Arial and TimesNewRoman in PostScript software; their solution was to add these fonts to the Acrobat4 distribution. Unfortunately, they also removed Times and Helvetica, hoping to emulate them with the Arial and TimesNewRoman variants.

This act introduced problems both for Adobe's software and for VTEX. In the case of Adobe, their claim to PostScript level II (and now Level III) compatibility ceased to be valid: it is easy to come up with examples of Level II code which will not correctly compile by Distiller due to the missing fonts.

In the case of VTEX, similar (or worse) problems would occur.
There are two solutions to the problem:

- Install the Times and Helvetica fonts from, for example, the Acrobat3 distribution.
- Use the new emulation switch, `oe`. With this switch, VTEX will load the Arial and the TimesNewRoman fonts, internally renaming them to Helvetica and TimesRoman. While the results will not be always identical to what standard Helvetica and TimesRoman should produce, it is sufficient for GeX to work.

Chapter 7

Dirty Tricks

7.1 show redefinition

In order to accomodate packages such as PSTricks and PSFrag, VTEX keeps track of redefinition of the `show` PostScript primitive within the **GeX** engine. In addition to supporting the mentioned packages, this allows rather nice font effects to be implemented with very simple inline code.

7.1.1 Simple outline

The below examples were produced with the

```
\def\outl#1{\special{pS: save /show{false charpath stroke}def}
   #1\special{pS: restore}}
```

macro.

This is a test.

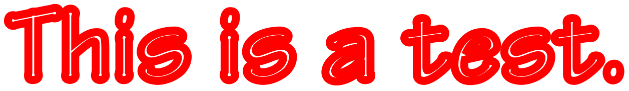
This is a test.

7.1.2 Wider outline with color

The

```
\def\outla#1{\special{pS: save /show{false 3 setlinewidth 1 0 0
  setrgbcolor charpath stroke}def} #1\special{pS: restore}}
```

macro produces

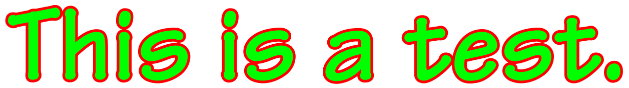


7.1.3 Filled letter with outline

The

```
\def\outlb#1{\special{pS: save /show{false charpath gsave 2 setlinewidth
1 0 0 setrgbcolor stroke grestore 0 1 0 setrgbcolor fill}def}
 #1\special{pS: restore}}
```

macro produces

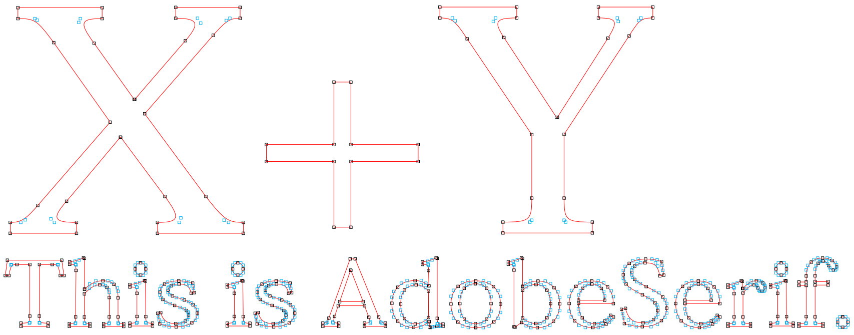


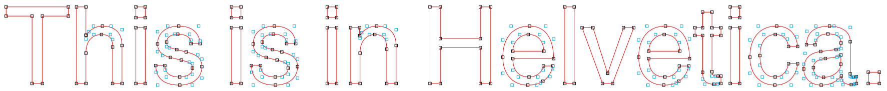
7.1.4 Charpath shown

We can also get inside the character representation (something which PostScript would not do on Type 1 fonts):

```
\def\outlc#1{\special{pS: save /show{false charpath gsave
  0 setlinewidth
  1 0 0 setrgbcolor
  stroke grestore
  0 setlinewidth 0 0 0 setrgbcolor
  {rct}
  {rct}
  {rct 1 0 0 0 setcmykcolor rct rct 0 0 0 setrgbcolor}
  {} pathforall }def}
  #1\special{pS: restore}}
```

to obtain:





Note: It is important to restore the definition of `show` before the end of page. Look carefully at the page number on the previous page to see what might happen if you do not.

7.2 Fragment repositioning

.

Several examples in PSTricks use the PostScript commands to move the text around in order to land it in an appropriate place on a drawing.

VTEX keeps track of PostScript attempts to group the TEX output; when such activity is detected, VTEX generates PostScript code rather than PDF and feeds this code into the **GeX** engine.

7.3 Letterspacing

Here is a feeble attempt in implementing letterspacing. Since Knuth explicitly warned against TEX supporting letterspacing, we accomplish this in **GeX**.

All examples below will letterspace with extra 2-pt space.

Thus,

```
\immediate\special{pS: /k 2 def}%
```

Notice that `\immediate\special` is *not* normal TEX: it causes a backend special to be evaluated right away, rather than waiting for the `output` cycle. Let

```
\def\forcefont{\hbox to 0pt{\phantom{!}}}
\def\as#1{\forcefont\special{pS: k 0 (#1) ashow}}
```

A first attempt is to simply plug-in `ashow`:

```
Letterspacing: \as{Letter-spaced text here.}!
```

to obtain

Letterspacing: Letter-spaced-tex

does not work very well: the exclamation mark overlaps “L”. Indeed, there is no way TEX would know how much space the `\special` would need.

Here is a macro that will measure the required width:

```
\newdimen\tmpz
\def\sw#1{%
\forcefont
\psconsole 255
\immediate\special{pS: (#1) dup stringwidth pop exch length 1 sub k mul add
10 string cvs print}%
\message{Result=[\the\toks255]}
\tmpz=\the\toks255pt
}
```

We call **GeX** in the `\immediate` mode since we want to reuse the result to set the string.

```
\sw{Letter-spaced text here.}
```

Now, it will work correctly:

```
Letterspacing: \hbox to \tmpz{\as{Letter-spaced text here.}\hss}!
```

Letterspacing: Letter-!spaced-tex

Note: Of course, it makes more sense to write a single macro but this example is provided for illustration purposes only.

Note: Even more sense would be to build the output code in the `\immediate` phase and re-use it later (one call to **GeX**, rather than two); such a feature might be added later.

Note: One possible pitfall: **GeX** executes a `save-restore` pair on each emitted page. Thus it is important to assure that no page breaks would occur in the middle of calls to **GeX**.

Chapter 8

Special Cases

8.1 PSTricks

GeX can handle PSTricks files. Notice that in order to deal with some of the more tricky tricks, **GeX** plays some tricks of its own.

Two tricks that appear in PSTricks are show redefinition and fragment repositioning.

8.2 PSFrag

GeX can handle PSFrag files.

The entire PSFrag system is built around the fragment repositioning hack.

We do not yet supply PSFrag as part of VTEX system; it will appear soon.

8.3 MetaPost

While **GeX** can handle MetaPost-generated files, it is important to state that MetaPost outputs invalid EPS files. Rather than use the standard fonts or embed fonts into the EPS, MetaPost merely includes declarations like

```
/cmr10 /cmr10 def
```

and expects post-processing to find and substitute the fonts. Instead of such post-postprocessing, **GeX** ignores (processes, which is the same really) this declaration, but requires either explicit loading of needed fonts via the `.loadfont` extension

```
\special{pS: /cmr10 .loadfont}
```

(one such command for each required font) or enabling of the autoloading feature via the `.autofontload` extension

```
\special{pS: 1 .autofontload}
```

These commands must be issued before a MetaPost-generated file is actually included.