# Creating Labeled "Stand-alone" Figures in LaTeX using `WARMreader` and Adobe Illustrator under Mac OS X

Francesco Costanzo

Department of Engineering Science and Mechanics, The Pennsylvania State University
212 Earth & Engineering Sciences Building, State College, PA 16802-1401, USA
email: `costanzo@engr.psu.edu`
`http://www.esm.psu.edu/faculty/costanzo/`


Gary L. Gray

Department of Engineering Science and Mechanics, The Pennsylvania State University
212 Earth & Engineering Sciences Building, State College, PA 16802-1401, USA
email: `gray@engr.psu.edu`
`http://www.esm.psu.edu/faculty/gray/`
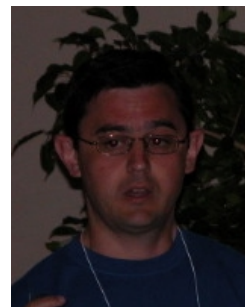
## Abstract

In this paper we discuss our experience as Mac users who lived through the transition that took us from LaTeXing with *Textures*[TM] under Mac OS 9 to LaTeXing with teTeX and TeXShop under Mac OS X. For us, the most difficult yet rewarding aspects of this transition concerned the creation of "stand-alone" figures containing labels and annotations prepared using LaTeX. By "stand-alone" figure we mean a figure in a given format (usually EPS or PDF), which can be imported by one of the many graphics import commands available in LaTeX.

Around April/May 2001, after our switch to Mac OS X, teTeX and TeXShop, because of a number of issues concerning font management under OS X, translation between EPS and PDF formats, as well as issues with TeX fonts in Adobe Illustrator[TM], we could no longer use our old labeling strategy. After some experimentation, we found a solution using the `WARMreader` package developed by Ross Moore and Wendy McKay along with an Illustrator plug-in named `MarkedObjects`, created by Tom Ruark. In this paper we will describe why we are interested in creating stand-alone figures, why we chose to use `WARMreader`, and some of the techniques we have developed to create figures. The paper also describes the use of an AppleScript created by the authors to aid the figure labeling process.

## Introduction

A good number of scientific journals as well as conferences now accept electronic submission of papers. Often, the instructions provided to authors require that the figures included in the paper be provided as files, typically in Encapsulated Postscript (EPS) format, distinct from the file containing the manuscript. Furthermore, depending on the journal, there is no guarantee that the final paper will be typeset using LaTeX. This means that if one's solution to including TeX output in a figure relies on the ability to typeset the manuscript using (LA)TeX, then trouble may be at hand. Hence, if one wishes to annotate a figure using LaTeX fonts or symbols, one must be able to create the figure in question in such a way that the notes and symbols in the figure are not generated along with the rest of the text when the main manuscript file is typeset. In other words, the annotated figures must be created so as to be a self-contained or stand-alone file.

Francesco Costanzo and Gary L. Gray

In this paper, we will describe how we have dealt with the creation of stand-alone figures in the past and the strategies we have developed to accomplish this task after the advent of Mac OS X. In particular, the following two sections will be devoted to presenting a history of our approach to the problem, especially after we started using teTeX and TeXShop under Mac OS X. In the rest of the paper we will present examples of how we currently create annotated figures in the form of a tutorial.

## Stand-alone figures before Mac OS X

Before the release of Max OS X, we used to use *Textures*™ for LaTeXing our papers. As far as the creation of figures is concerned, Adobe Illustrator has been our application of choice for many years.

In order to include LaTeX fonts into an Illustrator figure we were accustomed to simply creating a temporary LaTeX document containing all of the symbols to be included in the figure. After typesetting the temporary file, we made use of a feature in *Textures* allowing us to save the typeset output in the Adobe Illustrator 88 file format, which, for all intents and purposes, is the same thing as saving the output as a PostScript file. Next, we would open the newly created Adobe Illustrator 88 file in Illustrator and simply *copy* the LaTeX symbols from this file into the file containing the figure to be annotated. The precise positioning of the LaTeX symbols within the figure was a trivial matter in that it was accomplished by simply using the mouse to drag the annotation objects to their proper location.

The successful outcome of this very simple procedure relies on the two applications used to carry it out, namely Adobe Illustrator and *Textures*, accessing and correctly/consistently using the same set of fonts. By 'correct/consistent' use of fonts we mean, as we understand it, that the applications in question use the same font map.

## Switching to Mac OS X

Once we switched to Mac OS X, since *Textures* was not available under this new operating system and since we had decided to make a conscious effort not to rely on the Classic environment,[*] we decided to adopt the teTeX distribution as our TeX and LaTeX base, and to use TeXShop as our editing and previewing environment.

This transition felt rather uneventful until we started dealing with the creation of stand-alone figures. Our first instinct was to simply try the same strategy we had always used. Hence, we started by creating a temporary LaTeX document with teTeX and TeXShop, saved the output as a PostScript file, opened it in Adobe Illustrator ... and we discovered that the LaTeX fonts, as seen by Illustrator, had been translated, for the most part, into garbage. In fact, not all of the symbols were misinterpreted. Those that were misinterpreted were primarily, but not exclusively, mathematics and Greek symbols, that is, most of the symbols we use in our figures. At first, we thought that this behavior was caused by the fact that the teTeX fonts are installed in such a way that they are not available for use by other applications. Hence, we proceeded to install, under Mac OS X, the same PostScript fonts used by *Textures* under previous versions of the operating system. These fonts were installed in a location such that they could now be seen by Illustrator. This attempt at fixing the problem did not work, thus leading us to the conclusion that (*i*) the font mapping used by teTeX was different from that seen by Illustrator in the fonts used by *Textures*; and (*ii*) that our old strategy for labeling *any* figure was to be abandoned altogether.[†]

The search for alternative figure labeling methods (which took several weeks of unsuccessful attempts) led us to a package called `WARMreader` developed by Ross Moore and Wendy McKay (McKay and Moore, 1999; Moore, 2001). The capabilities offered by this package will be outlined in the next section.

## The `WARMreader` package: an overview

We begin this section with a disclaimer: we do not intend for this article to be exhaustive in its description of the `WARMreader` package. In particular, we will limit it to the description of those features offered by `WARMreader` that are most relevant to the type of figure labeling we do on a daily basis.

`WARMreader` allows one to overlay any graphics objects imported in (LA)TeX with labels that can be defined within the very (LA)TeX file in which said graphics objects are imported. In this sense, `WARMreader`

---

[*]In Mac OS X, the *Classic environment* refers to the running of Mac OS 9 as a process within Mac OS X so that pre-Mac OS X applications can be used.

[†]We tried several other strategies to solve the font mapping problem but with no success.

can be thought of as conveniently providing and extending the facilities that are made available by packages such as XY-pic (Rose, 1991) and `PSfrag` (Barratt, Grant, and Carlisle, 1996).

Roughly speaking, `WARMreader` overlays given (LA)TEX expressions at specified locations over the imported graphics object. The coordinates of the (LA)TEX labels to be overlaid need to be stored in a text file with the same name as the graphics file but with `.bb` extension. The coordinates in question are expressed in points and measured with respect to the lower left corner of the graphics object's bounding box. Provided that we will come back to a more precise description of a label's placement over a figure, we think that it is important to point out how the successful and convenient use of `WARMreader` relies on the user's ability to generate the labels' coordinates which, in general, could be a rather time consuming task.

For Adobe Illustrator users, the process of generating the labels' coordinates is greatly facilitated by the use of the `MarkedObjects` plug-in, created by Thomas Ruark at Adobe (cf. McKay, Moore, and Ruark, 2001). Although we will describe the use of the plug-in later in various examples, here we simply anticipate the fact that the use of the `MarkedObjects` plug-in allows one to define label position markers within the figure itself along with the definition of the label. Furthermore, the use of the plug-in is such that the required `.bb` file is created automatically, with a complete list of all of the labels' markers and the (LA)TEX annotations to be overlaid onto the figure.

### `WARMreader` and stand-alone figures

By its very nature, the `WARMreader` package is a tool that can only be used from within a (LA)TEX document. This implies that, contrary to what we stated earlier, the labeled figures one generates are not stand-alone objects but are objects embedded in a document. Hence, in order to use `WARMreader` to create stand-alone figures one must devise a strategy to extract the figures out the document containing them and endow each figure file with the necessary bounding box information. The strategy we have adopted is as follows:

1. we create a LATEX document consisting of a single page with `\thispagestyle{empty}`;
2. we then import the graphics object, to be annotated using `WARMreader`, into this document;
3. after adding the labels, we typeset the document using TEXShop with its settings as shown in Fig. 1, which displays the Engine tab within the TEXShop preferences. This step yields several files, two of
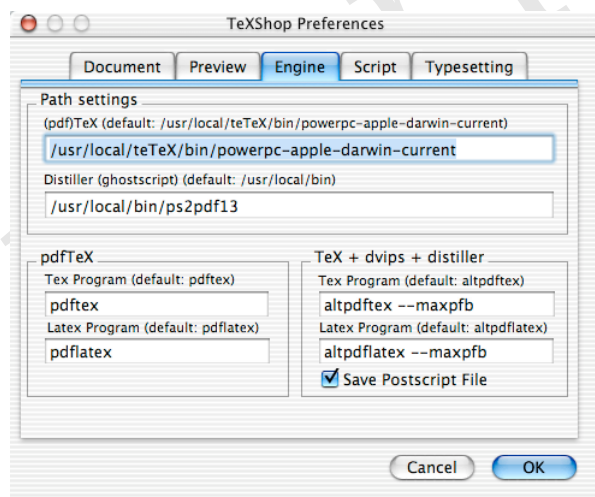


**Figure 1**: TEXShop Engine tab window.

which are of PostScript and PDF type, each consisting of a single page with the annotated figure.

4. At this point we use the `ps2epsi` command made available by Ghostscript to turn the PostScript file created by `dvips` into an EPSI file (EPS file with a bitmap preview);
5. finally, we distill the EPSI file in question using the `epstopdf` command in Ghostscript.

At the end of this operation we have two files:

(a) an EPS file with the annotated figure and the proper bounding box information; and

(b) a PDF file with, again, just the annotated figure with the proper bounding box information.

These files can be treated as graphic objects that can be imported in any other application supporting the import operation of images in EPS and/or PDF formats.

### Remarks and a script

The procedure described in the previous section is not logically complex but it is involved. In particular, what makes it involved is the combination of having to carefully position the labels on the figure and, when everything is in place, having to go through several command-line instructions in the UNIX terminal under OS X to get the final result.

To facilitate the use of this procedure, both by ourselves and by our graduate students, we have created a drag-and-drop AppleScript which makes the procedure essentially automatic. The only non-automatic part of the procedure remains the fine-level adjustment of the labels on the figure or the *nudging*, as it is referred to by Wendy McKay and Ross Moore (McKay and Moore, 1999).

Hence, to illustrate what we actually do in practice we now present an example. This example can be thought of as a tutorial for the use of the Illustrator `MarkedObjects` plug-in, `WARMreader`, and the AppleScript we have created to help make the overall process easy to perform. The script in question has been dubbed 𝕎A𝕉𝕄Fɪɢ*ToPDF* (Costanzo and Gray, 2002).

Before presenting any examples, we feel that it is important to clearly identify the basic tools with which the examples have been created. Here is a detailed list of the operating environment that we currently use and under which we know this procedure works:

1. Operating system: Mac OS X 10.2.4;
2. TeXShop: version 1.28, dated January 29, 2003 (the most current information on TeXShop can be obtained from `http://www.uoregon.edu/~koch/texshop/texshop.html`);
3. teTeX: our current version of teTeX should be more properly referred to as teTeX + gwTeX[*] and is dated February 10, 2003 (available from `http://www.rna.nl/ii.html`);
4. Ghostscript: version 7.05 (available from `http://www.rna.nl/ii.html`);
5. `WARMreader`: version 1.2, dated July 5, 2001 (downloaded from the official `WARMreader` website at `http://www-texdev.mpce.mq.edu.au/WARM/WARMhome/`);
6. Adobe Illustrator: version 10.0.3;
7. `MarkedObjects` Illustrator plug-in: release date of May 6, 2002 (available at `http://www.cds.caltech.edu/~wgm/WARM/adobe/`);
8. 𝕎A𝕉𝕄Fɪɢ*ToPDF*: version 1.0 (can be downloaded at `http://lpcm.esm.psu.edu/~gray/wftpdf.sit`).

### Example: Labeling the vertices and the center of a rectangle

This is a simple example in which we start by running Adobe Illustrator to create a simple rectangle. Once the rectangle is created, the file should be saved as an Illustrator EPS (EPS) file, as shown in Fig. 2. For future reference, `rectangleFig.eps` is the name we have given to the Illustrator EPS file used in this example. Figure 3, shows the content of the Illustrator window, namely a gradient filled rectangle. In addition, the figure displays the location of the Marked Objects Tool, which the `MarkedObjects` plug-in places among the Pen Tools.

Once the figure is created, the labeling process is accomplished by selecting the Marked Objects Tool and creating (by clicking) as many labels as one wishes to create. In particular, we would like to create five Marked Objects, one for each vertex and one for the center of the rectangle. We begin by placing them in arbitrary locations, as shown in Fig. 4. Each Marked Object is numbered sequentially and consists of a ×-symbol with the object's number appearing as a subscript. The remaining part of each Marked Object is a string that can be edited using the Text Tool. The default appearance and text of these objects can be modified by the user via the editing of the `MarkedObjects` preferences. This can be done after opening the `MarkedObjects` dialog window. This window can be opened via: Window → SDK Dialogs → Show Marked Objects Dialog. This results in the appearance of the dialog window shown in Fig. 5. As it can be seen in this figure, the `MarkedObjects` dialog window displays a variety of pieces of information, including the figure's bounding box, as well as each label's text and coordinates. The `MarkedObjects` preferences can be

---

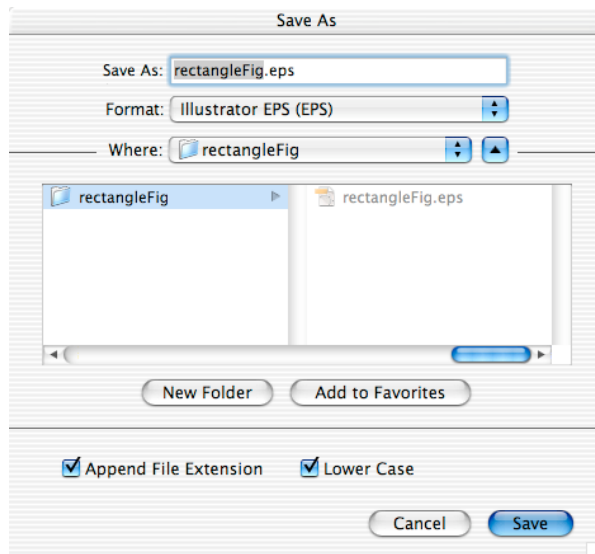[*]Where we understand that 'te' stands for Thomas Esser and 'gw' stands for Gerben Wierda.

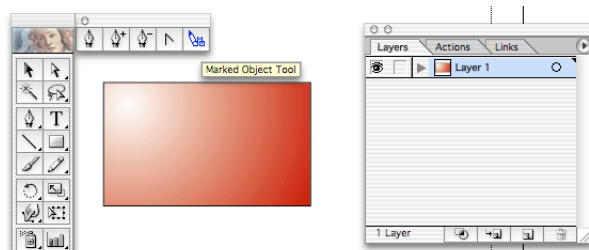**Figure 2**: Illustrator 'Save as' dialog window.



**Figure 3**: `MarkedObjects` Illustrator plug-in tool.

edited by clicking on the triangle-labeled radio-button placed in the upper right corner of the dialog, as is being shown in Fig. 5.

Going back to the description of what happens once the Marked Objects Tool is used and with reference to Fig. 4, it should be observed that using this tool *automatically* results in the creation of a new layer, called Marked Objects DO NOT MODIFY. Furthermore, once the layer is created, any Save operation performed by the user results in the creation (and subsequent updating) of a text file that is placed in the same location as the Adobe Illustrator file containing the figure. The new file in question is automatically given the same name as the Illustrator EPS file, except for the extension, which is automatically set to `.bb`. As far as the labels are concerned, their creation and editing can be done in any order desired by the user. In this example, the labels were first created (by clicking five times at arbitrary locations with the Marked Objects Tool) and then edited using the Text Tool. Figure 6 displays the Marked Objects after their text has been edited.

Now that the labels have been created, we can proceed to placing them at desired locations. In Illustrator, Marked Objects can be moved just like any other graphics object. When placing labels at desired locations, is it useful to keep in mind that the coordinates that the `MarkedObjects` plug-in assigns to a label are the coordinates of the center of the ×-symbol, measured (in points) with respect to the lower left corner of the bounding box of the figure. As far as the figure's bounding box determination is concerned, by monitoring the information provided by the `MarkedObjects` dialog window, it is easy to see that this calculation disregards the position of the Marked Objects. Continuing with the labeling process, in Fig. 7 we can see the labels in their final position. The labeling process is now completed by hiding the Marked
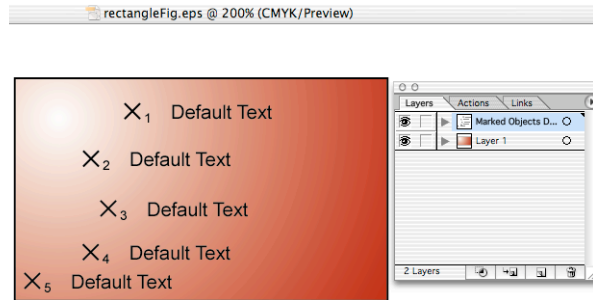
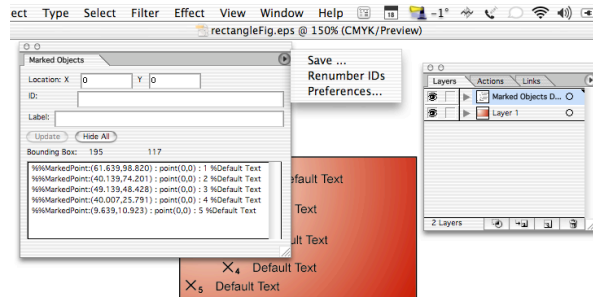**Figure 4**: `MarkedObjects` objects and layer.



**Figure 5**: `MarkedObjects` dialog window.

Objects Layer (by clicking on the "eye" in the Layers palette), saving the resulting Illustrator file, and closing the file in question.

At this point, we have two files in our working folder: the Illustrator EPS file and its companion `.bb` file. Next, we simply drag and drop the Illustrator EPS file onto the AppleScript WARMFIGTOPDF. The outcomes of this operation are

1. the renaming of the Illustrator EPS file and the text `.bb` file via the prepending of the `.eps` and `.bb`, respectively, extensions with the character string '`-AI`';[*] and

2. the creation of a LATEX file carrying the original name of the Illustrator EPS file (with extension `.tex`).

Before proceeding any further, a remark of practical importance must be made. When assigning a name to an Illustrator file, one needs to keep in mind that, at lease with the current version of Adobe Illustrator under Mac OS X, long file names *are not* supported. The importance of this observation lies in the fact that the current version of WARMFIGTOPDF does not check whether or not the addition of the string '`-AI`'

---

[*]If the original Illustrator file name ends in `-AI`, the renaming does not take place and the newly created `.tex` file will have the same name of the Illustrator file without the `-AI` ending.
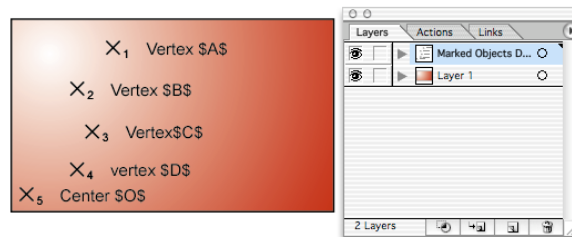


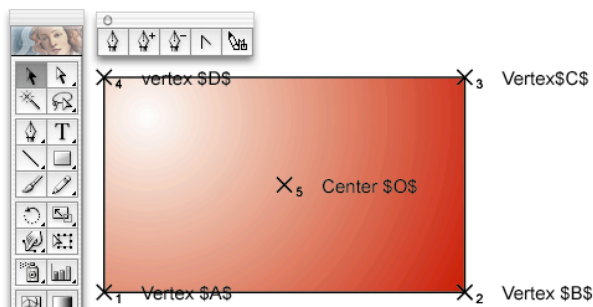**Figure 6**: `MarkedObjects` labels.

**Figure 7**: Marked Objects in their final position.

is compatible with the Illustrator file name length requirements. As the reader can imagine, we have lost more than one long-named Illustrator file by running WARMFIGTOPDF while the file in question was still open in Illustrator.

In the interest of completeness, we now report the content of the `.tex` file WARMFIGTOPDF creates.

```
%&latex
\documentclass[10pt]{article}

%%%%%%%%%%%
% PACKAGES %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%\usepackage[expert]{lucidabr}        %
\usepackage{amsmath}                 %
\usepackage{amssymb}                 %
\usepackage{amsthm}                  %
\usepackage{exscale}                 %
\usepackage[mathscr]{eucal}          %
\usepackage{ifthen}                  %
\usepackage[pdftex]{graphicx}        %
\usepackage[dvipsnames]{color}       %
\DeclareGraphicsExtensions{.pdf, .jpg} %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Settings for FC, GLG, MEP books.
%\input{../../../../../Settings/commands}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% EXOTIC PACKAGES: Figure Labeling within LaTeX %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\usepackage[all,color,frame,import]{xy}  %
\usepackage{warmread}                %
\let\xyWARMprocess\xyWARMprocessMo    %
\let\WARMprocessEPS\WARMprocessMoEPS  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% USEFUL WARMreader COMMANDS %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%\xyMarkedPos{##}*!D( 0.00)!L( 0.00)%        %
%\txt{\rotatebox{90}{yAxis-Title}}            %
%\xyMarkedPos{##}*!D( 0.00)!L( 0.00)%        %
```

```
%\txt{\includegraphics[scale=x.x]{InsetGraph}} %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\begin{document}
\thispagestyle{empty}
%
\WARMprocessEPS{rectangleFig-AI}{eps}{bb}
%
\begin{xy}
\xyMarkedImport{}

\small

% Marked Point Number: 1
% MarkedPoint:(0.500,0.500) : point(0,0) : 1 %Vertex $A$
\xyMarkedTextPoints!D( 0.00)!L( 0.00){1}

% Marked Point Number: 2
% MarkedPoint:(194.314,0.500) : point(0,0) : 2 %Vertex $B$
\xyMarkedTextPoints!D( 0.00)!L( 0.00){2}

% Marked Point Number: 3
% MarkedPoint:(194.314,115.964) : point(0,0) : 3 %Vertex$C$
\xyMarkedTextPoints!D( 0.00)!L( 0.00){3}

% Marked Point Number: 4
% MarkedPoint:(0.500,115.964) : point(0,0) : 4 %vertex $D$
\xyMarkedTextPoints!D( 0.00)!L( 0.00){4}

% Marked Point Number: 5
% MarkedPoint:(97.407,58.232) : point(0,0) : 5 %Center $O$
\xyMarkedTextPoints!D( 0.00)!L( 0.00){5}

\end{xy}
%
\end{document}
```

This `.tex` file is obtained by the use of a simple template which:

(i) includes the `WARMreader` package (along with all the other packages one wishes to include by default[*]);

(ii) properly sets up the `xy` environment, which will overlay the graphic image with the LATEX generated labels;

(iii) includes the graphics file containing the image to be labeled;

(iv) includes every Marked Object created in Illustrator, preceded by a summary of the information it carries by default, i.e., stored in the `.bb` file.

---

[*]The AppleScript source, found in the AppleScript Studio project, needs to be modified in order to change what is included by default.

By default, we have chosen to include the various **Marked Objects** by invoking the `WARMreader` command `\xyMarkedTextPoints`. For those users with an understanding of X̱Ɏ-pic and `WARMreader`, it is clear that this is simply a personal choice. Furthermore, it should be noted that every **Marked Object** is also accompanied by the syntactical elements `D` and `L`, which allow one to *nudge* the object's position in the vertical and horizontal directions, respectively. By default the the nudging amount is set to zero.

The LaTeX file thus created is ready to be typeset and the outcome of that typesetting will be, among other things, two files, one PostScript and one PDF file. At this point, each of these files provides a *page* containing the annotated figure. Figure 8 displays the content of the working folder after typesetting the file `rectangleFig.tex`.
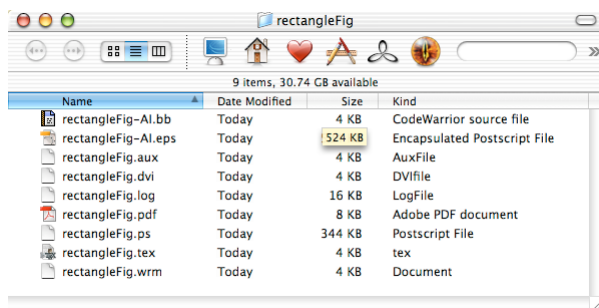


**Figure 8**: Content of the working folder after typesetting the `.tex` file created by WARMFigToPDF.

Files `rectangleFig.ps` and its PDF counterpart are almost the final desired product. The only feature they lack is a bounding box that properly encapsulates the figure. As mentioned earlier, turning the file `rectangleFig.ps` into an equivalent EPS file is accomplished by operating on this file via the **Ghostscript** command `ps2epsi`. To be precise, this operation yields a file of EPSI type. As it turns out, and as will be discussed in greater detail in the next to the last section of the paper, the bounding box computed by `ps2epsi` is not entirely correct in that is often causes elements of the figure to be slightly clipped. Hence, to avoid any unwanted clipping, after creating the `rectangleFig.epsi` file, we edit its bounding box so as to enlarge it by one point in each direction. The EPSI file with the modified bounding box is given the name `rectangleFig.eps`. Finally, this file is turned into a corresponding PDF file (i.e., named `rectangleFig.pdf`) by invoking the **Ghostscript** command `epstopdf`.

The AppleScript application WARMFigToPDF makes all of the operations just described automatic. Specifically, once the file `rectangleFig.tex` is created and typeset (i.e., once the working folder content is that in Fig. 8), one only needs to drop the file `rectangleFig.ps` onto WARMFigToPDF. The outcome of this operation results in the creation of the desired EPS and PDF files, along with the deletion of all the files that are not needed for preserving the capability of future corrections to the figure. Hence, in the end, the content of the working folder is that depicted in Fig. 9. The final labeled figure is shown in Fig. 10. Clearly,
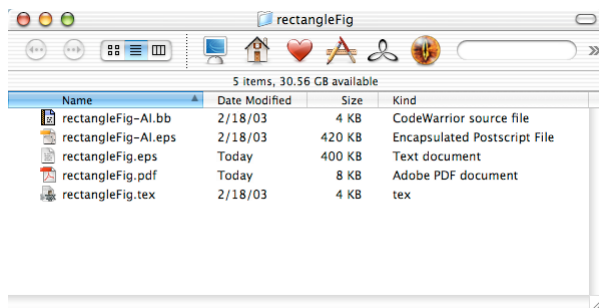


**Figure 9**: Content of the working folder after WARMFigToPDF has created the final EPS and PDF files.

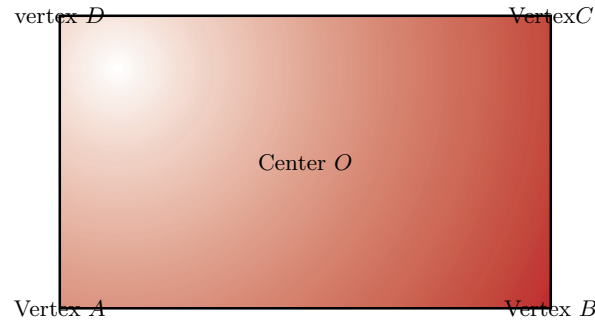the figure needs some adjusting. Hence, at this point, one can go back into the working folder and edit the

**Figure 10**: Appearance of the final PDF stand-alone figure.

`rectangleFig.tex` file to make the due corrections and adjustments. For example, the editing the content of the `xy` environment as listed below makes the figure appear as depicted in Fig. 11.

```
\begin{xy}
\xyMarkedImport{}
\small

% Marked Point Number: 1
% MarkedPoint:(0.500,0.500) : point(0,0) : 1 %Vertex $A$
\xyMarkedTextPoints!D(-1.50)!L( 0.00){1}

% Marked Point Number: 2
% MarkedPoint:(194.314,0.500) : point(0,0) : 2 %Vertex $B$
\xyMarkedTextPoints!D(-1.50)!L( 0.00){2}

% Marked Point Number: 3
% MarkedPoint:(194.314,115.964) : point(0,0) : 3 %Vertex$C$
%\xyMarkedTextPoints!D( 0.00)!L( 0.00){3}
\xyMarkedPos{3}*!D( 1.50)!L( 0.00)\txt{Vertex $C$}

% Marked Point Number: 4
% MarkedPoint:(0.500,115.964) : point(0,0) : 4 %vertex $D$
```

```
%\xyMarkedTextPoints!D( 0.00)!L( 0.00){4}
\xyMarkedPos{4}*!D( 1.50)!L( 0.00)\txt{Vertex $D$}

% Marked Point Number: 5
% MarkedPoint:(97.407,58.232) : point(0,0) : 5 %Center $O$
%\xyMarkedTextPoints!D( 0.00)!L( 0.00){5}
\xyMarkedPos{5}*%
!D( 0.00)!L( 0.00)%
\txt{\rotatebox{45}{\textcolor{white}{Center $O$}}}

% Marked Point Number: 5
% MarkedPoint:(97.407,58.232) : point(0,0) : 5 %Center $O$
%\xyMarkedTextPoints!D( 0.00)!L( 0.00){5}
\xyMarkedPos{5}*%
!D( 0.00)!L( 0.00)%
\txt{\rotatebox{-45}{\textcolor{blue}{$O$ Center}}}

\end{xy}
```
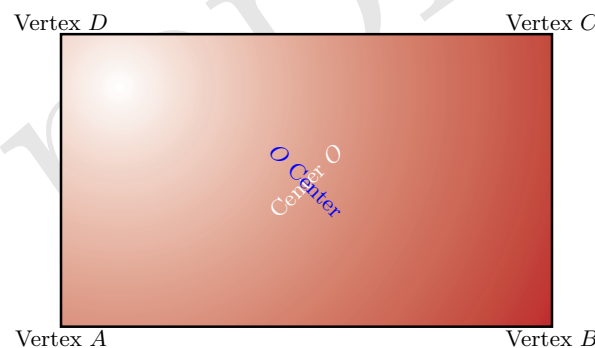


**Figure 11**: Modified appearance of the stand-alone `rectangleFig`.

Figure 11, along with the LaTeX source code used to generate it, is meant to illustrate the following points:

1. as discussed in greater detail later, the default position of the labels can be adjusted by taking advantage of the positioning directives `D` and `L`;

2. one can use the command `\xyMarkedPos`, possibly the most flexible command `WARMreader` provides, to make a label out of virtually any displayable LaTeX construct;

3. with reference to the labels originally defined as 'Vertex$C$' and 'vertex $D$' and then subsequently corrected to 'Vertex $C$' and 'Vertex $D$', one can use the `\xyMarkedPos` command to correct the content of

the labels directly in the .tex file that includes the WARMreader package, instead of, for example, going back to the original Illustrator file;

4. regardless of the command used to include a label, the same label can be included multiple times with different variations.

### On *nudging*

Although this topic has been already discussed elsewhere (see, e.g., McKay and Moore, 1999; Moore, 2001), for the sake of completeness we will now touch upon the subject of how to accurately position labels.

The accurate placement of a label relies on understanding the exact meaning of the label's coordinates. To this end, let us be reminded that a label, being a (LA)TEX object, can be thought of as a *box*. Next, with reference to Fig. 12, let $w$ and $h$ be the width and height of the box bounding the label, respectively.
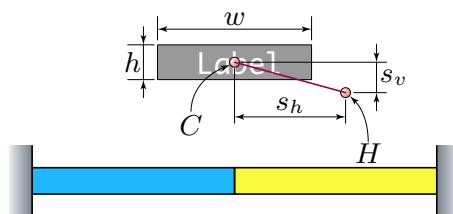


**Figure 12**: Elements defining the geometry of a label.

Furthermore, let $C$ denote the center of the label's box and let $H$ denote the point we will refer to as the label's *handle*. The quantities $s_h$ and $s_v$ will be referred to as the horizontal and vertical shifts, respectively. Now that these geometric descriptors have been introduced, we are ready to define the meaning of the expression "the label's coordinates", as provided by the .bb file — they are the coordinates of the point $H$, the label's *handle*, with respect to the lower left corner of the figure's bounding box. As such, the label's handle is always to be considered a *fixed point*. The quantities $s_h$ and $s_v$ are to be understood as the position coordinates of $H$ with respect to $C$. By default, $s_h = s_v = 0$, that is, the center of the box is made to coincide with the label's handle. WARMreader commands are such that one *cannot* directly specify the values of $s_h$ and $s_v$ is some chosen unit. In order to cause $s_h$ and $s_v$ to have non-zero values one actually specifies the value taken on by the ratios $2s_h/w$ and $2s_v/h$, respectively.

As an example, consider the position directives used to specify the position of the label 'Vertex $A$' shown in Fig. 11:

```
% Marked Point Number: 1
% MarkedPoint:(0.500,0.500) : point(0,0) : 1 %Vertex $A$
\xyMarkedTextPoints!D(-1.50)!L( 0.00){1}
```

The directives in question are !D(-1.50) and !L( 0.00). The first directive, namely !D(-1.50), indicates that the position of $H$ with respect to $C$ must be 1.5 times the half-height of the box bounding the expression 'Vertex $A$.' Furthermore, the vertical position of $H$ relative to that of $C$ is *downward* (!D) and *negative* (-1.50). Given that the coordinates of the label's handle are those of the lower left corner of the rectangle, as shown in Fig. 7,* these instructions result in a position of the label's box center 1.5 times the box's half-height *below* the box's handle while leaving the $s_h = 0$ (since !L( 0.00)).

As additional examples, here below we provide the directives to make one of the corners of the label's bounding box coincide with label's handle:

1. !D( 1.00)!L( 1.00): the lower left corner of the label's box is made to coincide with '$H$;'
2. !D( 1.00)!L(-1.00): the lower right corner of the label's box is made to coincide with '$H$;'
3. !D(-1.00)!L(-1.00): the upper right corner of the label's box is made to coincide with '$H$;'
4. !D(-1.00)!L( 1.00): the upper left corner of the label's box is made to coincide with '$H$.'

### Discoveries and Observations

While trying to understand the failure of our pre-Mac OS X labeling strategy and while coming up with an alternative strategy, we have run into a few interesting quirks that we would like to share.

---

*Recall that the label's coordinates are the coordinates of the center of the ×-symbol.

**Illustrator, PDF format, and bounding boxes** The labeling procedure we have outlined starts with the creation of a figure. As discussed earlier, when using Adobe Illustrator, this figure must be saved an Illustrator EPS (EPS) file. Why not save the figure directly in PDF format? In other words, why not go through the labeled stand-alone figure creation process in a full PDF "environment"? The answer to this question is manifold. First of all, most of the journals to which we submit our papers prefer to receive graphics in EPS format. In other words, it is useful to have the same figure in both EPS and PDF formats. Second, while it is possible to save an Adobe Illustrator file in PDF format, with the current version of Illustrator the resulting figure is assigned a bounding box which, in reality, is as large as the paper media specified in the Page Setup... dialog. This means that when importing the PDF figure in the corresponding `.tex` document for the purpose of labeling, one is faced with the problem of determining the true bounding box of this figure. Finally, once the labeling process is carried out, the resulting PDF file has, again, a bounding box equal to that of the page. Thus, again, one would have to find a way to determine the true bounding box of the *labeled figure*, which. in general, is not the same as that of the figure in the corresponding Illustrator file. This last problem would need to be solved even if future versions of Adobe Illustrator were to provide the possibility of saving a PDF file with a bounding box limited to that of the figure (as opposed to that of the page). However, we are not aware of any facilities (other than the Crop Pages ... facility offered by Adobe Acrobat$^{TM}$) that allows one to determine and edit bounding boxes of PDF files. Hence, at least for now, the only way for us to create a figure in PDF format with a proper bounding box is to create an EPS figure first and then distill it via Ghostscript.

**To clip or not to clip** While developing our labeling strategy, the calculation of the bounding box of the labeled figure was performed using the following Ghostscript command: `gs -q -dNOPAUSE -dBATCH -sDEVICE=bbox`. The argument of the command is the PostScript file generated by typesetting the `.tex` file which contains the `WARMreader` commands. As strange as it may sound, in one of the Ghostscript distributions we used, the `bbox` device was no longer available. Not certain as to whether or not this device was going to be made available again, we decided to rely on the bounding box information contained in the EPSI file generated by running the `ps2epsi` command. What we discovered in doing so, is that the bounding box computed by `ps2epsi` is often different from that computed by Ghostscript (via use of the `bbox` device). As a matter of fact, Ghostscript computes both the bounding box and the high resolution bounding box while `ps2epsi` only computes the bounding box of the content of a PostScript file. The other behavior we observed was that whether using the bounding box information computed by `ps2epsi` or that computed by Ghostscript, the figures we were extracting from the PostScript files generated by our procedure were often clipped around the edges. In other words, somehow the bounding box as delivered by `ps2epsi` (or, although less often, by Ghostscript) was ever so slightly too small. The problem was solved pragmatically by enlarging the `ps2epsi` generated bounding box by one point in all directions. However, we never found the time to identify the source of the `ps2epsi` and Ghostscript errors.

### Summary of the Figure Labeling Procedure

For convenience, we now summarize the steps in using WARMFIGTOPDF to label figures:

1. Use Adobe Illustrator to create the figure you wish to label. Save the file, keeping in mind that:
   (a) Illustrator 10 under Mac OS X does not yet support long file names, and
   (b) WARMFIGTOPDF will insert `-AI` into your filename.
2. Using the Marked Objects Tool in Illustrator, place the labels in the desired positions and change the text of those labels to the desired content using the Text Tool. When you are done, hide the layer containing the Marked Objects and then save and close the file.
3. Drag and drop the Illustrator file you have just created and marked onto WARMFIGTOPDF to create a `.tex` file with the proper `WARMreader` commands. This file is ready to be typeset.
4. Open the resulting `.tex` file and typeset it using the 'TEX + Ghostscript' setting in TEXShop.
5. Iteratively adjust the positions of the Marked Objects by editing the `.tex` source and typesetting.
6. Once you are happy with the position of the Marked Objects, close the `.tex` file and drop the PostScript file that results from typesetting onto WARMFIGTOPDF. This will delete all unnecessary files and will create an EPS and PDF file of your marked-up figure, each with the proper bounding box.

Francesco Costanzo and Gary L. Gray

**The Future of** W̶a̶R̶M̶**FigToPDF**

We use gradients rather extensively in our Illustrator work to generate the appearance of depth. We recently discovered that there can be problems with gradients in Illustrator EPS files that have been converted to PDF using Ghostscript. Figure 13 shows a simple rectangle that has been filled with blue and has had a



**Figure 13**: Elements defining the geometry of a label.

simple radial gradient applied. The radial gradient starts and ends *within* the rectangle. The PDF file shown in Figure 13 was saved out of Illustrator 10 as an Adobe PDF file and then cropped using Acrobat. On the other hand, if instead we save the file as an EPS file and then use Ghostscript to convert it to PDF, we obtain the result shown in Fig. 14. Notice the incorrect color to the left and right of where the radial gradient



**Figure 14**: Elements defining the geometry of a label.

started and finished using the Gradient Tool. This appears to be a problem with Ghostscript and is an issue that we need corrected for the types of figures we create.

Clearly, if an image is saved out of Illustrator as a PDF file, there is no problem with the gradient. Therefore we can "work around" this problem by working with PDF files rather than with EPS files. In addition, since the future of LaTeX seems to be heading in the direction of PDF rather than PostScript, it is our feeling that the future of WaRMFigToPDF should also be in the direction of PDF. With this in mind, we have undertaken the revision of WaRMFigToPDF with the goal of avoiding Ghostscript to create the PDF images we wish to include in our work. In addition, since some publishers still require the submission of images for papers in EPS format, WaRMFigToPDF will still automatically create the appropriately marked up EPS file as part of the process. The general procedure used by the new version of WaRMFigToPDF will be as follows:

1. The user will create an image in Illustrator, mark it up, and save it as both an Illustrator PDF file and an Illustrator EPS file.

2. WaRMFigToPDF will create the `.tex` file with the marked objects embedded and ready for typesetting.

3. The user will then typset the `.tex` file and adjust the positions of the marked objects in the usual way. In this step, the `.tex` file is typsetting using TeX + Ghostscript and is reading in the EPS file. The outcome of this step is PostScript file as well as a PDF file that was created using Ghostscript.

4. The user will then drop the PostScript file on WaRMFigToPDF and then WaRMFigToPDF will use Ghostscript to determine the bounding box of the marked up PostScript file. This bounding box information is then used to create a corresponding final EPS file as well as a final PDF file via a typesetting process that the user never sees. The PDF is created by typesetting using `pdflatex`, reading in the original Illustrator PDF and setting the `viewport` by parsing the Illustrator PDF for the appropriate bounding box information. The bounding box used for cropping the resulting PDF file is that which was previously obtained by running the PostScript file through Ghostscript. This bounding box is incor-

porated into the PDF via the `\pdfpageattr` command, which has been included in the `.tex` file (which the user never sees).

We should also mention that the new version of WARMFIGToPDF will still process EPS files as described in this paper, but it will also have the new capability outlined above.

This new version of WARMFIGToPDF should be available by the time you read this.

## References

Barratt, C., M. C. Grant, and D. Carlisle. "PSfrag". 1996. Available at http://www.ctan.org/tex-archive/macros/latex/contrib/supported/psfrag/.

Costanzo, F. and G. L. Gray. "WARMFIGToPDF". 2002. Available at http://lpcm.esm.psu.edu/~gray/wftpdf.sit.

McKay, W. and R. Moore. "Convenient Labelling of Graphics, the `WARMreader` Way". *TUGboat* **20**(3), 262271, 1999. Available as a preprint at http://www-texdev.mpce.mq.edu.au/WARM/WARMhome/.

McKay, W., R. Moore, and T. Ruark. "Adobe `MarkedObjects` plugin for `WARMreader`". 2001. Preprint available at http://www.tug.org/tug2001/bulletin/preprints/rossIII.html.

Moore, R. "What is `WARMreader`?". 2001. Available online at http://www-texdev.mpce.mq.edu.au/WARM/ and http://cds.caltech.edu/~wgm/WARM/reader2001.html.

Rose, K. H. "XY-pic". 1991. Available at http://www.ctan.org/tex-archive/systems/mac/textures/graphics/.