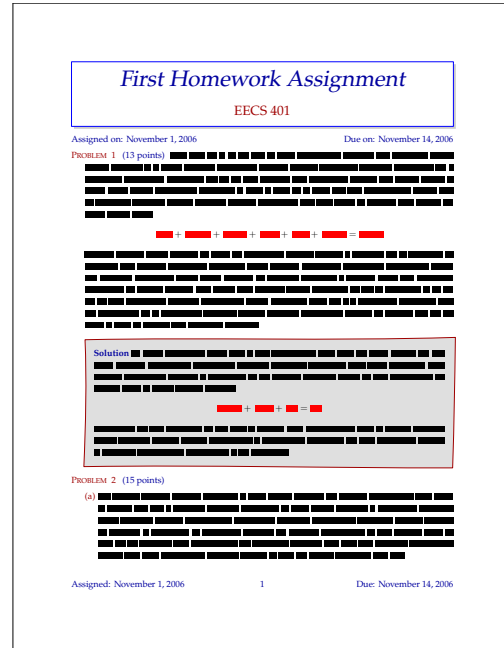


Problem set



Solutions

Figure 1: Sample assignment and solution

course a product; and each homework assignment a component. A PDF file for a particular assignment can be generated by compiling a component; a PDF file for all assignments (which is useful for students who want to archive the course material) can be generated by compiling the product file. While compiling a component, CONTEXT searches the parent directory for project, product and environment files. This feature allowed me to use a directory structure shown in Figure 2 (the course was called EECS 401). This structure ensures that all the files related to a particular course are in one directory. In the future, if I have to create assignments for a different course, I can create a new directory parallel to EECS401 and define it as a new product. This structure is also portable as I can copy the assignments directory to a different computer and everything will work there.

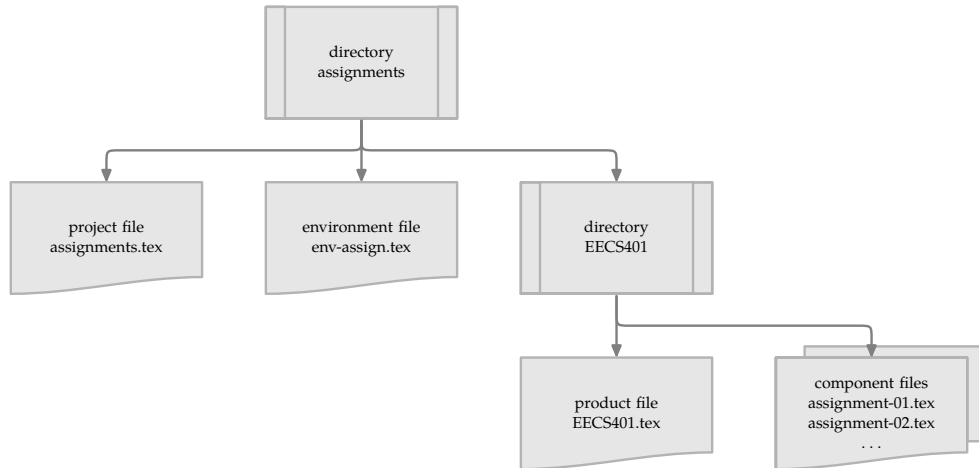


Figure 2: The project structure

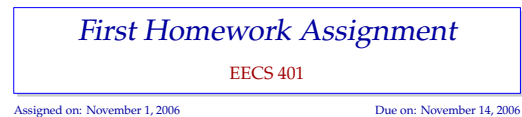
### 3 The Work Flow

An assignment has a title —containing assignment name, course name, assigned date and due date— and a set of problems with solutions. I defined three main macros: `\assignment`, `\startproblem` and `\startsolution`. The `\assignment` macro collects the information needed for the title of the assignment, and prints the title in a fancy layout. This macro looks as follows, with the typeset title shown on the right.

```

\assignment[
  title=First Homework Assignment,
  course=EECS 401,
  assigned={d=1,m=11,y=2006},
  due={d=14,m=11,y=2006}]

```



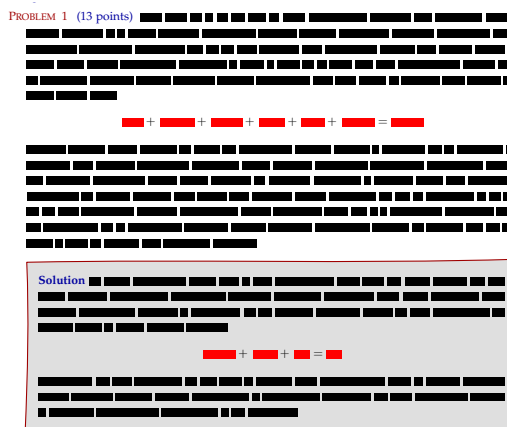
The environments `\startproblem` and `\startsolution` are for creating problems and solutions. The solution is only typeset in solution mode, which can be set by

passing `--mode=solution` to `texexec`, or by saying `\enablemode[solution]` at the beginning of the file. I prefer the solution environment to be nested inside the problem environment as shown below.

```

\startproblem[reference tag]{points}
  problem statement
  .....
  .....
  \startsolution
  solution
  .....
  .....
  \stopsolution
\stopproblem

```



The title information, problems and solutions for a single assignment are entered in a component file (`assignment-01.tex`). One also needs to do some initializations in a component file to specify the project and the product to which that component belongs. Thus a component file looks like

```

\startcomponent assignment-01
\project      assignments
\product      EECS401

\assignment[
  title=First Homework Assignment,
  course=EECS 401,
  assigned={d=1,m=11,y=2006},
  due={d=14,m=11,y=2006}]

\startproblem[reference tag]{points}
  problem statement
  \startsolution

```

```

        solution
    \stopsolution
\stopproblem
% Other problems
\stopcomponent

```

The layout and formatting of the problems and the solutions are taken care by the environment file `env-assign.tex`. The details are given in [env-assign.pdf](#).<sup>1</sup> I will elaborate on the main macros `\startproblem` and `\startsolution`.

The macro `\startproblem` needs to have a tag for referencing<sup>2</sup> and the number of points for the problem. Both these parameters should be optional, and it should be possible to distinguish between them when only one argument is present. One way to implement such a macro is using parameter assignment, like `\startproblem[tag=..., points=...]`. This is too verbose, so I came up with an alternative solution —provide the second optional argument in curly brackets! So the macro `\startproblem` can have four forms:

1. `\startproblem[tag]{points}` where `tag` is a tag for referring the problem and `points` denotes the number of points for the problem.
2. `\startproblem[tag]` where only a tag for referring the problem is defined, and no points are declared.
3. `\startproblem{points}` where only the number of points are declared, and no tag for referring is defined.
4. `\startproblem` where there is neither tag for referring, nor points are defined.

Notice that I want two optional arguments here, one inside square brackets, and the other inside curly brackets. I define this macro as follows: First define an enumeration<sup>3</sup> `PROBLEM` that takes care of numbering and referring of problems.

---

1. This file was typeset from the source file using `texexec --module --mode=color env-assign.tex`. This gives a nicely typeset PDF of the source code and the comments. Lines starting with `%D` are typeset using `CONTEXT`, and code lines are pretty printed in color (note the `--mode=color` in the command).

2. A tag for referencing is a label in  $\LaTeX$  parlance. In all `CONTEXT` environments, the label is passed as an optional argument to the macro.

3. Enumerations in `CONTEXT` are different from `enumerate` environment in  $\LaTeX$ . One can think of them as being equivalent to `theorem` environment in  $\LaTeX$ .

```

\defineenumeration
[PROBLEM]
[
  text={\labeltext{problem}},
  location=hanging,
  headstyle=\sc,
  headcolor=colorone,
  before={\resetnumber [formula] \page [desirable]},
  after=\blank,
]

```

This does most of the book-keeping, and is hidden from the user. All enumerations accept an optional argument, in square brackets, as a tag for referring. The main macro `\startproblem` is defined as

```

\def\startproblem%
{\dosingleempty\dostartproblem}

\def\dostartproblem[#1]%
{\startPROBLEM[#1]\dosinglegroupempty\dodostartproblem}

\def\dodostartproblem#1%
{\iffirstargument
% Check if #1 = 1, use point, else use points
\bgroup \colortwo (#1 \dopoints{#1}) \egroup
\fi}

```

This definition takes care of the two different optional arguments. To understand what is happening here, let's break it down in pieces.

```

\def\startproblem%
{\dosingleempty\dostartproblem}

```

The macro `\dosingleempty` is `CONTEXT`'s way of doing optional arguments —it checks for an optional argument in square brackets. If you want two or more optional arguments (all in square brackets), you can use `\dodoubleempty`, `\dotripleempty`, etc. The conditionals `\iffirstargument`, `\ifsecondargument`, etc., check which arguments were present. These macros are very robust and form a foundation for the entire `CONTEXT`

macro package. See the wiki<sup>4</sup> for more details. The macro `\dostartproblem` is defined as

```
\def\dostartproblem[#1]%
  {\startPROBLEM[#1]\dosinglegroupempty\dodostartproblem}
```

It passes the optional argument to `\startPROBLEM` which uses it as a tag for referring. `\dosinglegroupempty` is a `CONTEXT` macro that checks for an optional argument in curly brackets. This is the *group* analogue of `\dosingleempty`. It checks for a left brace, so `\startproblem 2` is not the same as `\startproblem{2}`. In the first case `\dosinglegroupempty` does not *see* the optional arguments, while in the second it does. The macro `\dodostartproblem` is defined as

```
\def\dodostartproblem#1%
  {\iffirstargument
  % Check if #1 = 1, use point, else use points
  \bgroup \colortwo (#1 \dopoints{#1}) \egroup
  \fi}
```

The conditional `\iffirstargument` is set to true by `\dosinglegroupempty` when the optional argument is present. If the optional argument is present, it is printed followed by *point* or *points* as appropriate.

The macro `\startsolution` should hide its contents in a normal run, and print its contents in solution mode. I first define `\startsolution` as a local buffer (which collects its contents and does nothing with them).

```
\definebuffer[solution]
\setupbuffer[solution] [local=yes]
```

When solution mode is enabled, I redefine `\startsolution` as an enumeration without a number.<sup>5</sup>

```
\defineenumeration
  [solution]
```

---

4. [http://wiki.contextgarden.net/Inside\\_Context](http://wiki.contextgarden.net/Inside_Context)

5. Even though I do not want numbered solutions, I use (rather abuse) an enumeration since it is easy to change its look.

```
[
    text=\labeltext{solution},
    number=no,
    headstyle=bold,
    headcolor=colortwo,
    location=serried,
    width=fit,
    before=\startsolutionbackground,
    after=\stopsolutionbackground
]
```

A frame which can break across pages is placed around the solutions. The options `before=...` and `after=...` to `\defineenumeration` call the `textbackground` macros, which draw the frame using `METAPOST`. To read more about `textbackgrounds`, read the *details* manual;<sup>6</sup> to read more about integrating `METAPOST` to draw backgrounds in `CONTEXT`, read the *MetaFun* manual.<sup>7</sup>

To generate the homework and its solutions change directory to `EECS401` and

1. run:

```
texmfstart texexec --result=assignment-01-p.pdf assignment-01
```

to generate `assignment-01-p.pdf` that only contains the problems.

2. run (in one line):

```
texmfstart texexec --mode=solution
                    --result=assignment-01-s.pdf assignment-01
```

to generate `assignment-01-s.pdf` that contains problems and their solutions.

Notice the `--result=...` option to `texexec` renames the output PDF files, which makes it easier to maintain two different PDF outputs. `texexec` has other useful features, which are described in the `texexec` manual.<sup>8</sup> The attached zip file `assignments.tar.gz`

---

6. <http://www.pragma-ade.com/general/manuals/details.pdf>

7. <http://www.pragma-ade.com/general/manuals/metafun-p.pdf>

8. <http://www.pragma-ade.com/general/manuals/mtexexec.pdf>. This manual is a bit out of date, `CONTEXT` recently moved to a RUBY `texexec` while the manual describes the options for the old PERL `texexec`. However, most of the options explained in the manual are still valid as the user-interface has not changed much.

contains the whole setup, including a sample assignment `assignment-01.tex`. The typeset problem set and solution of this sample are available as [assignment-01-p.pdf](#) and [assignment-01-s.pdf](#).

## 4 A Note about up to date CONTEX<sub>T</sub>

The example assignments/EECS401/assignment-01.tex provided with this article uses a macro `\sometxt` which has been recently (in July 2006) added to CONTEX<sub>T</sub>. If that does not work in your system, you need to update your CONTEX<sub>T</sub> distribution by running

```
texmfstart cxttools --update
```

In case you cannot update your distribution, you can replace `\sometxt{...}` with `texttext"..."`.<sup>9</sup> `\sometxt` really speeds up METAPOST processing by avoiding separate T<sub>E</sub>X runs to typeset text. CONTEX<sub>T</sub> is developed at an fast pace and I would recommend updating periodically to avail new features.

## 5 Conclusion

The whole experience of getting homeworks assignments the way I wanted them turned out to be much easier than expected. CONTEX<sub>T</sub> is a coherent set of macros, with comprehensive manuals, exhaustive explanations in the source, and a very helpful mailing list.<sup>10</sup> With a little playing around, you can easily get a look *that you want*.

---

9. The regular expression `s/\\sometxt{\(\{-\}\)}/texttext"\1"` can be used in Vim.

10. [http://wiki.contextgarden.net/ConTeXt\\_Mailing\\_Lists](http://wiki.contextgarden.net/ConTeXt_Mailing_Lists)