The xtemplate package: An example

Clemens Niederberger

Abstract

One of the most important points in the development of LaTeX3 is — roughly speaking — the separation of implementation, layout design and user interface. The package xtemplate connects the first two — it is part of the *Designer Interface Foundation Layer*. This article tries to demonstrate the idea behind the package and its usage with a (not necessarily practical) example.

1 Introduction

Not too long ago I had a first look at the xtemplate package which is part of the l3packages bundle.[1] After I understood the idea behind it I was immediately excited. So: what idea am I talking about?

The underlying structure for LaTeX3 has been discussed, for instance, by Frank Mittelbach at the TUG 2011 conference [1]. Of course I'm not going to repeat that.[2] An important part — if not the main idea — is the strict separation of different so-called layers. I'm confident you've already heard about the *Core Programming Layer* — `expl3`.

The xtemplate is part of a different layer, the *Designer Interface Foundation Layer*. So principally it is directed at package and class authors but I believe it will also play a big role in a LaTeX3 kernel, at least conceptually. The idea behind it isn't new, as one can read in "New Interfaces for LaTeX Class Design" (1999) [2].

Roughly speaking, the idea is this: a class has to provide a suitable design for different objects, such as section headings or lists. Preferably this would be done via a simple interface that would allow authors to adjust certain parameters to their own wishes or requirements. In other words (from xtemplate's documentation [4]):

1. semantic elements such as the ideas of sections and lists;
2. a set of design solutions for representing these elements visually;
3. specific variations for these designs that represent the elements in the document.

One should be able to determine the number and possibly the kind of arguments from both the definition and the interface.

xtemplate now allows one to declare objects, and so-called templates for these objects. For every object instances can be defined (figure 1). The user interface is then defined with the help of xparse [3].[3]

I for my part learn best through examples and I'm guessing I am not alone with that. So, I am going to demonstrate the concept and the different commands using an example that is not necessarily a practical application of xtemplate. Inspired by a question on `tex.stackexchange.com` [5], I will declare an object `names` and two templates `fullname` and `initial`. Declaring the instances will then show how flexible and easily extendable the concept is.

In the end, the code

```
\name{Jack Skellington} \par
\name[init-it-rev]{Jack Skellington}
```

will give:

Jack Skellington
Skellington, *J.*

A small warning: if you're not familiar with `expl3` — the "programming language" of LaTeX3 — details of the code might seem cryptic. But I will keep the example short so you should be able to follow the important parts.

2 The important commands

Basically there are four commands that are important for the definition of the structures:

1. `\DeclareObjectType`
   {⟨*object*⟩}
   {⟨*number of args*⟩}
2. `\DeclareTemplateInterface`
   {⟨*object*⟩}
   {⟨*template*⟩}
   {⟨*number of args*⟩}
   {⟨*interface*⟩}
3. `\DeclareTemplateCode`
   {⟨*object*⟩}
   {⟨*template*⟩}
   {⟨*number of args*⟩}
   {⟨*parameter*⟩}
   {⟨*code*⟩}
4. `\DeclareInstance`
   {⟨*object*⟩}
   {⟨*instance*⟩}
   {⟨*template*⟩}
   {⟨*parameter*⟩}

The first command, `\DeclareObjectType`, declares the object and specifies how many arguments it gets.

Then the object can be specified with the second command, `\DeclareTemplateInterface`. More precisely an interface is declared, i.e., the number

[1] From directory `macros/latex/contrib/l3packages`
[2] I wouldn't be qualified anyway.
[3] Also part of the l3packages bundle.

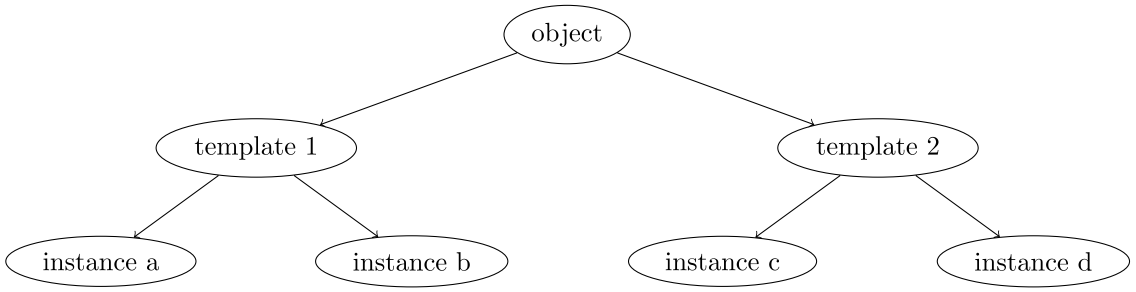

Figure 1: Schematic figure of the relationships between object, templates and instances.

and type of parameters are declared with which the template can be customized later.

The third command, `\DeclareTemplateCode`, is where the actual definitions are made. The parameters defined in the interface get variables assigned, and code is defined that determines the behaviour of the template.

The fourth command, `\DeclareInstance`, at last defines an instance that instantiates a template with concrete values for the parameters. Each of these instances can then be used in the user command with `\UseInstance`.

3 An example

Now let's consider an actual example.

3.1 The object

The first thing to do is to think about the basic interface. The user command of our object `names` is going to be `\name`, with one argument taking the lastname and firstname separated with a space. At a level deeper, though, we want to handle lastname and firstname as two separate arguments. Thus the object is going to get two arguments:

```
\usepackage{xtemplate,xparse}
% we use expl3, so activate the expl3 namespace
\ExplSyntaxOn
% #1: lastname, #2: firstname
\DeclareObjectType { names } { 2 }
```

The next thing to do is to specify the templates.

3.2 The templates

Templates are declared for an existing object. First the interface has to be specified. The number of arguments of the template and a possible list of parameters has to be declared. Every parameter is given a certain type and can get a default value.

```
% the interface for the template ‘fullname’:
\DeclareTemplateInterface{names}{fullname}{2}
{
  reversed           : boolean   = false ,
  use-last-name      : boolean   = true  ,
  use-first-name     : boolean   = true  ,
  last-name-format   : tokenlist         ,
  first-name-format  : tokenlist         ,
}
% the interface for the template ‘initial’:
\DeclareTemplateInterface{names}{initial}{2}
{
  reversed           : boolean   = false ,
  use-last-name      : boolean   = true  ,
  use-first-name     : boolean   = true  ,
  last-name-format   : tokenlist         ,
  first-name-format  : tokenlist         ,
  last-name-initial  : boolean   = false ,
  first-name-initial : boolean   = true  ,
}
```

The parameters that are defined here can easily be extended and are determined by the type of object and of course the desired degree of complexity. Here we have just a few to demonstrate the concept.

After the interfaces for the templates have been declared the actual code can be defined. Let's start with `fullname`. We're going to need suitable variables or functions for the parameters. In the ⟨*code*⟩ part `\AssignTemplateKeys` will define them with actual values and activate them.

Our code now only tests the values of the boolean variables and writes the names in the given order. The solution is not the most elegant one but will do for this demonstration:

```
% variables first:
\bool_new:N \l_names_reversed_bool
\bool_new:N \l_names_use_last_bool
\bool_new:N \l_names_use_first_bool
\tl_new:N   \l_names_last_format_tl
\tl_new:N   \l_names_first_format_tl

% now the template code:
% #1: lastname, #2: firstname
\DeclareTemplateCode {names} {fullname} {2}
{
  reversed          = \l_names_reversed_bool   ,
  use-last-name     = \l_names_use_last_bool   ,
  use-first-name    = \l_names_use_first_bool  ,
  last-name-format  = \l_names_last_format_tl  ,
  first-name-format = \l_names_first_format_tl,
}
{
  \AssignTemplateKeys
  \bool_if:NTF \l_names_reversed_bool
  {
    \bool_if:NT \l_names_use_last_bool
      {{\tl_use:N \l_names_last_format_tl #2}}
    \bool_if:nT
      {
        \l_names_use_first_bool &&
        \l_names_use_last_bool
      }
      {,~}
    \bool_if:NT \l_names_use_first_bool
      {{\tl_use:N \l_names_first_format_tl #1}}
  }
  {
    \bool_if:NT \l_names_use_first_bool
      {{\tl_use:N \l_names_first_format_tl #1}}
    \bool_if:nT
      {
        \l_names_use_first_bool &&
        \l_names_use_last_bool
      }
      {\tl_use:N \c_space_tl}
    \bool_if:NT \l_names_use_last_bool
      {{\tl_use:N \l_names_last_format_tl #2}}
  }
}
```

We can reuse most of the variables for the template `initial` but we're going to need a helper function that gets all but the initial of a name. The code is going to become a little bigger. Of course it could be written in a more elegant way but again, this will suffice for our demonstration purposes.

```
% two additional variables:
\bool_new:N \l_names_last_initial_bool
\bool_new:N \l_names_first_initial_bool

% helper function:
\cs_new:Npn \names_get_initial:w #1#2\q_stop
  {#1.}

% the template code:
% #1: lastname, #2: firstname
\DeclareTemplateCode {names}{initial}{2}
{
  reversed           = \l_names_reversed_bool     ,
  use-last-name      = \l_names_use_last_bool     ,
  use-first-name     = \l_names_use_first_bool    ,
  last-name-format   = \l_names_last_format_tl    ,
  first-name-format  = \l_names_first_format_tl   ,
  last-name-initial  = \l_names_last_initial_bool ,
  first-name-initial = \l_names_first_initial_bool,
}
{
  \AssignTemplateKeys
  \bool_if:NTF \l_names_reversed_bool
  {
    \bool_if:NT \l_names_use_last_bool
      {
        \group_begin:
          \tl_use:N \l_names_last_format_tl
          \bool_if:NTF
            \l_names_last_initial_bool
            {\names_get_initial:w #2\q_stop}
            {#2}
        \group_end:
      }
    \bool_if:nT
      {
        \l_names_use_first_bool &&
        \l_names_use_last_bool
      }
      {,~}
    \bool_if:NT \l_names_use_first_bool
      {
        \group_begin:
          \tl_use:N \l_names_first_format_tl
          \bool_if:NTF
            \l_names_first_initial_bool
            {\names_get_initial:w #1\q_stop}
            {#1}
        \group_end:
      }
  }
  {
    \bool_if:NT \l_names_use_first_bool
      {
        \group_begin:
          \tl_use:N \l_names_first_format_tl
          \bool_if:NTF
            \l_names_first_initial_bool
            {\names_get_initial:w #1\q_stop}
            {#1}
        \group_end:
      }
    \bool_if:nT
      {
        \l_names_use_first_bool &&
        \l_names_use_last_bool
      }
      {\tl_use:N \c_space_tl}
    \bool_if:NT \l_names_use_last_bool
      {
        \group_begin:
          \tl_use:N \l_names_last_format_tl
          \bool_if:NTF
            \l_names_last_initial_bool
            {\names_get_initial:w #2 \q_stop}
            {#2}
        \group_end:
      }
  }
}
```

We're nearly there. For every template we need to declare at least one instance. And of course we need to define the user command.

3.3 The instances

Declaring the instances is not complicated at all. You choose the template and assign values to the parameters. Here we will make three instances for each template:

```
% a few instances, starting with 'fullname':
\DeclareInstance {names}{standard}{fullname}{}
\DeclareInstance {names}{it-rev}{fullname}
  {
    first-name-format = \itshape ,
    reversed          = true
  }
\DeclareInstance {names}{first-only}{fullname}
  {use-last-name = false}
% and now 'initial':
\DeclareInstance {names}{init-first}{initial}{}
\DeclareInstance {names}{init-it-rev}{initial}
  {
    first-name-format = \itshape ,
    reversed          = true
  }
\DeclareInstance {names} init-all}{initial}
  {last-name-initial = true}
```

Defining more instances wouldn't be any problem. With every additional instance the user command we're going to define next would get another option.

3.4 The user command

For the definition of this command we're going to use the package xparse. This makes it easy to define the wanted argument input: lastname and firstname separated with a blank space.

The command is going to get an optional argument with which the instance to be used can be specified. It should test if the chosen instance exists and if not, use the `standard` instance. Of course it could also raise a warning or an error.

```
% yet more variables:
\tl_new:N  \l_names_instance_tl
\tl_set:Nn \l_names_instance_tl { standard }

% the internal command:
\cs_new:Npn \names_typeset_name:nnn #1#2#3
  {
    \IfInstanceExistTF {names} {#1}
      { \UseInstance {names} {#1} }
      { \UseInstance {names} {standard} }
    {#2} {#3}
  }
\cs_generate_variant:Nn
  \names_typeset_name:nnn {V}

% the user command:
\DeclareDocumentCommand \name
  {o>{\SplitArgument{1}{~}}m}
  {
    \group_begin:
      \IfNoValueF {#1}
        {\tl_set:Nn \l_names_instance_tl {#1}}
      \names_typeset_name:Vnn
        \l_names_instance_tl #2
    \group_end:
  }
\ExplSyntaxOff
```

Now we're ready to comprehend the examples from the beginning (and a few others):

```
\name{Jack Skellington} \par
\name[it-rev]{Jack Skellington} \par
\name[first-only]{Jack Skellington} \par
\name[init-first]{Jack Skellington} \par
\name[init-it-rev]{Jack Skellington} \par
\name[init-all]{Jack Skellington}
```

And the output:

Jack Skellington
Skellington, *Jack*
Jack
J. Skellington
Skellington, *J.*
J. S.

4 Before the end

I hope this little excursion can provide a first insight into the functionality of xtemplate. My own knowledge is not much deeper. In my opinion the idea behind xtemplate has a great future and I am excited to see how it will be used in LaTeX3.

References

[1] Frank Mittelbach. LaTeX3 architecture and current work in progress, 2011. `http://river-valley.tv/latex3-architecture-and-current-work-in-progress`.

[2] Frank Mittelbach, David Carlisle, and Chris Rowley. New interfaces for LaTeX class design. 1999. `http://www.latex-project.org/papers/tug99.pdf`.

[3] The LaTeX3 Project. The xparse package. Available from CTAN, `macros/latex/contrib/l3packages/xparse.dtx`.

[4] The LaTeX3 Project. The xtemplate package. Available from CTAN, `macros/latex/contrib/l3packages/xtemplate.dtx`.

[5] Emit Taste. Macro for formatting names (initials or full name). `http://tex.stackexchange.com/q/57636/5049`, 2012.

◇ Clemens Niederberger
Am Burgrain 3
71083 Herrenberg
Germany
contact (at) mychemistry dot eu
http://www.mychemistry.eu/