The obj module
Objects for expl3*

Sean Allred†

Released 2015-07-08

This module provides simple support for objects in expl3.

Example of use

```
\obj_new:nn { rectangle }
  {
    length: int,
    width: int,
    stroke-color: color = black,
    fill-color: color = white
  }

\obj_new:nn { rectangle / rhombus }
  { angle: int = 45 }

\rhombus_new:Nn \l_demo_rhombus
  {
    length = 4,
    stroke-color = gray,
  }

\rhombus_show:N \l_demo_rhombus
```

```
The property list \l_demo_rhombus contains the pairs [...]:
>  {length}  =>  {4}
>  {width}  =>  {}
>  {stroke-color}  =>  {gray}
>  {fill-color}  =>  {white}
>  {angle}  =>  {45}.
```

*This file describes v0.2, last revised 2015-07-08.

†E-mail: tex@seanallred.com

Contents

Introduction

Object-orientism is a well-known paradigm in computer programming. Simply put, it is a way of thinking about programs that perceives its logic to be manipulating 'objects': setting and retrieving properties of any given object and using them to drive the program.

Implementing this in TEX is a challenge made much easier with expl3. By cleverly using property lists, classes and objects can be simply realized without much trouble. This implementation supports data types, defaults, and method creation (as well as the plumbing of instantiation/population).

Part I
Interface Documentation

\obj_new:nn `\obj_new:nn {⟨class-name⟩} {⟨specification⟩}`

New: 2015-07-08

\obj_show:n `\obj_show:n {⟨class-name⟩}`

New: 2015-07-08

\obj_method:nnn `\obj_method:nnn {⟨class-name⟩} {⟨method-name⟩} {⟨implementation⟩}`

New: 2015-07-08

\obj_this:nN `\obj_this:nN {⟨field-name⟩} ⟨destination-var⟩`

New: 2015-07-08

Created dynamically by `\obj_method:nnn`,

\obj_use_method:nn `\obj_use_method:nn {⟨class-name⟩} {⟨method-name⟩}`

New: 2015-07-08

\g__obj_types_clist

New: 2015-07-08

The list of types known to obj. By default, the 'primitive' types of expl3 are available. Not much is done with this value, currently. Eventually, it will be used to automatically instantiate fields. (A corresponding 'no-instantiate' list will also need to be maintained; e.g., it doesn't make sense to create an 'onject' field.)

Part II
Examples

Part III
Implementation

Before I begin, I want to establish a few definitions:

object A complex collection of data with defined behavior. These are also called instantiations.

class A definition of (or blueprint for) a kind of object. Often used interchangeably with 'type'.

super-class, sub-class ***

method A function designed to work with a particular class of objects.

field A piece of data that belongs to an object.

```
1 ⟨*package⟩
2 ⟨@@=obj⟩
```

1 Dependencies

This module depends on parsing. It is included with this documentation, but is not documented itself. Its status is still in limbo (much like this module, but less so).

```
3 \file_input:n { parsing }
```

2 Creating classes

Classes are implemented as a pair of property lists:

specification A mapping of fields to types.

prototype A mapping of fields to default values.

These property lists are both stored in a sequence of the form `\g__obj_spec_?_seq`, where `?` is the class name. This sequence will only ever have two items: the specification is on the left while the prototype is on the right.

They were collected into one sequence to keep the namespace free of clutter and to ease confusion during the development of this package, but storing the property lists 'naked' in their own variables would be faster.

`\obj_new:nn` When a new class is defined, several actions have to take place:

- The specification and the prototype must be created. (These must obey any inheritence specified.)
- Each field must be processed and added to the specification and prototype as appropriate.

- Certain plumbing methods must be created to instantiate objects and set/retrieve the data in their fields.

For ease of reading, these parts have been split up. In a maintenance release, these different pieces will be merged together. For this first release though, it is much easier to take advantage of `\cs_generate_variant:Nn` to simplify the code for review.

```
\tl_new:N \l__obj_class_name_tl
\cs_new:Nn \obj_new:nn
  {
    \__obj_define:nnN {#1} {#2} \l__obj_class_name_tl
    \__obj_process_fields:Vn \l__obj_class_name_tl {#2}
    \__obj_make_metas:V \l__obj_class_name_tl
  }
```

(*End definition for* `\obj_new:nn`*. This function is documented on page 3.*)

`\__obj_define:nnN` A `name` of the form `{ superclass / class }` is pieced out into ⟨*superclass*⟩ and ⟨*class*⟩. ⟨*class*⟩ is placed into ⟨*tl-var*⟩ and ⟨*superclass*⟩ is placed into `\l__obj_tmp_tl`. If there is no superclass, ⟨*class*⟩ is created without inheriting from any other class (i.e., there is no pre-existing specification or prototype).

```
\prop_new:N \l__obj_spec_prop
\prop_new:N \l__obj_proto_prop
\cs_new:Nn \__obj_define:nnN
  {
    \seq_set_split:Nnn \l__obj_tmp_seq { / } {#1}
    \seq_pop_right:NN  \l__obj_tmp_seq #3
    \seq_pop_right:NN  \l__obj_tmp_seq \l__obj_tmp_tl
    \quark_if_no_value:NTF \l__obj_tmp_tl
      { \__obj_setup:VNN #3 \c_empty_prop \c_empty_prop }
      {
```

Otherwise, we grab the superclass's specification and prototype and base our new object off of it.

```
        \__obj_spec:VN      \l__obj_tmp_tl    \l__obj_spec_prop
        \__obj_proto:VN     \l__obj_tmp_tl    \l__obj_proto_prop
        \__obj_setup:VNN #3 \l__obj_spec_prop \l__obj_proto_prop
```

We have to make sure that defaults for existing fields are updated correctly.

```
        \__obj_proto:VN #3 \l__obj_tmp_prop
        \parse_dictionary:nn {#2}
          {
            \prop_if_in:NVF \l__obj_proto_prop \l_parse_dictionary_key_tl
              {
                \prop_put:NVV \l__obj_tmp_prop
                  \l_parse_dictionary_key_tl
                  \l_parse_dictionary_value_tl
              }
          }
        \__obj_proto:NV \l__obj_tmp_prop \l__obj_tmp_tl
      }
  }
```

(*End definition for* `\__obj_define:nnN`.)

`\__obj_process_fields:nn`

```
\cs_new:Nn \__obj_process_fields:nn
  {
    \parse_dictionary:nn {#2}
      {
```

If we don't recognize this type, issue a warning.

```
        \clist_if_in:NVF
          \g__obj_types_clist
          \l_parse_dictionary_type_tl
          {
            \msg_warning:nnx { __obj } { unknown-type }
              { \tl_use:N \l_parse_dictionary_type_tl }
          }
```

Create the specification property list.

```
        \__obj_spec:nN {#1} \l__obj_tmp_prop
        \prop_put:NVV \l__obj_tmp_prop
          \l_parse_dictionary_key_tl
          \l_parse_dictionary_type_tl
        \__obj_spec:Nn \l__obj_tmp_prop {#1}
```

Create the prototype.

```
        \__obj_proto:nN {#1} \l__obj_tmp_prop
        \prop_put:NVV \l__obj_tmp_prop
          \l_parse_dictionary_key_tl
          \l_parse_dictionary_value_tl
        \__obj_proto:Nn \l__obj_tmp_prop {#1}
      }
  }
```

(*End definition for* `\__obj_process_fields:nn`.)

`\__obj_make_metas:nn`

```
\cs_new:Nn \__obj_make_metas:n
  {
    \obj_method:nnn {#1} { show:N } { \prop_show:N ##1 }
    \obj_method:nnn {#1} { new:N }
      {
        \__obj_proto:nN {#1} \l__obj_tmp_prop
        \prop_set_eq:NN ##1  \l__obj_tmp_prop
      }
    \obj_method:nnn {#1} { new:Nn }
      {
        \obj_use_method:nn {#1} { new:N } ##1
        \obj_use_method:nn {#1} { set:Nn } ##1 {##2}
```

put in defaults

```
      \__obj_proto:nN {#1} \l__obj_tmp_prop
      \prop_map_inline:Nn \l__obj_tmp_prop
        { \prop_put_if_new:Nnn ##1 {####1} {####2} }
    }
  \obj_method:nnn {#1} { get:Nn }
    { \prop_item:Nn ##1 {##2} }
  \obj_method:nnn {#1} { set:Nn }
    {
      \prop_clear_new:N ##1
      \__obj_spec:nN {#1} \l__obj_tmp_prop
      \parse_prop:nn {##2}
        {
          \prop_if_in:NVF \l__obj_tmp_prop \l_parse_prop_key_tl
            {
              \msg_warning:nnxx { obj } { unknown-field } {#1}
                { \tl_use:N \l_parse_prop_key_tl }
            }
          \prop_put:NVV ##1 \l_parse_prop_key_tl \l_parse_prop_value_tl
        }
    }
}
```

(*End definition for* `\__obj_make_metas:nn`.)

`\obj_show:n`

```
\cs_new:Nn \obj_show:n
  {
    \__obj_spec:nN {#1} \l__obj_tmp_prop
    \prop_show:N \l__obj_tmp_prop
    \__obj_proto:nN {#1} \l__obj_tmp_prop
    \prop_show:N \l__obj_tmp_prop
  }
```

(*End definition for* `\obj_show:n`. *This function is documented on page 3.*)

`\obj_method:nnn`
`\obj_use_method:nn`
`\obj_this:nN`

```
\cs_new:Nn \obj_method:nnn
  {
    \cs_new:cn { #1_#2 }
      {
        \cs_set:Nn \obj_this:nN
          { \prop_get:NnN ##1 {####1} ####2 }
        #3
      }
  }
\cs_new:Nn \obj_use_method:nn { \use:c { #1_#2 } }
```

(*End definition for* `\obj_method:nnn`, `\obj_use_method:nn`, *and* `\obj_this:nN`. *These functions are documented on page 3.*)

__obj_make_getter:nn
__obj_make_getter:nn

```
\cs_new:Nn \__obj_make_getter:nn
  { \obj_method:nnn {#1} { #2:N  } { \prop_item:Nn ##1 {#2}       } }
\cs_new:Nn \__obj_make_setter:nn
  { \obj_method:nnn {#1} { #2:Nn } { \prop_put:Nnn ##1 {#2} {##2} } }
```

(*End definition for* `\__obj_make_getter:nn` *and* `\__obj_make_getter:nn`.)

__obj_setup:nNN
__obj_setup:n

```
\cs_new:Nn \__obj_setup:nNN
  {
    \seq_new:c { g__obj_spec_#1_seq }
    \seq_put_left:cV  { g__obj_spec_#1_seq } #2
    \seq_put_right:cV { g__obj_spec_#1_seq } #3
  }
```

No superclass. Base off the empty property lists.

```
\cs_new:Nn \__obj_setup:n
  { \__obj_setup:nNN {#1} \c_empty_prop \c_empty_prop }
```

(*End definition for* `\__obj_setup:nNN` *and* `\__obj_setup:n`.)

__obj_spec:nN
__obj_proto:nN
__obj_spec:Nn
__obj_proto:Nn

```
\cs_new:Nn \__obj_spec:nN  { \seq_get_left:cN  { g__obj_spec_#1_seq } #2 }
\cs_new:Nn \__obj_proto:nN { \seq_get_right:cN { g__obj_spec_#1_seq } #2 }
```

Set functions.

```
\cs_new:Nn \__obj_spec:Nn
  {
    \seq_pop_left:cN  { g__obj_spec_#2_seq } \l__obj_tmp_seq
    \seq_put_left:cV  { g__obj_spec_#2_seq } #1
  }
\cs_new:Nn \__obj_proto:Nn
  {
    \seq_pop_right:cN { g__obj_spec_#2_seq } \l__obj_tmp_seq
    \seq_put_right:cV { g__obj_spec_#2_seq } #1
  }
```

(*End definition for* `\__obj_spec:nN` *and others.*)

3 Messages

```
\tl_new:N \l__obj_tmp_tl
\seq_new:N \l__obj_tmp_seq
\clist_new:N \g__obj_types_clist
\clist_set:Nn \g__obj_types_clist
  { obj, tl, seq, int, color }
\msg_new:nnn { obj } { unknown-type }
  {
```

```
    The~type~you~have~provided~is~not~known~to~exist:~
    `#1'
  }
\msg_new:nnn { obj } { unknown-field }
  {
    The~field~you~have~provided~is~not~known~to~exist~
    for~this~object:~`#1.#2'
  }
```

4 Variants

```
\cs_generate_variant:Nn \__obj_make_getter:nn    { VV }
\cs_generate_variant:Nn \__obj_make_setter:nn    { VV }
\cs_generate_variant:Nn \__obj_setup:n           { V  }
\cs_generate_variant:Nn \__obj_setup:nNN         { V  }
\cs_generate_variant:Nn \__obj_spec:nN           { V  }
\cs_generate_variant:Nn \__obj_proto:nN          { V  }
\cs_generate_variant:Nn \__obj_spec:Nn           { NV }
\cs_generate_variant:Nn \__obj_proto:Nn          { NV }
\cs_generate_variant:Nn \__obj_process_fields:nn { V  }
\cs_generate_variant:Nn \__obj_make_metas:n      { V  }
```

⟨/package⟩

A To-Do

- Support .-scoping in `\obj_this:nN`:

  ```
  \obj_method:nnn { car } { shiny_rims:N }
    { \obj_this:nN { wheel . rim . color } \l_my_rim_color }
  ```

- Enforce `N`-type as `#1` in `#2` of `\obj_method:nnn`. Dynamic `\obj_this:n` creation in `\obj_method:nnn` depends on `#1` of the method being the object itself.

B Complete Example