

# The lt3rawobjects package

Paolo De Donato

Released on 2023/02/16 Version 2.3-beta-2

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Addresses</b>	<b>2</b>
<b>3</b>	<b>Objects</b>	<b>3</b>
<b>4</b>	<b>Items</b>	<b>3</b>
4.1	Constants . . . . .	4
4.2	Methods . . . . .	4
4.3	Members . . . . .	4
<b>5</b>	<b>Object members</b>	<b>4</b>
5.1	Create a pointer member . . . . .	4
5.2	Clone the inner structure . . . . .	5
5.3	Embedded objects . . . . .	6
<b>6</b>	<b>Library functions</b>	<b>7</b>
6.1	Common functions . . . . .	7
6.2	Base object functions . . . . .	7
6.3	Members . . . . .	8
6.4	Constants . . . . .	9
6.5	Methods . . . . .	10
6.6	Creation of constants . . . . .	11
6.7	Macros . . . . .	11
6.8	Proxies and object creation . . . . .	12
<b>7</b>	<b>Examples</b>	<b>14</b>
<b>8</b>	<b>Implementation</b>	<b>16</b>

## 1 Introduction

Package `lt3rawobjects` introduces a new mechanism to create and manage structured data called “objects” like the well known C structures. The functions exported by this package are quite low level, and many important mechanisms like member protection and name resolution aren’t already defined and should be introduced by intermediate packages. Higher level libraries built on top of `lt3rawobjects` could also implement an improved and simplified syntax since the main focus of `lt3rawobjects` is versatility and expandability rather than common usage.

This packages follows the [SemVer](https://semver.org/) specification (<https://semver.org/>). In particular any major version update (for example from 1.2 to 2.0) may introduce incompatible changes and so it’s not advisable to work with different packages that require different major versions of `lt3rawobjects`. Instead changes introduced in minor and patch version updates are always backward compatible, and any withdrawn function is declared deprecated instead of being removed.

## 2 Addresses

In this package a *pure address* is any string without spaces (so a sequence of tokens with category code 12 “other”) that uniquely identifies a resource or an entity. An example of pure address is the name of a control sequence `\<name>` that can be obtained by full expanding `\cs_to_str:N \<name>`. Instead an *expanded address* is a token list that contains only tokens with category code 11 (letters) or 12 (other) that can be directly converted to a pure address with a simple call to `\tl_to_str:n` or by assigning it to a string variable.

An *address* is instead a fully expandable token list which full expansion is an expanded address, where full expansion means the expansion process performed inside `c`, `x` and `e` parameters. Moreover, any address should be fully expandable according to the rules of `x` and `e` parameter types with same results, and the name of control sequence resulting from a `c`-type expansion of such address must be equal to its full expansion. For these reasons addresses should not contain parameter tokens like `#` (because they’re threat differently by `x` and `e`) or control sequences that prevents expansion like `\exp_not:n` (because they leave unexpanded control sequences after an `x` or `e` expansion, and expanded addresses can’t have control sequences inside them). In particular, `\tl_to_str:n{ ## }` is *not* a valid address (assuming standard category codes).

Addresses could be not full expanded inside an `f` argument, thus an address expanded in an `f` argument should be `x`, `e` or `c` expended later to get the actual pure address. If you need to fully expand an address in an `f` argument (because, for example, your macro should be fully expandable and your engine is too old to support `e` expansion efficiently) then you can put your address inside `\rwoobj_address_f:n` and pass them to your function. For example,

```
\your_function:f{ \rwoobj_address_f:n { your \address } }
```

Remember that `\rwoobj_address_f:n` only works with addresses, can’t be used to fully expand any token list.

Like functions and variables names, pure addresses should follows some basic naming conventions in order to avoid clashes between addresses in different modules. Each pure

address starts with the  $\langle module \rangle$  name in which such address is allocated, then an underscore (`_`) and the  $\langle identifier \rangle$  that uniquely identifies the resource inside the module. The  $\langle module \rangle$  should contain only lowercase ASCII letters.

A *pointer* is just a L<sup>A</sup>T<sub>E</sub>X3 string variable that holds a pure address. We don't enforce to use `str` or any special suffix to denote pointers so you're free to use `str` or a custom  $\langle type \rangle$  as suffix for your pointers in order to distinguish between them according to their type.

In `lt3rawobjects` all the macros ending with `_adr` or `_address` are fully expandable and can be used to compose valid addresses as explained in their documentation.

### 3 Objects

An *object* is just a collection of several related entities called *item*. Objects are themselves entities so they have addresses and could be contained inside other objects. Objects addresses are also used to compose the addresses of each of their inner entity, thus different objects can have items with the same name without clashing each other. Each object is uniquely identified by its pure address, which is composed by a  $\langle module \rangle$  and an  $\langle identifier \rangle$  as explained before. The use of underscore character in objects identifiers is reserved. You can retrieve the address of an object via the `\object_address:nn` function.

Objects are always created from already existing objects. An object that can be used to create other objects is called **proxy**, and the proxy that has created an object is its *generator*. In the `rawobjects` module is already allocated a particular proxy that can be used to create every other proxy. Its identifier is just `proxy` and its pure address is stored in `\c_proxy_address_str`. The functions `\object_create` can be used to create new objects.

### 4 Items

Remember that objects are just a collection of different items uniquely identified by a pure address. Here an item could be one of the following entities:

- a L<sup>A</sup>T<sub>E</sub>X3 variable, in which case the item is called *member*;
- a L<sup>A</sup>T<sub>E</sub>X3 constant, in which case the item is called just *constant*;
- a L<sup>A</sup>T<sub>E</sub>X3 function, in which case the item is called *method*;
- generic control sequences, in which case the item is called simply *macro*;
- an entire object, in which case the item is called *embedded object*.

Objects could be declared *local* or *global*. The only difference between a local and a global object is the scope of their members (that are L<sup>A</sup>T<sub>E</sub>X3 variables). You should always create global object unless you specifically need local members.

## 4.1 Constants

Constants in an object could be *near* and *remote*. A near constant is just a constant declared in such object and could be referred only by it, instead a remote constant is declared inside its generator and can be referred by any object created from that proxy, thus it's shared between all the generated objects. Functions in this library that work with near constants usually contain `ncmember` in their names, whereas those involving remote constants contain `rcmember` instead.

Both near and remote constants are created in the same way via the `_newconst` functions, however remote constant should be created in a proxy whereas near constant are created directly in the target object.

## 4.2 Methods

Methods are  $\text{\LaTeX}$  functions that can't be changed once they're created. Like constant, methods could be near or remote. Moreover, functions in this library dealing with near methods contain `ncmethod` whereas those dealing with remote methods contain `rcmethod` in their names.

## 4.3 Members

Members are just mutable  $\text{\LaTeX}$  variables. You can manually create new members in already existing objects or you can put the definition of a new member directly in a proxy with the `\proxy_push_member` functions. In this way all the objects created with that proxy will have a member according to such definition. If the object is local/global then all its members are automatically local/global.

A member can be *tracked* or *not tracked*. A tracked member have additional information, like its type, stored in the object or in its generator. In particular, you don't need to specify the type of a tracked member and some functions in `lt3rawobjects` are able to retrieve the required information. All the members declared in the generator are automatically tracked.

# 5 Object members

Sometimes it's necessary to store an instance of an object inside another object, since objects are structured entities that can't be entirely contained in a single  $\text{\LaTeX}$  variable you can't just put it inside a member or constant. However, there are some very easy workarounds to insert object instances as items of other objects.

For example, we're in module `MOD` and we have an object with id `PAR`. We want to provide `PAR` with an item that holds an instance of an object created by proxy `PRX`. We can achieve this in three ways:

## 5.1 Create a pointer member

We first create a new object from `PRX`

```
1 \object_create:nnn
2 { \object_address:nn { MOD }{ PRX } }{ MOD }{ INST }
```

then we create an **str** member in **PAR** that will hold the address of the newly created object.

```

1  \object_new_member:nnn
2  {
3      \object_address:nn { MOD }{ PAR }
4  }{ pointer }{ str }
5
6  \object_member_set:nnnx
7  {
8      \object_address:nn { MOD }{ PAR }
9  }
10 { pointer }{ str }
11 {
12     \object_address:nn { MOD }{ INST }
13 }

```

You can then get the pointed object by just using the **pointer** member. Notice that you're not forced to use the **str** type for the pointer member, but you can also use **tl** or any custom  $\langle type \rangle$ . In the latter case be sure to at least define the following functions:  $\backslash\langle type \rangle\_new:c$ ,  $\backslash\langle type \rangle\_set:cn$  and  $\backslash\langle type \rangle\_use:c$ .

### Advantages

- Simple and no additional function needed to create and manage included objects;
- you can share the same object between different containers;
- included objects are objects too, you can use address stored in pointer member just like any object address.

### Disadvantages

- You must manually create both the objects and link them;
- if you forgot to properly initialize the pointer member it'll contain the **null address** (the empty string). Despite other programming languages the null address is not treated specially by **lt3rawobjects**, which makes finding null pointer errors more difficult.

## 5.2 Clone the inner structure

Another solution is to copy the members declared in **PRX** to **PAR**. For example, if in **PRX** are declared a member with name **x** and type **str**, and a member with name **y** and type **int** then

```

1  \object_new_member:nnn
2  {
3      \object_address:nn { MOD }{ PAR }
4  }{ prx-x }{ str }
5  \object_new_member:nnn
6  {

```

```

7      \object_address:nn { MOD }{ PAR }
8    }{ prx-y }{ int }

```

### Advantages

- Very simple;
- no hidden item is created, this procedure has the lowest overhead among all the proposed solutions here.

### Disadvantages

- If you need the original instance of the stored object then you should create a temporary object and manually copy each item to it. Don't use this method if you later need to retrieve the stored object entirely and not only its items.

## 5.3 Embedded objects

From `lt3rawobjects 2.2` you can put *embedded objects* inside objects. Embedded objects are created with `\embedded_create` function

```

1  \embedded_create:nnn
2  {
3    \object_address:nn { MOD }{ PAR }
4  }
5  { PRX }{ emb }

```

and addresses of embedded objects can be retrieved with function `\object_embedded_adr`. You can also put the definition of embedded objects in a proxy by using `\proxy_push_embedded` just like `\proxy_push_member`.

### Advantages

- You can put a declaration inside a proxy so that embedded objects are automatically created during creation of parent object;
- included objects are objects too, you can use address stored in pointer member just like any object address.

### Disadvantages

- Needs additional functions available for version 2.2 or later;
- embedded objects must have the same scope and visibility of parent one;
- creating objects also creates additional hidden variables, taking so (little) additional space.

## 6 Library functions

### 6.1 Common functions

---

<code>\rwobj_address_f:n</code>	★	<code>\rwobj_address_f:n {⟨address⟩}</code>
---------------------------------	---	---

---

Fully expand an address in an f-type argument.  
From: 2.3

### 6.2 Base object functions

---

<code>\object_address:nn</code>	☆	<code>\object_address:nn {⟨module⟩} {⟨id⟩}</code>
---------------------------------	---	---

---

Composes the address of object in module *⟨module⟩* with identifier *⟨id⟩* and places it in the input stream. Notice that both *⟨module⟩* and *⟨id⟩* are converted to strings before composing them in the address, so they shouldn't contain any command inside.  
From: 1.0

---

<code>\object_address_set:Nnn</code>		<code>\object_address_set:nn ⟨str var⟩ {⟨module⟩} {⟨id⟩}</code>
<code>\object_address_gset:Nnn</code>		

---

Stores the address of selected object inside the string variable *⟨str var⟩*.  
From: 1.1

---

<code>\object_embedded_adr:nn</code>	☆	<code>\object_embedded_adr:nn {⟨address⟩} {⟨id⟩}</code>
<code>\object_embedded_adr:Vn</code>	☆	

---

Compose the address of embedded object with name *⟨id⟩* inside the parent object with address *⟨address⟩*. Since an embedded object is also an object you can use this function for any function that accepts object addresses as an argument.  
From: 2.2

---

<code>\object_if_exist_p:n</code>	★	<code>\object_if_exist_p:n {⟨address⟩}</code>
<code>\object_if_exist_p:V</code>	★	<code>\object_if_exist:nTF {⟨address⟩} {⟨true code⟩} {⟨false code⟩}</code>
<code>\object_if_exist:nTF</code>	★	
<code>\object_if_exist:VTF</code>	★	

---

Tests if an object was instantiated at the specified address.  
From: 1.0

---

<code>\object_get_module:n</code>	★	<code>\object_get_module:n {⟨address⟩}</code>
<code>\object_get_module:V</code>	★	<code>\object_get_proxy_adr:n {⟨address⟩}</code>
<code>\object_get_proxy_adr:n</code>	★	
<code>\object_get_proxy_adr:V</code>	★	

---

Get the object module and its generator.  
From: 1.0

---

<code>\object_if_local_p:n</code>	★	<code>\object_if_local_p:n {⟨address⟩}</code>
<code>\object_if_local_p:V</code>	★	<code>\object_if_local:nTF {⟨address⟩} {⟨true code⟩} {⟨false code⟩}</code>
<code>\object_if_local:nTF</code>	★	
<code>\object_if_local:VTF</code>	★	
<code>\object_if_global_p:n</code>	★	
<code>\object_if_global_p:V</code>	★	
<code>\object_if_global:nTF</code>	★	
<code>\object_if_global:VTF</code>	★	

---

Tests if the object is local or global.  
From: 1.0

---

<code>\object_if_public:p:n</code>	★	<code>\object_if_public:p:n {⟨address⟩}</code>
<code>\object_if_public:p:V</code>	★	<code>\object_if_public:nTF {⟨address⟩} {⟨true code⟩} {⟨false code⟩}</code>
<code>\object_if_public:nTF</code>	★	Tests if the object is public or private.
<code>\object_if_public:VTF</code>	★	From: 1.0
<code>\object_if_private:p:n</code>	★	
<code>\object_if_private:p:V</code>	★	
<code>\object_if_private:nTF</code>	★	
<code>\object_if_private:VTF</code>	★	

---

### 6.3 Members

---

<code>\object_member_adr:nnn</code>	☆	<code>\object_member_adr:nnn {⟨address⟩} {⟨member name⟩} {⟨member type⟩}</code>
<code>\object_member_adr:(Vnn nnv)</code>	☆	<code>\object_member_adr:nn {⟨address⟩} {⟨member name⟩}</code>
<code>\object_member_adr:nn</code>	☆	
<code>\object_member_adr:Vn</code>	☆	

---

Fully expands to the address of specified member variable. If the member is tracked then you can omit the type field.

From: 1.0

---

<code>\object_member_if_exist:p:nnn</code>	★	<code>\object_member_if_exist:p:nnn {⟨address⟩} {⟨member name⟩} {⟨member type⟩}</code>
<code>\object_member_if_exist:p:Vnn</code>	★	<code>type⟩}</code>
<code>\object_member_if_exist:nnnTF</code>	★	<code>\object_member_if_exist:nnnTF {⟨address⟩} {⟨member name⟩} {⟨member type⟩} {⟨true code⟩} {⟨false code⟩}</code>
<code>\object_member_if_exist:VnnTF</code>	★	<code>type⟩} {⟨true code⟩} {⟨false code⟩}</code>

---

Tests if the specified member exist.

From: 2.0

---

<code>\object_member_if_tracked:p:nn</code>	★	<code>\object_member_if_tracked:p:nn {⟨address⟩} {⟨member name⟩}</code>
<code>\object_member_if_tracked:p:Vn</code>	★	<code>\object_member_if_tracked:nnTF {⟨address⟩} {⟨member name⟩} {⟨true code⟩} {⟨false code⟩}</code>
<code>\object_member_if_tracked:nnTF</code>	★	<code>code⟩} {⟨false code⟩}</code>
<code>\object_member_if_tracked:VnTF</code>	★	

---

Tests if the specified member exist and is tracked.

From: 2.3

---

<code>\object_member_type:nn</code>	★	<code>\object_member_type:nn {⟨address⟩} {⟨member name⟩}</code>
<code>\object_member_type:Vn</code>	★	Fully expands to the type of specified tracked member.

---

From: 1.0

---

<code>\object_new_member:nnn</code>		<code>\object_new_member:nnn {⟨address⟩} {⟨member name⟩} {⟨member type⟩}</code>
<code>\object_new_member:(Vnn nnv)</code>		

---

Creates a new member variable with specified name and type. You can't retrieve the type of these variables with `\object_member_type` functions.

From: 1.0

---

<code>\object_member_use:nnn</code>	★	<code>\object_member_use:nnn {⟨address⟩} {⟨member name⟩} {⟨member type⟩}</code>
<code>\object_member_use:(Vnn nnv)</code>	★	<code>\object_member_use:nn {⟨address⟩} {⟨member name⟩}</code>
<code>\object_member_use:nn</code>	★	
<code>\object_member_use:Vn</code>	★	

---

Uses the specified member variable.

From: 1.0



---

<code>\object_member_set:nnnn</code>	<code>\object_member_set:nnnn {&lt;address&gt;} {&lt;member name&gt;} {&lt;member type&gt;}</code>
<code>\object_member_set:(nnvn Vnnn)</code>	<code>{&lt;value&gt;}</code>
<code>\object_member_set:nnn</code>	<code>\object_member_set:nnn {&lt;address&gt;} {&lt;member name&gt;} {&lt;value&gt;}</code>
<code>\object_member_set:Vnn</code>	

---

Sets the value of specified member to `{<value>}`. It calls implicitly `\<member type>_(g)set:cn` then be sure to define it before calling this method.

From: 2.1

---

<code>\object_member_set_eq:nnnN</code>	<code>\object_member_set_eq:nnnN {&lt;address&gt;} {&lt;member name&gt;}</code>
<code>\object_member_set_eq:(nnvN VnnN nnnc Vnnnc)</code>	<code>{&lt;member type&gt;} {&lt;variable&gt;}</code>
<code>\object_member_set_eq:nnN</code>	<code>\object_member_set_eq:nnN {&lt;address&gt;} {&lt;member name&gt;}</code>
<code>\object_member_set_eq:(VnN nnnc Vnc)</code>	<code>{&lt;variable&gt;}</code>

---

Sets the value of specified member equal to the value of `<variable>`.

From: 1.0

## 6.4 Constants

---

<code>\object_ncmember_adr:nnn</code>	☆	<code>\object_ncmember_adr:nnn {&lt;address&gt;} {&lt;member name&gt;} {&lt;member type&gt;}</code>
<code>\object_ncmember_adr:(Vnn vnn)</code>	☆	
<code>\object_rcmember_adr:nnn</code>	☆	
<code>\object_rcmember_adr:Vnn</code>	☆	

---

Fully expands to the address of specified near/remote constant member.

From: 2.0

---

<code>\object_ncmember_if_exist_p:nnn *</code>	<code>\object_ncmember_if_exist_p:nnn {&lt;address&gt;} {&lt;member name&gt;} {&lt;member type&gt;}</code>
<code>\object_ncmember_if_exist_p:Vnn *</code>	<code>{&lt;member type&gt;}</code>
<code>\object_ncmember_if_exist:nnnTF *</code>	<code>\object_ncmember_if_exist:nnnTF {&lt;address&gt;} {&lt;member name&gt;} {&lt;member type&gt;}</code>
<code>\object_ncmember_if_exist:VnnTF *</code>	<code>{&lt;true code&gt;} {&lt;false code&gt;}</code>
<code>\object_rcmember_if_exist_p:nnn *</code>	
<code>\object_rcmember_if_exist_p:Vnn *</code>	
<code>\object_rcmember_if_exist:nnnTF *</code>	
<code>\object_rcmember_if_exist:VnnTF *</code>	

---

Tests if the specified member constant exist.

From: 2.0

---

<code>\object_ncmember_use:nnn *</code>	<code>\object_ncmember_use:nnn {&lt;address&gt;} {&lt;member name&gt;} {&lt;member type&gt;}</code>
<code>\object_ncmember_use:Vnn *</code>	
<code>\object_rcmember_use:nnn *</code>	Uses the specified near/remote constant member.
<code>\object_rcmember_use:Vnn *</code>	From: 2.0

---

## 6.5 Methods

---

<code>\object_ncmethod_adr:nnn</code>	☆	<code>\object_ncmethod_adr:nnn {⟨address⟩} {⟨method name⟩} {⟨method</code>
<code>\object_ncmethod_adr:(Vnn vnn)</code>	☆	<code>variant⟩}</code>
<code>\object_rcmethod_adr:nnn</code>	☆	
<code>\object_rcmethod_adr:Vnn</code>	☆	

---

Fully expands to the address of the specified

- near constant method if `\object_ncmethod_adr` is used;
- remote constant method if `\object_rcmethod_adr` is used.

From: 2.0

---

<code>\object_ncmethod_if_exist_p:nnn</code>	★	<code>\object_ncmethod_if_exist_p:nnn {⟨address⟩} {⟨method name⟩} {⟨method</code>
<code>\object_ncmethod_if_exist_p:Vnn</code>	★	<code>variant⟩}</code>
<code>\object_ncmethod_if_exist:nnnTF</code>	★	<code>\object_ncmethod_if_exist:nnnTF {⟨address⟩} {⟨method name⟩} {⟨method</code>
<code>\object_ncmethod_if_exist:VnnTF</code>	★	<code>variant⟩} {⟨true code⟩} {⟨false code⟩}</code>
<code>\object_rcmethod_if_exist_p:nnn</code>	★	
<code>\object_rcmethod_if_exist_p:Vnn</code>	★	
<code>\object_rcmethod_if_exist:nnnTF</code>	★	
<code>\object_rcmethod_if_exist:VnnTF</code>	★	

---

Tests if the specified method constant exist.

From: 2.0

---

<code>\object_new_cmethod:nnnn</code>	<code>\object_new_cmethod:nnnn {⟨address⟩} {⟨method name⟩} {⟨method arguments⟩} {⟨code⟩}</code>
<code>\object_new_cmethod:Vnnn</code>	

---

Creates a new method with specified name and argument types. The `{⟨method arguments⟩}` should be a string composed only by n and N characters that are passed to `\cs_new:Nn`.

From: 2.0

---

<code>\object_ncmethod_call:nnn</code>	★	<code>\object_ncmethod_call:nnn {⟨address⟩} {⟨method name⟩} {⟨method variant⟩}</code>
<code>\object_ncmethod_call:Vnn</code>	★	
<code>\object_rcmethod_call:nnn</code>	★	
<code>\object_rcmethod_call:Vnn</code>	★	

---

Calls the specified method. This function is expandable if and only if the specified method was not declared `protected`.

From: 2.0

## 6.6 Creation of constants

---

```

\object_newconst_tl:nnn
\object_newconst_tl:Vnn
\object_newconst_str:nnn
\object_newconst_str:Vnn
\object_newconst_int:nnn
\object_newconst_int:Vnn
\object_newconst_clist:nnn
\object_newconst_clist:Vnn
\object_newconst_dim:nnn
\object_newconst_dim:Vnn
\object_newconst_skip:nnn
\object_newconst_skip:Vnn
\object_newconst_fp:nnn
\object_newconst_fp:Vnn

```

---

```

\object_newconst_⟨type⟩:nnn {⟨address⟩} {⟨constant name⟩} {⟨value⟩}

```

Creates a constant variable with type  $\langle type \rangle$  and sets its value to  $\langle value \rangle$ .

From: 1.1

---

```

\object_newconst_seq_from_clist:nnn \object_newconst_seq_from_clist:nnn {⟨address⟩} {⟨constant name⟩}
\object_newconst_seq_from_clist:Vnn {⟨comma-list⟩}

```

---

Creates a `seq` constant which is set to contain all the items in  $\langle comma-list \rangle$ .

From: 1.1

---

```

\object_newconst_prop_from_keyval:nnn \object_newconst_prop_from_keyval:nnn {⟨address⟩} {⟨constant
\object_newconst_prop_from_keyval:Vnn name⟩}
{
  ⟨key⟩ = ⟨value⟩, ...
}

```

---

Creates a `prop` constant which is set to contain all the specified key-value pairs.

From: 1.1

---

```

\object_newconst:nnnn \object_newconst:nnnn {⟨address⟩} {⟨constant name⟩} {⟨type⟩} {⟨value⟩}

```

---

Invokes  $\backslash\langle type \rangle\_const:cn$  to create the specified constant.

From: 2.1

## 6.7 Macros

---

```

\object_macro_adr:nn ☆
\object_macro_adr:Vn ☆

```

---

```

\object_macro_adr:nn {⟨address⟩} {⟨macro name⟩}

```

Address of specified macro.

From: 2.2

---

```

\object_macro_use:nn ☆
\object_macro_use:Vn ☆

```

---

```

\object_macro_use:nn {⟨address⟩} {⟨macro name⟩}

```

Uses the specified macro. This function is expandable if and only if the specified macro is it.

From: 2.2

There isn't any standard function to create macros, and macro declarations can't be inserted in a `proxy` object. In fact a macro is just an unspecialized control sequence at the disposal of users that usually already know how to implement them.

## 6.8 Proxies and object creation

<hr/>	
<code>\object_if_proxy_p:n *</code>	<code>\object_if_proxy_p:n {⟨address⟩}</code>
<code>\object_if_proxy_p:V *</code>	<code>\object_if_proxy:nTF {⟨address⟩} {⟨true code⟩} {⟨false code⟩}</code>
<code>\object_if_proxy:nTF *</code>	Test if the specified object is a proxy object.
<code>\object_if_proxy:VTF *</code>	From: 1.0
<hr/>	
<code>\object_test_proxy_p:nn *</code>	<code>\object_test_proxy_p:nn {⟨object address⟩} {⟨proxy address⟩}</code>
<code>\object_test_proxy_p:Vn *</code>	<code>\object_test_proxy:nnTF {⟨object address⟩} {⟨proxy address⟩} {⟨true code⟩} {⟨false code⟩}</code>
<code>\object_test_proxy:nnTF *</code>	
<code>\object_test_proxy:VnTF *</code>	Test if the specified object is generated by the selected proxy, where <i>⟨proxy variable⟩</i> is a string variable holding the proxy address.
	<b>TeXhackers note:</b> Remember that this command uses internally an <b>e</b> expansion so in older engines (any different from Lua <sup>®</sup> TeX before 2019) it'll require slow processing. Don't use it in speed critical parts, instead use <code>\object_test_proxy:nN</code> .
	From: 2.0
<hr/>	
<code>\object_test_proxy_p:nN *</code>	<code>\object_test_proxy_p:nN {⟨object address⟩} {⟨proxy variable⟩}</code>
<code>\object_test_proxy_p:VN *</code>	<code>\object_test_proxy:nNTF {⟨object address⟩} {⟨proxy variable⟩} {⟨true code⟩} {⟨false code⟩}</code>
<code>\object_test_proxy:nNTF *</code>	
<code>\object_test_proxy:VNNTF *</code>	Test if the specified object is generated by the selected proxy, where <i>⟨proxy variable⟩</i> is a string variable holding the proxy address. The <code>:nN</code> variant don't use <b>e</b> expansion, instead of <code>:nn</code> command, so it can be safely used with older compilers.
	From: 2.0
<hr/>	
<code>\c_proxy_address_str</code>	The address of the proxy object in the <code>rawobjects</code> module.
	From: 1.0
<hr/>	
<code>\object_create:nnnNN</code>	<code>\object_create:nnnNN {⟨proxy address⟩} {⟨module⟩} {⟨id⟩} {⟨scope⟩} {⟨visibility⟩}</code>
<code>\object_create:VnnNN</code>	
	Creates an object by using the proxy at <i>⟨proxy address⟩</i> and the specified parameters. Use this function only if you need to create private objects (at present private objects are functionally equivalent to public objects) or if you need to compile your project with an old version of this library (< 2.3).
	From: 1.0
<hr/>	
<code>\object_create:nnnN</code>	<code>\object_create:nnnN {⟨proxy address⟩} {⟨module⟩} {⟨id⟩} {⟨scope⟩}</code>
<code>\object_create:VnnN</code>	<code>\object_create:nnn {⟨proxy address⟩} {⟨module⟩} {⟨id⟩}</code>
<code>\object_create:nnn</code>	Same as <code>\object_create:nnnNN</code> but both create only public objects, and the <code>:nnn</code> version only global ones. Always use these two function instead of <code>\object_create:nnnNN</code> unless you strictly need private objects.
<code>\object_create:Vnn</code>	From: 2.3
<hr/>	
<code>\embedded_create:nnn</code>	<code>\embedded_create:nnn {⟨parent object⟩} {⟨proxy address⟩} {⟨id⟩}</code>
<code>\embedded_create:(Vnn nvn)</code>	Creates an embedded object with name <i>⟨id⟩</i> inside <i>⟨parent object⟩</i> .
	From: 2.2

---

<code>\c_object_local_str</code>	Possible values for $\langle scope \rangle$ parameter.
<code>\c_object_global_str</code>	From: 1.0

---



---

<code>\c_object_public_str</code>	Possible values for $\langle visibility \rangle$ parameter.
<code>\c_object_private_str</code>	From: 1.0

---



---

<code>\object_create_set:NnnnNN</code>	<code>\object_create_set:NnnnNN</code> $\langle str var \rangle$ $\{\langle proxy address \rangle\}$ $\{\langle module \rangle\}$
<code>\object_create_set:(NVnnNN NnnfNN)</code>	$\{\langle id \rangle\}$ $\langle scope \rangle$ $\langle visibility \rangle$
<code>\object_create_gset:NnnnNN</code>	
<code>\object_create_gset:(NVnnNN NnnfNN)</code>	

---

Creates an object and sets its fully expanded address inside  $\langle str var \rangle$ .  
From: 1.0

---

<code>\object_allocate_incr:NnnnNN</code>	<code>\object_allocate_incr:NnnnNN</code> $\langle str var \rangle$ $\langle int var \rangle$ $\{\langle proxy address \rangle\}$
<code>\object_allocate_incr:NNVnnNN</code>	$\{\langle module \rangle\}$ $\langle scope \rangle$ $\langle visibility \rangle$
<code>\object_gallocate_incr:NnnnNN</code>	
<code>\object_gallocate_incr:NNVnnNN</code>	
<code>\object_allocate_gincr:NnnnNN</code>	
<code>\object_allocate_gincr:NNVnnNN</code>	
<code>\object_gallocate_gincr:NnnnNN</code>	
<code>\object_gallocate_gincr:NNVnnNN</code>	

---

Build a new object address with module  $\langle module \rangle$  and an identifier generated from  $\langle proxy address \rangle$  and the integer contained inside  $\langle int var \rangle$ , then increments  $\langle int var \rangle$ . This is very useful when you need to create a lot of objects, each of them on a different address. the `_incr` version increases  $\langle int var \rangle$  locally whereas `_gincr` does it globally.  
From: 1.1

---

<code>\proxy_create:nnN</code>	<code>\proxy_create:nnN</code> $\{\langle module \rangle\}$ $\{\langle id \rangle\}$ $\langle visibility \rangle$
<code>\proxy_create_set:NnnN</code>	<code>\proxy_create_set:NnnN</code> $\langle str var \rangle$ $\{\langle module \rangle\}$ $\{\langle id \rangle\}$ $\langle visibility \rangle$
<code>\proxy_create_gset:NnnN</code>	

---

These commands are deprecated because proxies should be global and public. Use instead `\proxy_create:nn`, `\proxy_create_set:Nnn` and `\proxy_create_gset:Nnn`.  
From: 1.0  
Deprecated in: 2.3

---

<code>\proxy_create:nn</code>	<code>\proxy_create:nn</code> $\{\langle module \rangle\}$ $\{\langle id \rangle\}$
<code>\proxy_create_set:Nnn</code>	<code>\proxy_create_set:Nnn</code> $\langle str var \rangle$ $\{\langle module \rangle\}$ $\{\langle id \rangle\}$
<code>\proxy_create_gset:Nnn</code>	

---

Creates a global public proxy object.  
From: 2.3

---

<code>\proxy_push_member:nnn</code>	<code>\proxy_push_member:nnn</code> $\{\langle proxy address \rangle\}$ $\{\langle member name \rangle\}$ $\{\langle member type \rangle\}$
<code>\proxy_push_member:Vnn</code>	

---

Updates a proxy object with a new member specification, so that every subsequential object created with this proxy will have a member variable with the specified name and type that can be retrieved with `\object_member_type` functions.  
From: 1.0

---

```
\proxy_push_embedded:nnn
\proxy_push_embedded:Vnn
```

---

```
\proxy_push_embedded:nnn {⟨proxy address⟩} {⟨embedded object name⟩} {⟨embedded
object proxy⟩}
```

Updates a proxy object with a new embedded object specification.  
From: 2.2

---

```
\proxy_add_initializer:nN
\proxy_add_initializer:VN
```

---

```
\proxy_add_initializer:nN {⟨proxy address⟩} {⟨initializer⟩}
```

Pushes a new initializer that will be executed on each created objects. An initializer is a function that should accept five arguments in this order:

- the full expanded address of used proxy as an `n` argument;
- the module name as an `n` argument;
- the full expanded address of created object as an `n` argument.

Initializer will be executed in the same order they're added.

---

```
\object_assign:nn
\object_assign:(Vn|nV|VV)
```

---

```
\object_assign:nn {⟨to address⟩} {⟨from address⟩}
```

Assigns the content of each variable of object at `⟨from address⟩` to each corresponsive variable in `⟨to address⟩`. Both the objects should be created with the same proxy object and only variables listed in the proxy are assigned.

From: 1.0

## 7 Examples

### Example 1

Create a public proxy with id `myproxy` with the specification of a single member variable with name `myvar` and type `tl`, then set its address inside `\g_myproxy_str`.

```
1 \str_new:N \g_myproxy_str
2 \proxy_create_gset:Nnn \g_myproxy_str { example }{ myproxy }
3 \proxy_push_member:Vnn \g_myproxy_str { myvar }{ tl }
```

Then create a new object with name `myobj` with that proxy, assign then token list `\c_dollar_str{} ~ dollar ~ \c_dollar_str{}` to `myvar` and then print it.

```
1 \str_new:N \g_myobj_str
2 \object_create_gset:NVnn \g_myobj_str \g_myproxy_str
3 { example }{ myobj }
4 \tl_gset:cn
5 {
6   \object_member_adr:Vn \g_myobj_str { myvar }
7 }
8 { \c_dollar_str{} ~ dollar ~ \c_dollar_str{} }
9 \object_member_use:Vn \g_myobj_str { myvar }
```

Output: \$ dollar \$

You can also avoid to specify an object identify and use `\object_gallocate_gincr` instead:

```

1 \int_new:N \g_intc_int
2 \object_gallocate_gincr:NNVnNN \g_myobj_str \g_intc_int \g_myproxy_str
3   { example } \c_object_local_str \c_object_public_str
4 \tl_gset:cn
5   {
6     \object_member_adr:Vn \g_myobj_str { myvar }
7   }
8   { \c_dollar_str{} ~ dollar ~ \c_dollar_str{} }
9 \object_member_use:Vn \g_myobj_str { myvar }

```

Output: \$ dollar \$

## Example 2

In this example we create a proxy object with an embedded object inside.  
Internal proxy

```

1 \proxy_create:nn { mymod }{ INT }
2 \proxy_push_member:nnn
3   {
4     \object_address:nn { mymod }{ INT }
5   }{ var }{ tl }

```

Container proxy

```

1 \proxy_create:nn { mymod }{ EXT }
2 \proxy_push_embedded:nnn
3   {
4     \object_address:nn { mymod }{ EXT }
5   }
6   { emb }
7   {
8     \object_address:nn { mymod }{ INT }
9   }

```

Now we create a new object from proxy EXT. It'll contain an embedded object created with INT proxy:

```

1 \str_new:N \g_EXTObj_str
2 \int_new:N \g_intcount_int
3 \object_gallocate_gincr:NNnnNN
4   \g_EXTObj_str \g_intcount_int
5   {
6     \object_address:nn { mymod }{ EXT }
7   }
8   { mymod }
9   \c_object_local_str \c_object_public_str

```

and use the embedded object in the following way:

```

1  \object_member_set:nnn
2  {
3    \object_embedded_adr:Vn \g_EXTObj_str { emb }
4  }{ var }{ Hi }
5  \object_member_use:nn
6  {
7    \object_embedded_adr:Vn \g_EXTObj_str { emb }
8  }{ var }

```

Output: Hi

## 8 Implementation

```

1  <*package>
2  <@@=rawobjects>
3  Deprecation message
4  \msg_new:nnn { rawobjects }{ deprecate }
5  {
6    Command ~ #1 ~ is ~ deprecated. ~ Use ~ instead ~ #2
7  }
8
9  \cs_new_protected:Nn \__rawobjects_launch_deprecate:NN
10 {
11   \msg_warning:nnnn{ rawobjects }{ deprecate }{ #1 }{ #2 }
12 }
13

```

`\rwoobj_address_f:n` It just performs a c expansion before passing it to `\cs_to_str:N`.

```

14
15 \cs_new:Nn \rwoobj_address_f:n
16 {
17   \exp_args:Nc \cs_to_str:N { #1 }
18 }
19

```

(End definition for `\rwoobj_address_f:n`. This function is documented on page 7.)

```

\c_object_local_str
\c_object_global_str
\c_object_public_str
\c_object_private_str
20 \str_const:Nn \c_object_local_str {l}
21 \str_const:Nn \c_object_global_str {g}
22 \str_const:Nn \c_object_public_str {_}
23 \str_const:Nn \c_object_private_str {__}
24
25
26 \cs_new:Nn \__rawobjects_scope:N
27 {
28   \str_use:N #1
29 }
30
31 \cs_new:Nn \__rawobjects_scope_pfx:N

```



```

32 {
33   \str_if_eq:NNF #1 \c_object_local_str
34   { g }
35 }
36
37 \cs_generate_variant:Nn \__rawobjects_scope_pfx:N { c }
38
39 \cs_new:Nn \__rawobjects_scope_pfx_cl:n
40 {
41   \__rawobjects_scope_pfx:c{
42     \object_ncmember_adr:nnn
43     {
44       \object_embedded_adr:nn { #1 } { /_I_/ }
45     }
46   { S } { str }
47 }
48 }
49
50 \cs_new:Nn \__rawobjects_vis_var:N
51 {
52   \str_use:N #1
53 }
54
55 \cs_new:Nn \__rawobjects_vis_fun:N
56 {
57   \str_if_eq:NNT #1 \c_object_private_str
58   {
59     --
60   }
61 }
62

```

(End definition for `\c_object_local_str` and others. These variables are documented on page 13.)

**\object\_address:nn** Get address of an object

```

63 \cs_new:Nn \object_address:nn {
64   \tl_to_str:n { #1 _ #2 }
65 }

```

(End definition for `\object_address:nn`. This function is documented on page 7.)

**\object\_embedded\_adr:nn** Address of embedded object

```

66
67 \cs_new:Nn \object_embedded_adr:nn
68 {
69   #1 \tl_to_str:n{ _SUB_ #2 }
70 }
71
72 \cs_generate_variant:Nn \object_embedded_adr:nn{ Vn }
73

```

(End definition for `\object_embedded_adr:nn`. This function is documented on page 7.)

`\object_address_set:Nnn` Saves the address of an object into a string variable

`\object_address_gset:Nnn`

```
74
75 \cs_new_protected:Nn \object_address_set:Nnn {
76   \str_set:Nn #1 { #2 _ #3 }
77 }
78
79 \cs_new_protected:Nn \object_address_gset:Nnn {
80   \str_gset:Nn #1 { #2 _ #3 }
81 }
82
```

*(End definition for \object\_address\_set:Nnn and \object\_address\_gset:Nnn. These functions are documented on page 7.)*

`\object_if_exist_p:n` Tests if object exists.

`\object_if_exist:nTF`

```
83
84 \prg_new_conditional:Nnn \object_if_exist:n { p, T, F, TF }
85 {
86   \cs_if_exist:cTF
87   {
88     \object_ncmember_adr:nnn
89     {
90       \object_embedded_adr:nn{ #1 }{ /_I_/ }
91     }
92     { S }{ str }
93   }
94   {
95     \prg_return_true:
96   }
97   {
98     \prg_return_false:
99   }
100 }
101
102 \prg_generate_conditional_variant:Nnn \object_if_exist:n { V }
103 { p, T, F, TF }
104
```

*(End definition for \object\_if\_exist:nTF. This function is documented on page 7.)*

`\object_get_module:n` Retrieve the name, module and generating proxy of an object

`\object_get_proxy_adr:n`

```
105 \cs_new:Nn \object_get_module:n {
106   \object_ncmember_use:nnn
107   {
108     \object_embedded_adr:nn{ #1 }{ /_I_/ }
109   }
110   { M }{ str }
111 }
112 \cs_new:Nn \object_get_proxy_adr:n {
113   \object_ncmember_use:nnn
114   {
115     \object_embedded_adr:nn{ #1 }{ /_I_/ }
116   }
117   { P }{ str }
```

```

118 }
119
120 \cs_generate_variant:Nn \object_get_module:n { V }
121 \cs_generate_variant:Nn \object_get_proxy_adr:n { V }

```

(End definition for `\object_get_module:n` and `\object_get_proxy_adr:n`. These functions are documented on page 7.)

```

\object_if_local_p:n Test the specified parameters.
\object_if_local:nTF
\object_if_global_p:n
\object_if_global:nTF
\object_if_public_p:n
\object_if_public:nTF
\object_if_private_p:n
\object_if_private:nTF
122 \prg_new_conditional:Nnn \object_if_local:n {p, T, F, TF}
123 {
124   \str_if_eq:cNTF
125   {
126     \object_ncmember_adr:nnn
127     {
128       \object_embedded_adr:nn{ #1 }{ /_I_/ }
129     }
130     { S }{ str }
131   }
132   \c_object_local_str
133   {
134     \prg_return_true:
135   }
136   {
137     \prg_return_false:
138   }
139 }
140
141 \prg_new_conditional:Nnn \object_if_global:n {p, T, F, TF}
142 {
143   \str_if_eq:cNTF
144   {
145     \object_ncmember_adr:nnn
146     {
147       \object_embedded_adr:nn{ #1 }{ /_I_/ }
148     }
149     { S }{ str }
150   }
151   \c_object_global_str
152   {
153     \prg_return_true:
154   }
155   {
156     \prg_return_false:
157   }
158 }
159
160 \prg_new_conditional:Nnn \object_if_public:n {p, T, F, TF}
161 {
162   \str_if_eq:cNTF
163   {
164     \object_ncmember_adr:nnn
165     {
166       \object_embedded_adr:nn{ #1 }{ /_I_/ }

```

```

167     }
168     { V }{ str }
169 }
170 \c_object_public_str
171 {
172     \prg_return_true:
173 }
174 {
175     \prg_return_false:
176 }
177 }
178
179 \prg_new_conditional:Nnn \object_if_private:n {p, T, F, TF}
180 {
181     \str_if_eq:cNTF
182     {
183         \object_ncmember_adr:nnn
184         {
185             \object_embedded_adr:nn{ #1 }{ /_I_/ }
186         }
187         { V }{ str }
188     }
189     \c_object_private_str
190     {
191         \prg_return_true:
192     }
193     {
194         \prg_return_false:
195     }
196 }
197
198 \prg_generate_conditional_variant:Nnn \object_if_local:n { V }
199 { p, T, F, TF }
200 \prg_generate_conditional_variant:Nnn \object_if_global:n { V }
201 { p, T, F, TF }
202 \prg_generate_conditional_variant:Nnn \object_if_public:n { V }
203 { p, T, F, TF }
204 \prg_generate_conditional_variant:Nnn \object_if_private:n { V }
205 { p, T, F, TF }

```

(End definition for `\object_if_local:nTF` and others. These functions are documented on page 7.)

`\object_macro_adr:nn` Generic macro address

`\object_macro_use:nn`

```

206
207 \cs_new:Nn \object_macro_adr:nn
208 {
209     #1 \tl_to_str:n{ _MACRO_ #2 }
210 }
211
212 \cs_generate_variant:Nn \object_macro_adr:nn{ Vn }
213
214 \cs_new:Nn \object_macro_use:nn
215 {
216     \use:c

```

```

217     {
218         \object_macro_adr:nn{ #1 }{ #2 }
219     }
220 }
221
222 \cs_generate_variant:Nn \object_macro_use:nn{ Vn }
223

```

(End definition for \object\_macro\_adr:nn and \object\_macro\_use:nn. These functions are documented on page 11.)

\\_rawobjects\_member\_adr:nnnNN Macro address without object inference

```

224
225 \cs_new:Nn \_rawobjects_member_adr:nnnNN
226 {
227     \_rawobjects_scope:N #4
228     \_rawobjects_vis_var:N #5
229     #1 \tl_to_str:n { _ MEMBER _ #2 _ #3 }
230 }
231
232 \cs_generate_variant:Nn \_rawobjects_member_adr:nnnNN { VnnNN, nnncc }
233

```

(End definition for \\_rawobjects\_member\_adr:nnnNN.)

\object\_member\_adr:nnn Get the address of a member variable  
\object\_member\_adr:nn

```

234
235 \cs_new:Nn \object_member_adr:nnn
236 {
237     \_rawobjects_member_adr:nnncc { #1 }{ #2 }{ #3 }
238     {
239         \object_ncmember_adr:nnn
240         {
241             \object_embedded_adr:nn{ #1 }{ /_I_/ }
242         }
243         { S }{ str }
244     }
245     {
246         \object_ncmember_adr:nnn
247         {
248             \object_embedded_adr:nn{ #1 }{ /_I_/ }
249         }
250         { V }{ str }
251     }
252 }
253
254 \cs_generate_variant:Nn \object_member_adr:nnn { Vnn, vnn, nnv }
255
256 \cs_new:Nn \object_member_adr:nn
257 {
258     \object_member_adr:nnv { #1 }{ #2 }
259     {
260         \object_rcmember_adr:nnn { #1 }
261         { #2 _ type }{ str }
262     }

```

```

263 }
264
265 \cs_generate_variant:Nn \object_member_adr:nn { Vn }
266

```

(End definition for \object\_member\_adr:nnn and \object\_member\_adr:nn. These functions are documented on page 8.)

```

\object_member_if_exist_p:nnn Tests if the specified member exists
\object_member_if_exist:nnnTF
267
268 \prg_new_conditional:Nnn \object_member_if_exist:nnn {p, T, F, TF }
269 {
270   \cs_if_exist:cTF
271   {
272     \object_member_adr:nnn { #1 }{ #2 }{ #3 }
273   }
274   {
275     \prg_return_true:
276   }
277   {
278     \prg_return_false:
279   }
280 }
281
282 \prg_generate_conditional_variant:Nnn \object_member_if_exist:nnn
283 { Vnn }{ p, T, F, TF }
284

```

(End definition for \object\_member\_if\_exist:nnnTF. This function is documented on page 8.)

```

\object_member_if_tracked_p:nn Tests if the member is tracked.
\object_member_if_tracked:nnTF
285
286 \prg_new_conditional:Nnn \object_member_if_tracked:nn {p, T, F, TF }
287 {
288   \cs_if_exist:cTF
289   {
290     \object_rcmember_adr:nnn
291     { #1 }{ #2 _ type }{ str }
292   }
293   {
294     \prg_return_true:
295   }
296   {
297     \cs_if_exist:cTF
298     {
299       \object_ncmember_adr:nnn
300       {
301         \object_embedded_adr:nn { #1 }{ /_T_/ }
302       }
303       { #2 _ type }{ str }
304     }
305     {
306       \prg_return_true:
307     }
308     {

```

```

309         \prg_return_false:
310     }
311 }
312 }
313
314 \prg_generate_conditional_variant:Nnn \object_member_if_tracked:nn { Vn }{ p, T, F, TF }
315
316 \prg_new_eq_conditional:NNn \object_member_if_exist:nn
317   \object_member_if_tracked:nn { p, T, F, TF }
318 \prg_new_eq_conditional:NNn \object_member_if_exist:Vn
319   \object_member_if_tracked:Vn { p, T, F, TF }
320

```

(End definition for \object\_member\_if\_tracked:nnTF. This function is documented on page 8.)

**\object\_member\_type:nn** Deduce the type of tracked members.

```

321
322 \cs_new:Nn \object_member_type:nn
323 {
324   \cs_if_exist:cTF
325   {
326     \object_rcmember_adr:nnn
327     { #1 }{ #2 _ type }{ str }
328   }
329   {
330     \object_rcmember_use:nnn
331     { #1 }{ #2 _ type }{ str }
332   }
333   {
334     \cs_if_exist:cT
335     {
336       \object_ncmember_adr:nnn
337       {
338         \object_embedded_adr:nn { #1 }{ /_T_/ }
339       }
340       { #2 _ type }{ str }
341     }
342     {
343       \object_ncmember_use:nnn
344       {
345         \object_embedded_adr:nn { #1 }{ /_T_/ }
346       }
347       { #2 _ type }{ str }
348     }
349   }
350 }
351

```

(End definition for \object\_member\_type:nn. This function is documented on page 8.)

The first argument is the new function name without argument. The second one is the function name you'll use, here #1 is the member type and #2 is equal to **g** if the object is global. The third one are the argument of the second function without the first N.

352

```

353 \cs_new_protected:Nn \__rawobjects_generator_mem:nnn
354 {
355   \cs_new:cn
356   {
357     rwobj-aux_ #1 : nn
358   }
359   {
360     \use:c{ #2 : c #3 }
361   }
362   \cs_new:cpn {#1 : nnn #3} ##1##2##3
363   {
364     \use:c
365     {
366       rwobj-aux_ #1 : nn
367     }
368     { ##3 }
369     {
370       \__rawobjects_scope_pfx_cl:n{ ##1 }
371     }
372     {
373       \object_member_adr:nnn{ ##1 }{ ##2 }{ ##3 }
374     }
375   }
376   \cs_generate_variant:cn { #1 : nnn #3 }{ Vnn #3, nnv #3 }
377
378   \cs_new:cpn { #1 : nn #3 } ##1##2
379   {
380     \use:c{ #1 : nnv #3 }
381     { ##1 }{ ##2 }
382     {
383       \object_rcmember_adr:nnn
384       { ##1 }{ ##2 _ type }{ str }
385     }
386   }
387
388   \cs_generate_variant:cn { #1 : nn #3 }{ Vn #3 }
389 }
390
391
392 \msg_new:nnn{ rawobjects }{ nonew }{ Unknown ~ function ~ #1 }
393
394 \cs_new_protected:Nn \__rawobjects_generator_mem_protected:nnn
395 {
396   \cs_new_protected:cn
397   {
398     rwobj-aux_ #1 : nn
399   }
400   {
401     \cs_if_exist_use:cF{ #2 : c #3 }
402     {
403       \msg_error:nnx{ rawobjects }{ nonew }{ #2 :c #3 }
404     }
405   }
406   \cs_new_protected:cpn {#1 : nnn #3} ##1##2##3

```



```

407     {
408         \use:c
409         {
410             rwojb-aux_ #1 : nn
411         }
412         { ##3 }
413         {
414             \__rawobjects_scope_pfx_cl:n{ ##1 }
415         }
416         {
417             \object_member_adr:nnn{ ##1 }{ ##2 }{ ##3 }
418         }
419     }
420     \cs_generate_variant:cn { #1 : nnn #3 }{ Vnn #3, nnv #3 }
421
422     \cs_new_protected:cpn { #1 : nn #3 } ##1##2
423     {
424         \use:c{ #1 : nnv #3 }
425         { ##1 }{ ##2 }
426         {
427             \object_rcmember_adr:nnn
428             { ##1 }{ ##2 _ type }{ str }
429         }
430     }
431
432     \cs_generate_variant:cn { #1 : nn #3 }{ Vn #3 }
433 }
434
435
436 \msg_new:nnnn { rawobjects }{ noerr }{ Unspecified ~ scope }
437 {
438     Object ~ #1 ~ hasn't ~ a ~ scope ~ variable
439 }
440
441 \msg_new:nnnn { rawobjects }{ scoperr }{ Nonstandard ~ scope }
442 {
443     Operation ~ not ~ permitted ~ on ~ object ~ #1 ~
444     ~ since ~ it ~ wasn't ~ declared ~ local ~ or ~ global
445 }
446
447 \cs_new_protected:Nn \__rawobjects_force_scope:n
448 {
449     \cs_if_exist:cTF
450     {
451         \object_ncmember_adr:nnn
452         {
453             \object_embedded_adr:nn{ #1 }{ /_I_/ }
454         }
455         { S }{ str }
456     }
457     {
458         \bool_if:nF
459         {
460             \object_if_local_p:n { #1 } || \object_if_global_p:n { #1 }

```

```

461         }
462         {
463             \msg_error:nxx { rawobjects }{ scoperr }{ #1 }
464         }
465     }
466     {
467         \msg_error:nxx { rawobjects }{ noerr }{ #1 }
468     }
469 }
470

```

**\object\_new\_member:nnn** Creates a new member variable

```

471
472
473 \cs_new_protected:Nn \object_new_member:nnn
474 {
475     \cs_if_exist_use:cT { #3 _ new:c }
476     {
477         { \object_member_adr:nnn { #1 }{ #2 }{ #3 } }
478     }
479 }
480
481 \cs_generate_variant:Nn \object_new_member:nnn { Vnn, nnv }
482

```

(End definition for \object\_new\_member:nnn. This function is documented on page 8.)

**\object\_member\_use:nnn** Uses a member variable

**\object\_member\_use:nn**

```

483
484 \__rawobjects_generator_mem:nnn {object_member_use}{ #1_use }{ }
485
486 \cs_generate_variant:Nn \object_member_use:nnn {vnn}
487

```

(End definition for \object\_member\_use:nnn and \object\_member\_use:nn. These functions are documented on page 8.)

**\object\_member\_set:nnnn** Set the value a member.

**\object\_member\_set:nnn**

```

488
489 \__rawobjects_generator_mem:nnn {object_member_set}{ #1_#2 set }{n}
490

```

(End definition for \object\_member\_set:nnnn and \object\_member\_set:nnn. These functions are documented on page 9.)

**\object\_member\_set\_eq:nnnN** Make a member equal to another variable.

**\object\_member\_set\_eq:nnN**

```

491
492 \__rawobjects_generator_mem_protected:nnn { object_member_set_eq }{ #1 _ #2 set_eq }{ N }
493
494 \cs_generate_variant:Nn \object_member_set_eq:nnnN { nnnc, Vnnnc }
495
496 \cs_generate_variant:Nn \object_member_set_eq:nnN { nnc, Vnc }
497

```

(End definition for \object\_member\_set\_eq:nnnN and \object\_member\_set\_eq:nnN. These functions are documented on page 9.)

`\object_ncmember_adr:nnn` Get address of near constant

```
498
499 \cs_new:Nn \object_ncmember_adr:nnn
500 {
501   \tl_to_str:n{ c _ } #1 \tl_to_str:n { _ CONST _ #2 _ #3 }
502 }
503
504 \cs_generate_variant:Nn \object_ncmember_adr:nnn { Vnn, vnn }
505
```

(End definition for `\object_ncmember_adr:nnn`. This function is documented on page 9.)

`\object_rcmember_adr:nnn` Get the address of a remote constant.

```
506
507 \cs_new:Nn \object_rcmember_adr:nnn
508 {
509   \object_ncmember_adr:vnn
510   {
511     \object_ncmember_adr:nnn
512     {
513       \object_embedded_adr:nn{ #1 }{ /_I_/ }
514     }
515     { P }{ str }
516   }
517   { #2 }{ #3 }
518 }
519
520 \cs_generate_variant:Nn \object_rcmember_adr:nnn { Vnn }
```

(End definition for `\object_rcmember_adr:nnn`. This function is documented on page 9.)

The first argument is the new function name without argument. The second one is the function name you'll use, here #1 is the constant type. The third one are the argument of the second function without the first N.

```
521
522 \cs_new_protected:Nn \__rawobjects_generator_ncmem:nnn
523 {
524   \cs_new:cn
525   {
526     rwojb-aux_ #1 : n
527   }
528   {
529     \use:c{ #2 : c #3 }
530   }
531   \cs_new:cpn {#1 : nnn #3} ##1##2##3
532   {
533     \use:c
534     {
535       rwojb-aux_ #1 : n
536     }
537     { ##3 }
538     {
539       \object_ncmember_adr:nnn{ ##1 }{ ##2 }{ ##3 }
540     }
541   }
```

```

542     \cs_generate_variant:cn { #1 : nnn #3 }{ Vnn #3 }
543 }
544
545 \cs_new_protected:Nn \__rawobjects_generator_ncmem_protected:nnn
546 {
547     \cs_new_protected:cn
548     {
549         rwojb-aux_ #1 : n
550     }
551     {
552         \cs_if_exist_use:cF{ #2 : c #3 }
553         {
554             \msg_error:nnx{ rawobjects }{ noneew }{ #2 :c #3 }
555         }
556     }
557     \cs_new_protected:cpn {#1 : nnn #3} ##1##2##3
558     {
559         \use:c
560         {
561             rwojb-aux_ #1 : n
562         }
563         { ##3 }
564         {
565             \object_ncmember_adr:nnn{ ##1 }{ ##2 }{ ##3 }
566         }
567     }
568     \cs_generate_variant:cn { #1 : nnn #3 }{ Vnn #3 }
569 }
570

```

`\object_ncmember_if_exist_p:nnn` Tests if the specified member constant exists.

`\object_ncmember_if_exist:nnn`*TF*

`\object_rcmember_if_exist_p:nnn`

`\object_rcmember_if_exist:nnn`*TF*

```

571
572 \prg_new_conditional:Nnn \object_ncmember_if_exist:nnn {p, T, F, TF }
573 {
574     \cs_if_exist:cTF
575     {
576         \object_ncmember_adr:nnn { #1 }{ #2 }{ #3 }
577     }
578     {
579         \prg_return_true:
580     }
581     {
582         \prg_return_false:
583     }
584 }
585
586 \prg_new_conditional:Nnn \object_rcmember_if_exist:nnn {p, T, F, TF }
587 {
588     \cs_if_exist:cTF
589     {
590         \object_rcmember_adr:nnn { #1 }{ #2 }{ #3 }
591     }
592     {
593         \prg_return_true:

```

```

594     }
595     {
596         \prg_return_false:
597     }
598 }
599
600 \prg_generate_conditional_variant:Nnn \object_ncmember_if_exist:nnn
601 { Vnn }{ p, T, F, TF }
602 \prg_generate_conditional_variant:Nnn \object_rcmember_if_exist:nnn
603 { Vnn }{ p, T, F, TF }
604

```

(End definition for \object\_ncmember\_if\_exist:nnnTF and \object\_rcmember\_if\_exist:nnnTF. These functions are documented on page 9.)

**\object\_ncmember\_use:nnn** Uses a near/remote constant.

```

\object_rcmember_use:nnn
605
606 \__rawobjects_generator_ncmem:nnn{ object_ncmember_use }{ #1_use }{ }
607
608 \cs_new:Nn \object_rcmember_use:nnn
609 {
610     \cs_if_exist_use:cT { #3 _ use:c }
611     {
612         { \object_rcmember_adr:nnn { #1 }{ #2 }{ #3 } }
613     }
614 }
615
616 \cs_generate_variant:Nn \object_rcmember_use:nnn { Vnn }
617

```

(End definition for \object\_ncmember\_use:nnn and \object\_rcmember\_use:nnn. These functions are documented on page 9.)

**\object\_newconst:nnnn** Creates a constant variable, use with caution

```

618
619 \__rawobjects_generator_ncmem_protected:nnn { object_newconst }{ #1 _ const }{ n }
620

```

(End definition for \object\_newconst:nnnn. This function is documented on page 11.)

**\object\_newconst\_tl:nnn** Create constants

**\object\_newconst\_str:nnn**

**\object\_newconst\_int:nnn**

**\object\_newconst\_clist:nnn**

**\object\_newconst\_dim:nnn**

**\object\_newconst\_skip:nnn**

**\object\_newconst\_fp:nnn**

```

621
622 \cs_new_protected:Nn \object_newconst_tl:nnn
623 {
624     \object_newconst:nnnn { #1 }{ #2 }{ tl }{ #3 }
625 }
626 \cs_new_protected:Nn \object_newconst_str:nnn
627 {
628     \object_newconst:nnnn { #1 }{ #2 }{ str }{ #3 }
629 }
630 \cs_new_protected:Nn \object_newconst_int:nnn
631 {
632     \object_newconst:nnnn { #1 }{ #2 }{ int }{ #3 }
633 }
634 \cs_new_protected:Nn \object_newconst_clist:nnn

```

```

635 {
636   \object_newconst:nnnn { #1 }{ #2 }{ clist }{ #3 }
637 }
638 \cs_new_protected:Nn \object_newconst_dim:nnn
639 {
640   \object_newconst:nnnn { #1 }{ #2 }{ dim }{ #3 }
641 }
642 \cs_new_protected:Nn \object_newconst_skip:nnn
643 {
644   \object_newconst:nnnn { #1 }{ #2 }{ skip }{ #3 }
645 }
646 \cs_new_protected:Nn \object_newconst_fp:nnn
647 {
648   \object_newconst:nnnn { #1 }{ #2 }{ fp }{ #3 }
649 }
650
651 \cs_generate_variant:Nn \object_newconst_tl:nnn { Vnn }
652 \cs_generate_variant:Nn \object_newconst_str:nnn { Vnn }
653 \cs_generate_variant:Nn \object_newconst_int:nnn { Vnn }
654 \cs_generate_variant:Nn \object_newconst_clist:nnn { Vnn }
655 \cs_generate_variant:Nn \object_newconst_dim:nnn { Vnn }
656 \cs_generate_variant:Nn \object_newconst_skip:nnn { Vnn }
657 \cs_generate_variant:Nn \object_newconst_fp:nnn { Vnn }
658
659
660 \cs_generate_variant:Nn \object_newconst_str:nnn { nnx }
661 \cs_generate_variant:Nn \object_newconst_str:nnn { nnV }
662

```

(End definition for `\object_newconst_tl:nnn` and others. These functions are documented on page 11.)

`\object_newconst_seq_from_clist:nnn` Creates a `seq` constant.

```

663
664 \cs_new_protected:Nn \object_newconst_seq_from_clist:nnn
665 {
666   \seq_const_from_clist:cn
667   {
668     \object_ncmember_adr:nnn { #1 }{ #2 }{ seq }
669   }
670   { #3 }
671 }
672
673 \cs_generate_variant:Nn \object_newconst_seq_from_clist:nnn { Vnn }
674

```

(End definition for `\object_newconst_seq_from_clist:nnn`. This function is documented on page 11.)

`\object_newconst_prop_from_keyval:nnn` Creates a `prop` constant.

```

675
676 \cs_new_protected:Nn \object_newconst_prop_from_keyval:nnn
677 {
678   \prop_const_from_keyval:cn
679   {
680     \object_ncmember_adr:nnn { #1 }{ #2 }{ prop }
681   }

```

```

682     { #3 }
683   }
684
685   \cs_generate_variant:Nn \object_newconst_prop_from_keyval:nnn { Vnn }
686

```

(End definition for \object\_newconst\_prop\_from\_keyval:nnn. This function is documented on page 11.)

\object\_ncmethod\_adr:nnn Fully expands to the method address.

\object\_rcmethod\_adr:nnn

```

687
688   \cs_new:Nn \object_ncmethod_adr:nnn
689   {
690     #1 \tl_to_str:n { _ CMETHOD _ #2 : #3 }
691   }
692
693   \cs_generate_variant:Nn \object_ncmethod_adr:nnn { Vnn , vnn }
694
695   \cs_new:Nn \object_rcmethod_adr:nnn
696   {
697     \object_ncmethod_adr:vnn
698     {
699       \object_ncmember_adr:nnn
700       {
701         \object_embedded_adr:nn{ #1 }{ /_I/ }
702       }
703       { P }{ str }
704     }
705     { #2 }{ #3 }
706   }
707
708   \cs_generate_variant:Nn \object_ncmethod_adr:nnn { Vnn , vnn }
709   \cs_generate_variant:Nn \object_rcmethod_adr:nnn { Vnn }
710

```

(End definition for \object\_ncmethod\_adr:nnn and \object\_rcmethod\_adr:nnn. These functions are documented on page 10.)

\object\_ncmethod\_if\_exist\_p:nnn Tests if the specified member constant exists.

\object\_ncmethod\_if\_exist:nnn $\overline{TF}$

\object\_rcmethod\_if\_exist\_p:nnn

\object\_rcmethod\_if\_exist:nnn $\overline{TF}$

```

711
712   \prg_new_conditional:Nnn \object_ncmethod_if_exist:nnn {p, T, F, TF }
713   {
714     \cs_if_exist:cTF
715     {
716       \object_ncmethod_adr:nnn { #1 }{ #2 }{ #3 }
717     }
718     {
719       \prg_return_true:
720     }
721     {
722       \prg_return_false:
723     }
724   }
725
726   \prg_new_conditional:Nnn \object_rcmethod_if_exist:nnn {p, T, F, TF }
727   {

```

```

728     \cs_if_exist:cTF
729     {
730         \object_rcmethodr_adr:nnn { #1 }{ #2 }{ #3 }
731     }
732     {
733         \prg_return_true:
734     }
735     {
736         \prg_return_false:
737     }
738 }
739
740 \prg_generate_conditional_variant:Nnn \object_ncmethod_if_exist:nnn
741 { Vnn }{ p, T, F, TF }
742 \prg_generate_conditional_variant:Nnn \object_rcmethod_if_exist:nnn
743 { Vnn }{ p, T, F, TF }
744

```

(End definition for `\object_ncmethod_if_exist:nnnTF` and `\object_rcmethod_if_exist:nnnTF`. These functions are documented on page 10.)

`\object_new_cmethod:nnnn` Creates a new method

```

745
746 \cs_new_protected:Nn \object_new_cmethod:nnnn
747 {
748     \cs_new:cn
749     {
750         \object_ncmethod_adr:nnn { #1 }{ #2 }{ #3 }
751     }
752     { #4 }
753 }
754
755 \cs_generate_variant:Nn \object_new_cmethod:nnnn { Vnnn }
756

```

(End definition for `\object_new_cmethod:nnnn`. This function is documented on page 10.)

`\object_ncmethod_call:nnn` Calls the specified method.

`\object_rcmethod_call:nnn`

```

757
758 \cs_new:Nn \object_ncmethod_call:nnn
759 {
760     \use:c
761     {
762         \object_ncmethod_adr:nnn { #1 }{ #2 }{ #3 }
763     }
764 }
765
766 \cs_new:Nn \object_rcmethod_call:nnn
767 {
768     \use:c
769     {
770         \object_rcmethod_adr:nnn { #1 }{ #2 }{ #3 }
771     }
772 }
773

```



```

774 \cs_generate_variant:Nn \object_ncmethod_call:nnn { Vnn }
775 \cs_generate_variant:Nn \object_rcmethod_call:nnn { Vnn }
776

```

(End definition for `\object_ncmethod_call:nnn` and `\object_rcmethod_call:nnn`. These functions are documented on page 10.)

```

777
778 \cs_new_protected:Nn \__rawobjects_initproxy:nnn
779 {
780   \object_newconst:nnnn
781   {
782     \object_embedded_adr:nn{ #3 }{ /_I_/ }
783   }
784   { ifprox }{ bool }{ \c_true_bool }
785 }
786 \cs_generate_variant:Nn \__rawobjects_initproxy:nnn { VnV }
787

```

`\object_if_proxy_p:n` Test if an object is a proxy.

`\object_if_proxy:nTF`

```

788
789 \cs_new:Nn \__rawobjects_bol_com:N
790 {
791   \cs_if_exist_p:N #1 && \bool_if_p:N #1
792 }
793
794 \cs_generate_variant:Nn \__rawobjects_bol_com:N { c }
795
796 \prg_new_conditional:Nnn \object_if_proxy:n {p, T, F, TF}
797 {
798   \cs_if_exist:cTF
799   {
800     \object_ncmember_adr:nnn
801     {
802       \object_embedded_adr:nn{ #1 }{ /_I_/ }
803     }
804     { ifprox }{ bool }
805   }
806   {
807     \bool_if:cTF
808     {
809       \object_ncmember_adr:nnn
810       {
811         \object_embedded_adr:nn{ #1 }{ /_I_/ }
812       }
813       { ifprox }{ bool }
814     }
815     {
816       \prg_return_true:
817     }
818     {
819       \prg_return_false:
820     }
821   }
822 }

```

```

823         \prg_return_false:
824     }
825 }
826

```

(End definition for \object\_if\_proxy:nTF. This function is documented on page 12.)

\object\_test\_proxy\_p:nn  
\object\_test\_proxy:nnTF  
\object\_test\_proxy\_p:nN  
\object\_test\_proxy:nNTF

Test if an object is generated from selected proxy.

```

827
828 \prg_generate_conditional_variant:Nnn \str_if_eq:nn { ve }{ TF }
829
830 \prg_new_conditional:Nnn \object_test_proxy:nn {p, T, F, TF}
831 {
832     \str_if_eq:veTF
833     {
834         \object_ncmember_adr:nnn
835         {
836             \object_embedded_adr:nn{ #1 }{ /_I_/ }
837         }
838         { P }{ str }
839     }
840     { #2 }
841     {
842         \prg_return_true:
843     }
844     {
845         \prg_return_false:
846     }
847 }
848
849 \prg_new_conditional:Nnn \object_test_proxy:nN {p, T, F, TF}
850 {
851     \str_if_eq:cNTF
852     {
853         \object_ncmember_adr:nnn
854         {
855             \object_embedded_adr:nn{ #1 }{ /_I_/ }
856         }
857         { P }{ str }
858     }
859     #2
860     {
861         \prg_return_true:
862     }
863     {
864         \prg_return_false:
865     }
866 }
867
868 \prg_generate_conditional_variant:Nnn \object_test_proxy:nn
869 { Vn }{p, T, F, TF}
870 \prg_generate_conditional_variant:Nnn \object_test_proxy:nN
871 { VN }{p, T, F, TF}
872

```

(End definition for `\object_test_proxy:nTF` and `\object_test_proxy:nNTF`. These functions are documented on page 12.)

Creates an object from a proxy.

```

\object_create:nnnNN
\object_create_set:NnnnNN
\object_create_gset:NnnnNN
\object_create:nnnN
\object_create_set:NnnnN
\object_create_gset:NnnnN
\object_create:nnn
\object_create_set:Nnnn
\object_create_gset:Nnnn
\embedded_create:nnn
873
874 \msg_new:nnnn { rawobjects }{ notproxy }{ Fake ~ proxy }
875 {
876   Object ~ #1 ~ is ~ not ~ a ~ proxy.
877 }
878
879 \cs_new_protected:Nn \__rawobjects_force_proxy:n
880 {
881   \object_if_proxy:nF { #1 }
882   {
883     \msg_error:nnn { rawobjects }{ notproxy }{ #1 }
884   }
885 }
886
887 \cs_new_protected:Nn \__rawobjects_create_anon:nnnNN
888 {
889   \tl_if_empty:nF{ #1 }
890   {
891
892     \__rawobjects_force_proxy:n { #1 }
893
894
895     \object_newconst_str:nnn
896     {
897       \object_embedded_adr:nn{ #3 }{ /_I_/ }
898     }
899     { M }{ #2 }
900     \object_newconst_str:nnn
901     {
902       \object_embedded_adr:nn{ #3 }{ /_I_/ }
903     }
904     { P }{ #1 }
905     \object_newconst_str:nnV
906     {
907       \object_embedded_adr:nn{ #3 }{ /_I_/ }
908     }
909     { S } #4
910     \object_newconst_str:nnV
911     {
912       \object_embedded_adr:nn{ #3 }{ /_I_/ }
913     }
914     { V } #5
915
916     \seq_map_inline:cn
917     {
918       \object_member_adr:nnn { #1 }{ varlist }{ seq }
919     }
920     {
921       \object_new_member:nnv { #3 }{ ##1 }
922       {

```

```

923         \object_ncmember_adr:nnn { #1 }{ ##1 _ type }{ str }
924     }
925 }
926
927 \seq_map_inline:cn
928 {
929     \object_member_adr:nnn { #1 }{ objlist }{ seq }
930 }
931 {
932     \embedded_create:nvn
933     { #3 }
934     {
935         \object_ncmember_adr:nnn { #1 }{ ##1 _ proxy }{ str }
936     }
937     { ##1 }
938 }
939
940 \tl_map_inline:cn
941 {
942     \object_member_adr:nnn { #1 }{ init }{ tl }
943 }
944 {
945     ##1 { #1 }{ #2 }{ #3 }
946 }
947
948 }
949 }
950
951 \cs_generate_variant:Nn \__rawobjects_create_anon:nnnNN { xnxNN, xvxc }
952
953 \cs_new_protected:Nn \object_create:nnnNN
954 {
955     \__rawobjects_create_anon:xnxNN { #1 }{ #2 }
956     { \object_address:nn { #2 }{ #3 } }
957     #4 #5
958 }
959
960 \cs_generate_variant:Nn \object_create:nnnNN { VnnNN }
961
962 \cs_new_protected:Nn \object_create_set:NnnnNN
963 {
964     \object_create:nnnNN { #2 }{ #3 }{ #4 } #5 #6
965     \str_set:Nx #1 { \object_address:nn { #3 }{ #4 } }
966 }
967
968 \cs_new_protected:Nn \object_create_gset:NnnnNN
969 {
970     \object_create:nnnNN { #2 }{ #3 }{ #4 } #5 #6
971     \str_gset:Nx #1 { \object_address:nn { #3 }{ #4 } }
972 }
973
974 \cs_generate_variant:Nn \object_create_set:NnnnNN { NVnnNN, NnnfNN }
975 \cs_generate_variant:Nn \object_create_gset:NnnnNN { NVnnNN, NnnfNN }
976

```

```

977
978
979 \cs_new_protected:Nn \object_create:nnnN
980 {
981   \object_create:nnnNN { #1 }{ #2 }{ #3 } #4 \c_object_public_str
982 }
983
984 \cs_generate_variant:Nn \object_create:nnnN { VnnN }
985
986 \cs_new_protected:Nn \object_create_set:NnnnN
987 {
988   \object_create_set:NnnnNN #1 { #2 }{ #3 }{ #4 } #5 \c_object_public_str
989 }
990
991 \cs_new_protected:Nn \object_create_gset:NnnnN
992 {
993   \object_create_gset:NnnnNN #1 { #2 }{ #3 }{ #4 } #5 \c_object_public_str
994 }
995
996 \cs_generate_variant:Nn \object_create_set:NnnnN { NVnnN }
997 \cs_generate_variant:Nn \object_create_gset:NnnnN { NVnnN }
998
999 \cs_new_protected:Nn \object_create:nnn
1000 {
1001   \object_create:nnnNN { #1 }{ #2 }{ #3 }
1002   \c_object_global_str \c_object_public_str
1003 }
1004
1005 \cs_generate_variant:Nn \object_create:nnn { Vnn }
1006
1007 \cs_new_protected:Nn \object_create_set:Nnnn
1008 {
1009   \object_create_set:NnnnNN #1 { #2 }{ #3 }{ #4 }
1010   \c_object_global_str \c_object_public_str
1011 }
1012
1013 \cs_new_protected:Nn \object_create_gset:Nnnn
1014 {
1015   \object_create_gset:NnnnNN #1 { #2 }{ #3 }{ #4 }
1016   \c_object_global_str \c_object_public_str
1017 }
1018
1019 \cs_generate_variant:Nn \object_create_set:Nnnn { NVnn }
1020 \cs_generate_variant:Nn \object_create_gset:Nnnn { NVnn }
1021
1022
1023
1024
1025 \cs_new_protected:Nn \embedded_create:nnn
1026 {
1027   \__rawobjects_create_anon:xvxc { #2 }
1028   {
1029     \object_ncmember_adr:nnn
1030     {

```

```

1031         \object_embedded_adr:nn{ #1 }{ /_I_/ }
1032     }
1033     { M }{ str }
1034 }
1035 {
1036     \object_embedded_adr:nn
1037     { #1 }{ #3 }
1038 }
1039 {
1040     \object_ncmember_adr:nnn
1041     {
1042         \object_embedded_adr:nn{ #1 }{ /_I_/ }
1043     }
1044     { S }{ str }
1045 }
1046 {
1047     \object_ncmember_adr:nnn
1048     {
1049         \object_embedded_adr:nn{ #1 }{ /_I_/ }
1050     }
1051     { V }{ str }
1052 }
1053 }
1054
1055 \cs_generate_variant:Nn \embedded_create:nnn { nvn, Vnn }
1056

```

(End definition for `\object_create:nnnNN` and others. These functions are documented on page 12.)

```

\proxy_create:nn    Creates a new proxy object
\proxy_create_set:Nnn
\proxy_create_gset:Nnn
1057
1058 \cs_new_protected:Nn \proxy_create:nn
1059 {
1060     \object_create:VnnNN \c_proxy_address_str { #1 }{ #2 }
1061     \c_object_global_str \c_object_public_str
1062 }
1063
1064 \cs_new_protected:Nn \proxy_create_set:Nnn
1065 {
1066     \object_create_set:NVnnNN #1 \c_proxy_address_str { #2 }{ #3 }
1067     \c_object_global_str \c_object_public_str
1068 }
1069
1070 \cs_new_protected:Nn \proxy_create_gset:Nnn
1071 {
1072     \object_create_gset:NVnnNN #1 \c_proxy_address_str { #2 }{ #3 }
1073     \c_object_global_str \c_object_public_str
1074 }
1075
1076
1077
1078 \cs_new_protected:Nn \proxy_create:nnN
1079 {
1080     \__rawobjects_launch_deprecate:NN \proxy_create:nnN \proxy_create:nn

```

```

1081     \object_create:VnnNN \c_proxy_address_str { #1 }{ #2 }
1082     \c_object_global_str #3
1083 }
1084
1085 \cs_new_protected:Nn \proxy_create_set:NnnN
1086 {
1087     \__rawobjects_launch_deprecate:NN \proxy_create_set:NnnN \proxy_create_set:Nnn
1088     \object_create_set:NVnnNN #1 \c_proxy_address_str { #2 }{ #3 }
1089     \c_object_global_str #4
1090 }
1091
1092 \cs_new_protected:Nn \proxy_create_gset:NnnN
1093 {
1094     \__rawobjects_launch_deprecate:NN \proxy_create_gset:NnnN \proxy_create_gset:Nnn
1095     \object_create_gset:NVnnNN #1 \c_proxy_address_str { #2 }{ #3 }
1096     \c_object_global_str #4
1097 }
1098

```

(End definition for `\proxy_create:nn`, `\proxy_create_set:Nnn`, and `\proxy_create_gset:Nnn`. These functions are documented on page 13.)

**`\proxy_push_member:nnn`** Push a new member inside a proxy.

```

1099
1100 \cs_new_protected:Nn \proxy_push_member:nnn
1101 {
1102     \object_newconst_str:nnn { #1 }{ #2 _ type }{ #3 }
1103     \seq_gput_left:cn
1104     {
1105         \object_member_adr:nnn { #1 }{ varlist }{ seq }
1106     }
1107     { #2 }
1108 }
1109
1110 \cs_generate_variant:Nn \proxy_push_member:nnn { Vnn }
1111

```

(End definition for `\proxy_push_member:nnn`. This function is documented on page 13.)

**`\proxy_push_embedded:nnn`** Push a new embedded object inside a proxy.

```

1112
1113 \cs_new_protected:Nn \proxy_push_embedded:nnn
1114 {
1115     \object_newconst_str:nnx { #1 }{ #2 _ proxy }{ #3 }
1116     \seq_gput_left:cn
1117     {
1118         \object_member_adr:nnn { #1 }{ objlist }{ seq }
1119     }
1120     { #2 }
1121 }
1122
1123 \cs_generate_variant:Nn \proxy_push_embedded:nnn { Vnn }
1124

```

(End definition for `\proxy_push_embedded:nnn`. This function is documented on page 14.)

`\proxy_add_initializer:nN` Push a new embedded object inside a proxy.

```
1125
1126 \cs_new_protected:Nn \proxy_add_initializer:nN
1127 {
1128   \tl_gput_right:cn
1129   {
1130     \object_member_adr:nnn { #1 }{ init }{ t1 }
1131   }
1132   { #2 }
1133 }
1134
1135 \cs_generate_variant:Nn \proxy_add_initializer:nN { VN }
1136
```

(End definition for `\proxy_add_initializer:nN`. This function is documented on page 14.)

`\c_proxy_address_str` Variable containing the address of the proxy object.

```
1137
1138 \str_const:Nx \c_proxy_address_str
1139 { \object_address:nn { rawobjects }{ proxy } }
1140
1141 \object_newconst_str:nnn
1142 {
1143   \object_embedded_adr:Vn \c_proxy_address_str { /_I/ }
1144 }
1145 { M }{ rawobjects }
1146
1147 \object_newconst_str:nnV
1148 {
1149   \object_embedded_adr:Vn \c_proxy_address_str { /_I/ }
1150 }
1151 { P } \c_proxy_address_str
1152
1153 \object_newconst_str:nnV
1154 {
1155   \object_embedded_adr:Vn \c_proxy_address_str { /_I/ }
1156 }
1157 { S } \c_object_global_str
1158
1159 \object_newconst_str:nnV
1160 {
1161   \object_embedded_adr:Vn \c_proxy_address_str { /_I/ }
1162 }
1163 { V } \c_object_public_str
1164
1165
1166 \__rawobjects_initproxy:VnV \c_proxy_address_str { rawobjects } \c_proxy_address_str
1167
1168 \object_new_member:Vnn \c_proxy_address_str { init }{ t1 }
1169
1170 \object_new_member:Vnn \c_proxy_address_str { varlist }{ seq }
1171
1172 \object_new_member:Vnn \c_proxy_address_str { objlist }{ seq }
1173
```



```

1174 \proxy_push_member:Vnn \c_proxy_address_str
1175 { init }{ t1 }
1176 \proxy_push_member:Vnn \c_proxy_address_str
1177 { varlist }{ seq }
1178 \proxy_push_member:Vnn \c_proxy_address_str
1179 { objlist }{ seq }
1180
1181 \proxy_add_initializer:VN \c_proxy_address_str
1182 \__rawobjects_initproxy:nnn
1183

```

(End definition for \c\_proxy\_address\_str. This variable is documented on page 12.)

**\object\_allocate\_incr:NNnnNN** Create an address and use it to instantiate an object

```

\object_gallocate_incr:NNnnNN
\object_allocate_gincr:NNnnNN
\object_gallocate_gincr:NNnnNN
1184
1185 \cs_new:Nn \__rawobjects_combine_aux:nnn
1186 {
1187     anon . #3 . #2 . #1
1188 }
1189
1190 \cs_generate_variant:Nn \__rawobjects_combine_aux:nnn { Vnf }
1191
1192 \cs_new:Nn \__rawobjects_combine:Nn
1193 {
1194     \__rawobjects_combine_aux:Vnf #1 { #2 }
1195     {
1196         \cs_to_str:N #1
1197     }
1198 }
1199
1200 \cs_new_protected:Nn \object_allocate_incr:NNnnNN
1201 {
1202     \object_create_set:NnnfNN #1 { #3 }{ #4 }
1203     {
1204         \__rawobjects_combine:Nn #2 { #3 }
1205     }
1206     #5 #6
1207
1208     \int_incr:N #2
1209 }
1210
1211 \cs_new_protected:Nn \object_gallocate_incr:NNnnNN
1212 {
1213     \object_create_gset:NnnfNN #1 { #3 }{ #4 }
1214     {
1215         \__rawobjects_combine:Nn #2 { #3 }
1216     }
1217     #5 #6
1218
1219     \int_incr:N #2
1220 }
1221
1222 \cs_generate_variant:Nn \object_allocate_incr:NNnnNN { NNvNNN }
1223

```

```

1224 \cs_generate_variant:Nn \object_gallocate_incr:NNnnNN { NNVnNN }
1225
1226 \cs_new_protected:Nn \object_allocate_gincr:NNnnNN
1227 {
1228   \object_create_set:NnnfNN #1 { #3 }{ #4 }
1229   {
1230     \__rawobjects_combine:Nn #2 { #3 }
1231   }
1232   #5 #6
1233
1234   \int_gincr:N #2
1235 }
1236
1237 \cs_new_protected:Nn \object_gallocate_gincr:NNnnNN
1238 {
1239   \object_create_gset:NnnfNN #1 { #3 }{ #4 }
1240   {
1241     \__rawobjects_combine:Nn #2 { #3 }
1242   }
1243   #5 #6
1244
1245   \int_gincr:N #2
1246 }
1247
1248 \cs_generate_variant:Nn \object_allocate_gincr:NNnnNN { NNVnNN }
1249
1250 \cs_generate_variant:Nn \object_gallocate_gincr:NNnnNN { NNVnNN }
1251

```

(End definition for `\object_allocate_incr:NNnnNN` and others. These functions are documented on page 13.)

**\object\_assign:nn** Copy an object to another one.

```

1252 \cs_new_protected:Nn \object_assign:nn
1253 {
1254   \seq_map_inline:cn
1255   {
1256     \object_member_adr:vnn
1257     {
1258       \object_ncmember_adr:nnn
1259       {
1260         \object_embedded_adr:nn{ #1 }{ /_I/ }
1261       }
1262       { P }{ str }
1263     }
1264     { varlist }{ seq }
1265   }
1266   {
1267     \object_member_set_eq:nnc { #1 }{ ##1 }
1268     {
1269       \object_member_adr:nn{ #2 }{ ##1 }
1270     }
1271   }
1272 }

```

```

1273
1274 \cs_generate_variant:Nn \object_assign:nn { nV, Vn, VV }
(End definition for \object_assign:nn. This function is documented on page 14.)
1275 \endpackage

```