# The lt3rawobjects package

Paolo De Donato

Released on 2023/03/17 Version 2.4

# Contents

# 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 specification (`https://semver.org/`). In particular any major version update (for example from `1.2` to `2.0`) may introduce imcompatible 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 Instead changes introduced in minor and patch version updates are always backward compatible, and any withdrawn function is declared deprecated instead of being removed.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 if the name of a control sequence \⟨*name*⟩ that can 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 a pure 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 `\rwobj_address_f:n` and pass them to your function. For example,

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

Remember that `\rwobj_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 ⟨*module*⟩ name in which such address is allocated, then an underscore (`_`) and the ⟨*identifier*⟩ that uniquely identifies the resource inside the module. The ⟨*module*⟩ should contain only lowercase ASCII letters.

A *pointer* is just a LaTeX3 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 ⟨*type*⟩ 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 later in this document.

## 3   Address spaces and objects

Since in LaTeX3 all the functions and variables are declared globally a package mantainer can't just allocate its resources on a random address in order to avoid possible clashes between independent packages. Moreover, a lot of packages need to create new resources during document composition from an user input. Since the user is not aware of the implementation the package owner should insure that any user input doesn't try to allocate new resources on already taken addresses.

For these reasons each address should be contained inside an *address space* which is just a string that avoid clashes between resources. More precisely, the address of a function should have the following form:

```
1    ⟨address space⟩_⟨function name⟩:⟨arguments⟩
```

whereas the address of variables and constants should be

```
1    ⟨scope⟩_⟨address space⟩_⟨variable name⟩_⟨type⟩
```

where ⟨*scope*⟩ is one of `g`, `l`, `c`.

More precisely, an address space consists in a sequence of one or more nonempty alphanumeric strings, called *address names*, separated by dots (`.`). For example

```
1    mod1
2    A.b2
3    pp.9.12w
```

are all valid address spaces. The leftmost address name in an address space is also called *module name* and should uniquely identify the entire package: different LaTeX3 packages can't have the same module name. An address space that has only the module name is also called *primary address space* or *module space*. Any other address name that follows the module name are called *subspace names*. For example in the address space `mod.2.ap` the module name is `mod` whereas `2` and `ap` are subspace names.

An address space is comtained in another one if and only if the first one contains all the address names of the second in the same order, for example `AA.BB.CC` is contained in `AA` and in `AA.BB` but not in `CC`, `AA.B` or in `AA.BB.CC.E`.

An address space can also be seen as a container that holds macros/variables/functions that share common functionalities. For example if you want to store the coordinated of a three dimensional point you can put them inside three variables contained in a subspace of your module space, in this way the resulting address space will represent your original point.

You can even pass an address space to a different package by just passing its name as a token list/string, but you and the destination package should before agree upon a common protocol in order to make the passed address space understandable. You can clearly make your own protocol for address spaces exchanging, but lt3rawobjects already introduces a valid exchanging protocol that you would use in your project.

An address space that follows the exchanging protocol defined in lt3rawobjects is called *object*. All the macros/variables/constants/functions/subspaces contained in an object are also called *fields*. Objects and fields should be always created with functions defined in this package. Remember that objects are also address spaces, so you can identify them by their address space name. Such string can be seen also as a pure address that points to your object and so it'll be called also *object address*.

## 4  Proxy

Instead of generic address spaces, objects should be created before to use them. In order to create a new object you should uuse a particular kind of object that holds all the necessary information to instantiate your object. These objects that are needed to instantiate new objects are called *proxy*, and the proxy used to instantiated an object is its *generator*. Every object knows the address of its generator.

For example the proxy `rwobj_empty` can be used to create empty objects, which are objects that contains only the required fields needed by the exchanging protocol. However, you can create your own proxy that initialized your object fields as you wish. To create new proxies you can use the proxy `rwobj_proxygen`.

Proxy can also be used as interfaces to group objects by common utilities. Indeed you can use the function `\rwobj_proxy_test_eq:nn` to test if an object has been created with the selected proxy.

## 5  Fields

Currently objects can contain as fields the following entities:

- LaTeX3 variables, which are also called *member*;

- LaTeX3 constants, which are also called *constant*;

- LaTeX3 functions, which are also called *method*;

- generic control sequences, which are also called *macros*;

- other objects, which are also 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 LaTeX3 variables). You should always create global object unless you specifically need local members.

## 5.1 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 remore 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 contant are created directly in the target object.

## 5.2 Methods

Methods are LaTeX3 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 `nmethod` whereas those dealing with remote methods contain `rmethod` in their names.

## 5.3 Members

Members are just mutable LaTeX3 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.

## 5.4 Embedded object

Objects can be created inside other objects by just using an address space contained in the parent object address, for example an object created at the address `mod.A.B` is automatically contained in the object at address `mod.A`. A contained object is independent from the contained, imparticular they can have different scope.

An *embedded object* is not just an object instantiated in a parent object, but it also inherit some properties of the parent object like the scope.

# 6 Library functions

## 6.1 Common functions

`\rwobj_address_f:n` ⋆   `\rwobj_address_f:n {⟨address⟩}`

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

## 6.2   Base object functions

`\object_address:nn` ☆   `\object_address:nn {⟨module⟩} {⟨id⟩}`

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. Equivalent to

```
1    \tl_to_str:n{⟨module⟩}.\tl_to_str:n{⟨id⟩}
```

From: 1.0

`\object_address_set:Nnn`   `\object_address_set:Nnn ⟨str var⟩ {⟨module⟩} {⟨id⟩}`
`\object_address_gset:Nnn`

Stores the adress of selected object inside the string variable ⟨*str var*⟩.

From: 1.1

`\object_embedded_adr:nn` ☆   `\object_embedded_adr:nn {⟨address⟩} {⟨id⟩}`
`\object_embedded_adr:Vn` ☆

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. Equivalent to

```
1    ⟨address⟩.\tl_to_str:n{⟨id⟩}
```

From: 2.2

`\object_if_exist_p:n` ⋆   `\object_if_exist_p:n {⟨address⟩}`
`\object_if_exist_p:V` ⋆   `\object_if_exist:nTF {⟨address⟩} {⟨true code⟩} {⟨false code⟩}`
`\object_if_exist:nTF` ⋆
`\object_if_exist:VTF` ⋆   Tests if an object was instantiated at the specified address.

From: 1.0

`\object_get_module:n` ⋆   `\object_get_module:n {⟨address⟩}`
`\object_get_module:V` ⋆   `\object_get_proxy_adr:n {⟨address⟩}`
`\object_get_proxy_adr:n` ⋆
`\object_get_proxy_adr:V` ⋆   Get the object module and its generator.

From: 1.0

`\object_if_local_p:n` ⋆   `\object_if_local_p:n {⟨address⟩}`
`\object_if_local_p:V` ⋆   `\object_if_local:nTF {⟨address⟩} {⟨true code⟩} {⟨false code⟩}`
`\object_if_local:nTF` ⋆
`\object_if_local:VTF` ⋆   Tests if the object is local or global.
`\object_if_global_p:n` ⋆
`\object_if_global_p:V` ⋆   From: 1.0
`\object_if_global:nTF` ⋆
`\object_if_global:VTF` ⋆

| | | |
|---|---|---|
| `\object_if_public_p:n` | ⋆ | `\object_if_public_p:n {⟨address⟩}` |
| `\object_if_public_p:V` | ⋆ | `\object_if_public:nTF {⟨address⟩} {⟨true code⟩} {⟨false code⟩}` |
| `\object_if_public:nTF` | ⋆ | Tests if the object is public or private. |
| `\object_if_public:VTF` | ⋆ | From: 1.0 |
| `\object_if_private_p:n` | ⋆ | |
| `\object_if_private_p:V` | ⋆ | |
| `\object_if_private:nTF` | ⋆ | |
| `\object_if_private:VTF` | ⋆ | |

## 6.3 Members

| | | |
|---|---|---|
| `\object_member_adr:nnn` | ☆ | `\object_member_adr:nnn {⟨address⟩} {⟨member name⟩} {⟨member type⟩}` |
| `\object_member_adr:(Vnn\|nnv)` | ☆ | `\object_member_adr:nn {⟨address⟩} {⟨member name⟩}` |
| `\object_member_adr:nn` | ☆ | |
| `\object_member_adr:Vn` | ☆ | |

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

From: 1.0

| | |
|---|---|
| `\object_member_if_exist_p:nnn` ⋆ | `\object_member_if_exist_p:nnn {⟨address⟩} {⟨member name⟩} {⟨member` |
| `\object_member_if_exist_p:Vnn` ⋆ | `type⟩}` |
| `\object_member_if_exist:nnnTF` ⋆ | `\object_member_if_exist:nnnTF {⟨address⟩} {⟨member name⟩} {⟨member` |
| `\object_member_if_exist:VnnTF` ⋆ | `type⟩} {⟨true code⟩} {⟨false code⟩}` |

Tests if the specified member exist.

From: 2.0

| | |
|---|---|
| `\object_member_if_tracked_p:nn` ⋆ | `\object_member_if_tracked_p:nn {⟨address⟩} {⟨member name⟩}` |
| `\object_member_if_tracked_p:Vn` ⋆ | `\object_member_if_tracked:nnTF {⟨address⟩} {⟨member name⟩} {⟨true` |
| `\object_member_if_tracked:nnTF` ⋆ | `code⟩} {⟨false code⟩}` |
| `\object_member_if_tracked:VnTF` ⋆ | |

Tests if the specified member exist and is tracked.

From: 2.3

| | |
|---|---|
| `\object_member_type:nn` ⋆ | `\object_member_type:nn {⟨address⟩} {⟨member name⟩}` |
| `\object_member_type:Vn` ⋆ | Fully expands to the type of specified tracked member. |

From: 1.0

| |
|---|
| `\object_new_member:nnn` `\object_new_member:nnn {⟨address⟩} {⟨member name⟩} {⟨member type⟩}` |
| `\object_new_member:(Vnn\|nnv)` |

Creates a new member with specified name and type. The created member is not tracked.

From: 1.0

| |
|---|
| `\object_new_member_tracked:nnn` `\object_new_member_tracked:nnn {⟨address⟩} {⟨member name⟩} {⟨member` |
| `\object_new_member_tracked:Vnn` `type⟩}` |

Creates a new tracked member.

From: 2.3

| | | |
|---|---|---|
| `\object_member_use:nnn` | ⋆ | `\object_member_use:nnn {`⟨*address*⟩`} {`⟨*member name*⟩`} {`⟨*member type*⟩`}` |
| `\object_member_use:(Vnn|nnv)` | ⋆ | `\object_member_use:nn {`⟨*address*⟩`} {`⟨*member name*⟩`}` |
| `\object_member_use:nn` | ⋆ | |
| `\object_member_use:Vn` | ⋆ | |

Uses the specified member variable.

From: 1.0

| | |
|---|---|
| `\object_member_set:nnnn` | `\object_member_set:nnnn {`⟨*address*⟩`} {`⟨*member name*⟩`} {`⟨*member type*⟩`}` |
| `\object_member_set:(nnvn|Vnnn)` | `{`⟨*value*⟩`}` |
| `\object_member_set:nnn` | `\object_member_set:nnn {`⟨*address*⟩`} {`⟨*member name*⟩`} {`⟨*value*⟩`}` |
| `\object_member_set:Vnn` | |

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

| | |
|---|---|
| `\object_member_set_eq:nnnN` | `\object_member_set_eq:nnnN {`⟨*address*⟩`} {`⟨*member name*⟩`}` |
| `\object_member_set_eq:(nnvN|VnnN|nnnc|Vnnc)` | `{`⟨*member type*⟩`}` ⟨*variable*⟩ |
| `\object_member_set_eq:nnN` | `\object_member_set_eq:nnN {`⟨*address*⟩`} {`⟨*member name*⟩`}` |
| `\object_member_set_eq:(VnN|nnc|Vnc)` | ⟨*variable*⟩ |

Sets the value of specified member equal to the value of ⟨*variable*⟩.

From: 1.0

| | |
|---|---|
| `\object_member_generate:NN` | `\object_member_generate:NN \`⟨*name₁*⟩ `\`⟨*name₂*⟩`:`⟨*arg1*⟩⟨*args*⟩ |
| `\object_member_generate_protected:NN` | |

Define the new functions `\`⟨*name₁*⟩`:nnn`⟨*Targs*⟩ and `\`⟨*name₁*⟩`:nn`⟨*Targs*⟩ that pass to
`\`⟨*name₂*⟩`:`⟨*arg1*⟩⟨*args*⟩ the specified member address as the first argument. ⟨*Targs*⟩ is
a list of argument specifications obtained by transforming each element of ⟨*args*⟩ to `n`, `N`,
`w`, `T` or `F`.

The first three parameters of `\`⟨*name₁*⟩`:nnn`⟨*args*⟩ should be in the following order:

1. an object address;

2. a member name;

3. the type of specified member.

Function `\`⟨*name₁*⟩`:nn`⟨*args*⟩ only accepts the first two parameters and works only
with tracked members. Notice that ⟨*arg1*⟩ must be only one of the following: `n`, `c`, `v`, `x`,
`f`, `e`, `o`.

From: 2.3

| | |
|---|---|
| `\object_member_generate_inline:Nnn` | `\object_member_generate_inline:Nnn \`⟨*name₁*⟩ `{`⟨*name₂*⟩`}` |
| `\object_member_generate_protected_inline:Nnn` | `{`⟨*arg1*⟩⟨*args*⟩`}` |

Works as `\object_member_generate:NN`, however in ⟨*name₂*⟩ you can use parameters `#1`
and `#2` to compose the needed function. Parameter `#1` expands to the (fully expanded)
member type and `#2` is equal to `g` if the object is global and it's empty if it is local.

From: 2.3

## 6.4   Constants

| | |
|---|---|
| `\object_ncmember_adr:nnn` ☆ | `\object_ncmember_adr:nnn {⟨address⟩} {⟨member name⟩} {⟨member type⟩}` |
| `\object_ncmember_adr:(Vnn|vnn)` ☆ | |
| `\object_rcmember_adr:nnn` ☆ | |
| `\object_rcmember_adr:Vnn` ☆ | |

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

      From:   2.0

| | |
|---|---|
| `\object_ncmember_if_exist_p:nnn` ⋆ | `\object_ncmember_if_exist_p:nnn {⟨address⟩} {⟨member name⟩} {⟨member` |
| `\object_ncmember_if_exist_p:Vnn` ⋆ | `type⟩}` |
| `\object_ncmember_if_exist:nnnTF` ⋆ | `\object_ncmember_if_exist:nnnTF {⟨address⟩} {⟨member name⟩} {⟨member` |
| `\object_ncmember_if_exist:VnnTF` ⋆ | `type⟩} {⟨true code⟩} {⟨false code⟩}` |
| `\object_rcmember_if_exist_p:nnn` ⋆ | |
| `\object_rcmember_if_exist_p:Vnn` ⋆ | |
| `\object_rcmember_if_exist:nnnTF` ⋆ | |
| `\object_rcmember_if_exist:VnnTF` ⋆ | |

Tests if the specified member constant exist.

      From:   2.0

| | |
|---|---|
| `\object_ncmember_use:nnn` ⋆ | `\object_ncmember_use:nnn {⟨address⟩} {⟨member name⟩} {⟨member type⟩}` |
| `\object_ncmember_use:Vnn` ⋆ | |
| `\object_rcmember_use:nnn` ⋆ | Uses the specified near/remote constant member. |
| `\object_rcmember_use:Vnn` ⋆ | |

      From:   2.0

| | |
|---|---|
| `\object_ncmember_generate:NN` | `\object_ncmember_generate:NN \⟨name₁⟩ \name₂:⟨arg1⟩⟨args⟩` |
| `\object_ncmember_protected_generate:NN` | |
| `\object_rcmember_generate:NN` | |
| `\object_rcmember_protected_generate:NN` | |

Works as `\object_member_generate:NN` but with constants instead of members.

      From:   2.3

| | |
|---|---|
| `\object_ncmember_generate_inline:Nnn` | `\object_ncmember_generate_inline:Nnn \⟨name₁⟩ {⟨name₂⟩}` |
| `\object_ncmember_protected_generate_inline:Nnn` | `{⟨arg1⟩⟨args⟩}` |
| `\object_rcmember_generate_inline:Nnn` | |
| `\object_rcmember_protected_generate_inline:Nnn` | |

Works as `\object_member_generate_inline:Nnn` but with constants instead of members.

      From:   2.3

## 6.5   Methods

| | |
|---|---|
| `\object_ncmethod_adr:nnn` | ☆ `\object_ncmethod_adr:nnn {`⟨*address*⟩`} {`⟨*method name*⟩`} {`⟨*method* |
| `\object_ncmethod_adr:(Vnn|vnn)` | ☆ *variant*⟩`}` |
| `\object_rcmethod_adr:nnn` | ☆ |
| `\object_rcmethod_adr:Vnn` | ☆ |

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

| | |
|---|---|
| `\object_ncmethod_if_exist_p:nnn` | ⋆ `\object_ncmethod_if_exist_p:nnn {`⟨*address*⟩`} {`⟨*method name*⟩`} {`⟨*method* |
| `\object_ncmethod_if_exist_p:Vnn` | ⋆ *variant*⟩`}` |
| `\object_ncmethod_if_exist:nnnTF` | ⋆ `\object_ncmethod_if_exist:nnnTF {`⟨*address*⟩`} {`⟨*method name*⟩`} {`⟨*method* |
| `\object_ncmethod_if_exist:VnnTF` | ⋆ *variant*⟩`} {`⟨*true code*⟩`} {`⟨*false code*⟩`}` |
| `\object_rcmethod_if_exist_p:nnn` | ⋆ |
| `\object_rcmethod_if_exist_p:Vnn` | ⋆ |
| `\object_rcmethod_if_exist:nnnTF` | ⋆ |
| `\object_rcmethod_if_exist:VnnTF` | ⋆ |

Tests if the specified method constant exist.
From:   2.0

| |
|---|
| `\object_new_cmethod:nnnn` |
| `\object_new_cmethod:Vnnn` |

`\object_new_cmethod:nnnn {`⟨*address*⟩`} {`⟨*method name*⟩`} {`⟨*method arguments*⟩`} {`⟨*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

| | |
|---|---|
| `\object_ncmethod_call:nnn` | ⋆ `\object_ncmethod_call:nnn {`⟨*address*⟩`} {`⟨*method name*⟩`} {`⟨*method variant*⟩`}` |
| `\object_ncmethod_call:Vnn` | ⋆ |
| `\object_rcmethod_call:nnn` | ⋆ |
| `\object_rcmethod_call:Vnn` | ⋆ |

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 ⟨type⟩ and sets its value to ⟨value⟩.

    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 ⟨comma-list⟩.

    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 \⟨type⟩_const:cn to create the specified constant.

    From: 2.1

## 6.7 Macros

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

Address of specified macro.

    From: 2.2

---

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

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

| | |
|---|---|
| `\object_if_proxy_p:n` ⋆<br>`\object_if_proxy_p:V` ⋆<br>`\object_if_proxy:nTF` ⋆<br>`\object_if_proxy:VTF` ⋆ | `\object_if_proxy_p:n {⟨address⟩}`<br>`\object_if_proxy:nTF {⟨address⟩} {⟨true code⟩} {⟨false code⟩}`<br><br>Test if the specified object is a proxy object.<br>     From:   1.0 |

| | |
|---|---|
| `\object_test_proxy_p:nn` ⋆<br>`\object_test_proxy_p:Vn` ⋆<br>`\object_test_proxy:nnTF` ⋆<br>`\object_test_proxy:VnTF` ⋆ | `\object_test_proxy_p:nn {⟨object address⟩} {⟨proxy address⟩}`<br>`\object_test_proxy:nnTF {⟨object address⟩} {⟨proxy address⟩} {⟨true code⟩} {⟨false code⟩}`<br><br>Test if the specified object is generated by the selected proxy, where ⟨*proxy variable*⟩ is a string variable holding the proxy address.<br><br>**TEXhackers note:** Remember that this command uses internally an `e` expansion so in older engines (any different from LuaLaTeX before 2019) it'll require slow processing. Don't use it in speed critical parts, instead use `\object_test_proxy:nN`.<br><br>     From:   2.0 |

| | |
|---|---|
| `\object_test_proxy_p:nN` ⋆<br>`\object_test_proxy_p:VN` ⋆<br>`\object_test_proxy:nNTF` ⋆<br>`\object_test_proxy:VNTF` ⋆ | `\object_test_proxy_p:nN {⟨object address⟩} ⟨proxy variable⟩`<br>`\object_test_proxy:nNTF {⟨object address⟩} ⟨proxy variable⟩ {⟨true code⟩} {⟨false code⟩}`<br><br>Test if the specified object is generated by the selected proxy, where ⟨*proxy variable*⟩ is a string variable holding the proxy address. The `:nN` variant don't use `e` expansion, instead of `:nn` command, so it can be safely used with older compilers.<br>     From:   2.0 |

| | |
|---|---|
| `\c_proxy_address_str` | The address of the `proxy` object in the `rawobjects` module.<br>     From:   1.0 |

| | |
|---|---|
| `\object_create:nnnNN`<br>`\object_create:VnnNN` | `\object_create:nnnNN {⟨proxy address⟩} {⟨module⟩} {⟨id⟩} ⟨scope⟩ ⟨visibility⟩`<br><br>Creates an object by using the proxy at ⟨*proxy address*⟩ 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`).<br>     From:   1.0 |

| | |
|---|---|
| `\object_create:nnnN`<br>`\object_create:VnnN`<br>`\object_create:nnn`<br>`\object_create:Vnn` | `\object_create:nnnN {⟨proxy address⟩} {⟨module⟩} {⟨id⟩} ⟨scope⟩`<br>`\object_create:nnn {⟨proxy address⟩} {⟨module⟩} {⟨id⟩}`<br><br>Same as `\object_create:nnnNN` but both create only public objects, and the `:nnn` version only global ones. Always use these two function instead of `\object_create:nnnNN` unless you strictly need private objects.<br>     From:   2.3 |

| | |
|---|---|
| `\embedded_create:nnn`<br>`\embedded_create:(Vnn|nvn)` | `\embedded_create:nnn {⟨parent object⟩} {⟨proxy address⟩} {⟨id⟩}`<br><br>Creates an embedded object with name ⟨*id*⟩ inside ⟨*parent object*⟩.<br>     From:   2.2 |

| | |
|---|---|
| `\c_object_local_str` `\c_object_global_str` | Possible values for ⟨*scope*⟩ parameter. |

| | |
|---|---|
| `\c_object_public_str` `\c_object_private_str` | Possible values for ⟨*visibility*⟩ parameter. |

| | |
|---|---|
| `\object_create_set:NnnnNN` `\object_create_set:(NVnnNN\|NnnfNN)` `\object_create_gset:NnnnNN` `\object_create_gset:(NVnnNN\|NnnfNN)` | `\object_create_set:NnnnNN` ⟨*str var*⟩ {⟨*proxy address*⟩} {⟨*module*⟩} {⟨*id*⟩} ⟨*scope*⟩ ⟨*visibility*⟩ |

Creates an object and sets its fully expanded address inside ⟨*str var*⟩.

| | |
|---|---|
| `\object_allocate_incr:NNnnNN` `\object_allocate_incr:NNVnNN` `\object_gallocate_incr:NNnnNN` `\object_gallocate_incr:NNVnNN` `\object_allocate_gincr:NNnnNN` `\object_allocate_gincr:NNVnNN` `\object_gallocate_gincr:NNnnNN` `\object_gallocate_gincr:NNVnNN` | `\object_allocate_incr:NNnnNN` ⟨*str var*⟩ ⟨*int var*⟩ {⟨*proxy address*⟩} {⟨*module*⟩} ⟨*scope*⟩ ⟨*visibility*⟩ |

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

| | |
|---|---|
| `\proxy_create:nnN` `\proxy_create_set:NnnN` `\proxy_create_gset:NnnN` | `\proxy_create:nnN` {⟨*module*⟩} {⟨*id*⟩} ⟨*visibility*⟩ `\proxy_create_set:NnnN` ⟨*str var*⟩ {⟨*module*⟩} {⟨*id*⟩} ⟨*visibility*⟩ |

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`.

| | |
|---|---|
| `\proxy_create:nn` `\proxy_create_set:Nnn` `\proxy_create_gset:Nnn` | `\proxy_create:nn` {⟨*module*⟩} {⟨*id*⟩} `\proxy_create_set:Nnn` ⟨*str var*⟩ {⟨*module*⟩} {⟨*id*⟩} |

Creates a global public proxy object.

| | |
|---|---|
| `\proxy_push_member:nnn` `\proxy_push_member:Vnn` | `\proxy_push_member:nnn` {⟨*proxy address*⟩} {⟨*member name*⟩} {⟨*member type*⟩} |

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.

| | |
|---|---|
| `\proxy_push_embedded:nnn`<br>`\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`<br>`\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.

      From: 2.3

| | |
|---|---|
| `\object_assign:nn`<br>`\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 correspective 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:

```
\int_new:N \g_intc_int
\object_gallocate_gincr:NNVnNN \g_myobj_str \g_intc_int \g_myproxy_str
  { example } \c_object_local_str \c_object_public_str
\tl_gset:cn
  {
    \object_member_adr:Vn \g_myobj_str { myvar }
  }
  { \c_dollar_str{} ~ dollar ~ \c_dollar_str{} }
\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

```
\proxy_create:nn { mymod }{ INT }
\proxy_push_member:nnn
  {
    \object_address:nn { mymod }{ INT }
  }{ var }{ tl }
```

Container proxy

```
\proxy_create:nn { mymod }{ EXT }
\proxy_push_embedded:nnn
  {
    \object_address:nn { mymod }{ EXT }
  }
  { emb }
  {
    \object_address:nn { mymod }{ INT }
  }
```

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

```
\str_new:N \g_EXTobj_str
\int_new:N \g_intcount_int
\object_gallocate_gincr:NNnnNN
  \g_EXTobj_str \g_intcount_int
  {
    \object_address:nn { mymod }{ EXT }
  }
  { mymod }
  \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

## Example 3

Here we show how to properly use `\object_member_generate:NN`. Suppose we don't know `\object_member_use` and we want to use `\tl_use:N` to get the value stored in member `MEM` of object `U` in module `MD3`.

We can do it in this way:

```
1  \tl_use:c
2    {
3      \object_member_adr:nnn
4        { \object_address:nn { MD3 }{ U } }
5        { MEM }{ tl }
6    }
```

but this solution is not so pratical since we should write a lot of code each time. We can then use `\object_member_generate:NN` to define an auxilary macro `\myaux_print_-tl:nnn` in this way:

```
1  \object_member_generate:NN \myaux_print_tl \tl_use:c
```

then we can get the content of our member in this way:

```
1  \myaux_print_tl:nnn
2    { \object_address:nn { MD3 }{ U } }
3    { MEM }{ tl }
```

For example if `U` contains `Hi` then the preceding code will output Hi. If member `MEM` is tracked then you can use also the following command, which is generated together with `\myaux_print_tl:nnn`

```
1  \myaux_print_tl:nn
2    { \object_address:nn { MD3 }{ U } }
3    { MEM }
```

However, this function only works with `tl` members since we use `\tl_use:N`, so you should define a new function for every possible type, and even if you do it newer types introduced in other packages will not be supported. In such cases you can use `\object_member_generate_inline:Nnn` which allows you to build the called function by specifying its name and its parameters. The preceding code then becomes

```
1  \object_member_generate_inline:Nnn \myaux_print_tl { tl_use }{ c }
```

This function does much more: in the second argument you can put also the parameters `#1` and `#2` that will expand respectively to the type of specified member and its scope. Let `\myaux_print:nnn` be our version of `\object_member_use:nnn` that retrieves the valued of the specified member, we are now able to define it in this way:

```
1  \object_member_generate_inline:Nnn \myaux_print { #1_use }{ c }
```

When you use `\myaux_print:nnn` on a member of type `int` it replaces all the recurrences of `#1` with `int`, thus it will call `\int_use:c`.

# 8 Implementation

```
1  ⟨*package⟩
```

```
2  ⟨@@=rawobjects⟩
```

Deprecation message

```
3
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
```

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

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

(*End definition for* `\rwobj_address_f:n`*. This function is documented on page 5.*)

`\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* *6*.)

\object_embedded_adr:nn  Address of embedded object

```
66
67 \cs_new:Nn \object_embedded_adr:nn
68   {
69     #1 \tl_to_str:n{ . #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* *6*.)

`\object_address_set:Nnn`
`\object_address_gset:Nnn`

Saves the address of an object into a string variable

```
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* 6.)

`\object_if_exist_p:n`
`\object_if_exist:n`*TF*

Tests if object exists.

```
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* 6.)

`\object_get_module:n`
`\object_get_proxy_adr:n`

Retrieve the name, module and generating proxy of an object

```
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 }
```

19

```
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 docu-*
*mented on page* 6*.*)

`\object_if_local_p:n`
`\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`

Test the specified parameters.

```
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 6.*)

**\object_macro_adr:nn**
**\object_macro_use:nn**

Generic macro address

```
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
```

21

```
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

```
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, nnf }
255
```

(*End definition for* \object_member_adr:nnn. *This function is documented on page 7.*)

\object_member_if_exist_p:nnn  Tests if the specified member exists
\object_member_if_exist:nnnTF

```
256
257 \prg_new_conditional:Nnn \object_member_if_exist:nnn {p, T, F, TF }
258    {
```

```
259      \cs_if_exist:cTF
260        {
261          \object_member_adr:nnn { #1 }{ #2 }{ #3 }
262        }
263        {
264          \prg_return_true:
265        }
266        {
267          \prg_return_false:
268        }
269    }
270
271  \prg_generate_conditional_variant:Nnn \object_member_if_exist:nnn
272    { Vnn }{ p, T, F, TF }
273
```

(*End definition for* `\object_member_if_exist:nnnTF`*. This function is documented on page 7.*)

`\object_member_if_tracked_p:nn`
`\object_member_if_tracked:nnTF`   Tests if the member is tracked.

```
274
275  \prg_new_conditional:Nnn \object_member_if_tracked:nn {p, T, F, TF }
276    {
277      \cs_if_exist:cTF
278        {
279          \object_rcmember_adr:nnn
280            { #1 }{ #2 _ type }{ str }
281        }
282        {
283          \prg_return_true:
284        }
285        {
286          \cs_if_exist:cTF
287            {
288              \object_ncmember_adr:nnn
289                {
290                  \object_embedded_adr:nn { #1 }{ /_T_/ }
291                }
292                { #2 _ type }{ str }
293            }
294            {
295              \prg_return_true:
296            }
297            {
298              \prg_return_false:
299            }
300        }
301    }
302
303  \prg_generate_conditional_variant:Nnn \object_member_if_tracked:nn
304    { Vn }{ p, T, F, TF }
305
306  \prg_new_eq_conditional:NNn \object_member_if_exist:nn
307    \object_member_if_tracked:nn { p, T, F, TF }
308  \prg_new_eq_conditional:NNn \object_member_if_exist:Vn
```

```
309    \object_member_if_tracked:Vn { p, T, F, TF }
310
```

(*End definition for* \object_member_if_tracked:nnTF. *This function is documented on page* *7.*)

\object_member_type:nn  Deduce the type of tracked members.

```
311
312  \cs_new:Nn \object_member_type:nn
313    {
314      \cs_if_exist:cTF
315        {
316          \object_rcmember_adr:nnn
317            { #1 }{ #2 _ type }{ str }
318        }
319        {
320          \object_rcmember_use:nnn
321            { #1 }{ #2 _ type }{ str }
322        }
323        {
324          \cs_if_exist:cT
325            {
326              \object_ncmember_adr:nnn
327                {
328                  \object_embedded_adr:nn { #1 }{ /_T_/ }
329                }
330                { #2 _ type }{ str }
331            }
332            {
333              \object_ncmember_use:nnn
334                {
335                  \object_embedded_adr:nn { #1 }{ /_T_/ }
336                }
337                { #2 _ type }{ str }
338            }
339        }
340    }
341
```

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

\object_member_adr:nn  Get the address of a member variable

```
342
343  \cs_new:Nn \object_member_adr:nn
344    {
345      \object_member_adr:nnf { #1 }{ #2 }
346        {
347          \object_member_type:nn { #1 }{ #2 }
348        }
349    }
350
351  \cs_generate_variant:Nn \object_member_adr:nn { Vn }
352
```

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

Helper functions for \object_*_generate functions.

```
353
354 \cs_new:Nn \__rawobjects_par_trans:N
355   {
356     \str_case:nnF { #1 }
357       {
358         { N }{ N }
359         { V }{ N }
360         { n }{ n }
361         { v }{ n }
362         { f }{ n }
363         { x }{ n }
364         { e }{ n }
365         { o }{ n }
366         { ~ }{}
367       }
368       { #1 }
369   }
370
371 \cs_new:Nn \__rawobjects_par_trans:n
372   {
373     \str_map_function:nN { #1 } \__rawobjects_par_trans:N
374   }
375
376 \str_new:N \l__rawobjects_tmp_fa_str
377
378 \cs_new_protected:Nn \__rawobjects_save_dat:n
379   {
380     \str_set:Nx \l__rawobjects_tmp_fa_str
381       { \str_tail:n{ #1 } }
382   }
383 \cs_new_protected:Nn \__rawobjects_save_dat:nnN
384   {
385     \str_set:Nx \l__rawobjects_tmp_fa_str
386       { \str_tail:n{ #2 } }
387   }
388 \cs_new_protected:Nn \__rawobjects_save_dat_aux:n
389   {
390     \__rawobjects_save_dat:nnN #1
391   }
392 \cs_generate_variant:Nn \__rawobjects_save_dat_aux:n { f }
393
394 \cs_new_protected:Nn \__rawobjects_save_fun:N
395   {
396     \__rawobjects_save_dat_aux:f { \cs_split_function:N #1 }
397   }
398
399 \cs_new:Nn \__rawobjects_use_dat:nn
400   {
401     #1 : #2 \str_use:N \l__rawobjects_tmp_fa_str
402   }
403
```

Generate member versions of specified functions.

```
404
405 \cs_new_protected:Nn \__rawobjects_mgen:nN
406   {
407     \__rawobjects_save_fun:N #2
408     \cs_new:cpn { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
409       {
410         #2
411           {
412             \object_member_adr:nnn{ ##1 }{ ##2 }{ ##3 }
413           }
414       }
415     \cs_new:cpn { #1 : nn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2
416       {
417         #2
418           {
419             \object_member_adr:nn{ ##1 }{ ##2 }
420           }
421       }
422   }
423 \cs_new_protected:Nn \__rawobjects_mgen_pr:nN
424   {
425     \__rawobjects_save_fun:N #2
426     \cs_new_protected:cpn
427       { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
428       {
429         #2
430           {
431             \object_member_adr:nnn{ ##1 }{ ##2 }{ ##3 }
432           }
433       }
434     \cs_new_protected:cpn
435       { #1 : nn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2
436       {
437         #2
438           {
439             \object_member_adr:nn{ ##1 }{ ##2 }
440           }
441       }
442   }
443
444 \cs_new_protected:Nn \__rawobjects_mgen:nnn
445   {
446     \__rawobjects_save_dat:n { #3 }
447
448     \cs_new:cpn { __rawobjects_auxfun_#1 :nn } ##1##2
449       {
450         \use:c{ #2 : #3 }
451       }
452     \cs_generate_variant:cn { __rawobjects_auxfun_#1 :nn }{ nf, ff }
453
454     \cs_new:cpn { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
455       {
456         \use:c { __rawobjects_auxfun_#1 :nf }
```

26

```
457        { ##3 }
458        {
459          \__rawobjects_scope_pfx_cl:n{ ##1 }
460        }
461        {
462          \object_member_adr:nnn{ ##1 }{ ##2 }{ ##3 }
463        }
464      }
465    \cs_new:cpn { #1 : nn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2
466      {
467        \use:c { __rawobjects_auxfun_#1 :ff }
468          {
469            \object_member_type:nn { ##1 }{ ##2 }
470          }
471          {
472            \__rawobjects_scope_pfx_cl:n{ ##1 }
473          }
474          {
475            \object_member_adr:nn{ ##1 }{ ##2 }
476          }
477      }
478  }
479 \cs_new_protected:Nn \__rawobjects_mgen_pr:nnn
480   {
481     \__rawobjects_save_dat:n { #3 }
482
483     \cs_new:cpn { __rawobjects_auxfun_#1 :nn } ##1##2
484       {
485         \use:c{ #2 : #3 }
486       }
487     \cs_generate_variant:cn { __rawobjects_auxfun_#1 :nn }{ nf, ff }
488
489     \cs_new_protected:cpn
490       { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
491       {
492         \use:c { __rawobjects_auxfun_#1 :nf }
493           { ##3 }
494           {
495             \__rawobjects_scope_pfx_cl:n{ ##1 }
496           }
497           {
498             \object_member_adr:nnn{ ##1 }{ ##2 }{ ##3 }
499           }
500       }
501     \cs_new_protected:cpn
502       { #1 : nn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2
503       {
504         \use:c { __rawobjects_auxfun_#1 :ff }
505           {
506             \object_member_type:nn { ##1 }{ ##2 }
507           }
508           {
509             \__rawobjects_scope_pfx_cl:n{ ##1 }
510           }
```

```
511            {
512              \object_member_adr:nn{ ##1 }{ ##2 }
513            }
514          }
515      }
516
517   \cs_generate_variant:Nn \__rawobjects_mgen:nN { fN }
518   \cs_generate_variant:Nn \__rawobjects_mgen:nnn { fnn }
519   \cs_generate_variant:Nn \__rawobjects_mgen_pr:nN { fN }
520   \cs_generate_variant:Nn \__rawobjects_mgen_pr:nnn { fnn }
521
522   \cs_new_protected:Nn \object_member_generate:NN
523      {
524        \__rawobjects_mgen:fN { \cs_to_str:N #1 } #2
525      }
526
527   \cs_new_protected:Nn \object_member_generate_inline:Nnn
528      {
529        \__rawobjects_mgen:fnn { \cs_to_str:N #1 }{ #2 }{ #3 }
530      }
531   \cs_new_protected:Nn \object_member_generate_protected:NN
532      {
533        \__rawobjects_mgen_pr:fN { \cs_to_str:N #1 } #2
534      }
535
536   \cs_new_protected:Nn \object_member_generate_protected_inline:Nnn
537      {
538        \__rawobjects_mgen_pr:fnn { \cs_to_str:N #1 }{ #2 }{ #3 }
539      }
540
```

(*End definition for* \object_member_generate:NN *and others. These functions are documented on page 8.*)

\object_ncmember_generate:NN  
\object_ncmember_generate_inline:Nnn  
\object_ncmember_generate_protected:NN  
\object_ncmember_generate_protected_inline:Nnn  

Generate ncmember versions of specified functions.

```
541
542   \cs_new_protected:Nn \__rawobjects_ncgen:nN
543      {
544        \__rawobjects_save_fun:N #2
545        \cs_new:cpn { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
546          {
547            #2
548              {
549                \object_ncmember_adr:nnn{ ##1 }{ ##2 }{ ##3 }
550              }
551          }
552      }
553   \cs_new_protected:Nn \__rawobjects_ncgen_pr:nN
554      {
555        \__rawobjects_save_fun:N #2
556        \cs_new_protected:cpn
557          { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
558          {
559            #2
```

28

```
560       {
561           \object_ncmember_adr:nnn{ ##1 }{ ##2 }{ ##3 }
562       }
563     }
564   }

565
566 \cs_new_protected:Nn \__rawobjects_ncgen:nnn
567   {
568     \__rawobjects_save_dat:n { #3 }
569
570     \cs_new:cpn { __rawobjects_auxfun_#1 :nn } ##1##2
571       {
572         \use:c{ #2 : #3 }
573       }
574     \cs_generate_variant:cn { __rawobjects_auxfun_#1 :nn }{ nf }
575
576     \cs_new:cpn { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
577       {
578         \use:c { __rawobjects_auxfun_#1 :nf }
579           { ##3 }
580           {
581             \__rawobjects_scope_pfx_cl:n{ ##1 }
582           }
583           {
584             \object_ncmember_adr:nnn{ ##1 }{ ##2 }{ ##3 }
585           }
586       }
587   }
588 \cs_new_protected:Nn \__rawobjects_ncgen_pr:nnn
589   {
590     \__rawobjects_save_dat:n { #3 }
591
592     \cs_new:cpn { __rawobjects_auxfun_#1 :nn } ##1##2
593       {
594         \use:c{ #2 : #3 }
595       }
596     \cs_generate_variant:cn { __rawobjects_auxfun_#1 :nn }{ nf }
597
598     \cs_new_protected:cpn
599       { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
600       {
601         \use:c { __rawobjects_auxfun_#1 :nf }
602           { ##3 }
603           {
604             \__rawobjects_scope_pfx_cl:n{ ##1 }
605           }
606           {
607             \object_ncmember_adr:nnn{ ##1 }{ ##2 }{ ##3 }
608           }
609       }
610   }

611
612 \cs_generate_variant:Nn \__rawobjects_ncgen:nN { fN }
613 \cs_generate_variant:Nn \__rawobjects_ncgen:nnn { fnn }
```

```
614 \cs_generate_variant:Nn \__rawobjects_ncgen_pr:nN { fN }
615 \cs_generate_variant:Nn \__rawobjects_ncgen_pr:nnn { fnn }
616
617 \cs_new_protected:Nn \object_ncmember_generate:NN
618   {
619     \__rawobjects_ncgen:fN { \cs_to_str:N #1 } #2
620   }
621
622 \cs_new_protected:Nn \object_ncmember_generate_inline:Nnn
623   {
624     \__rawobjects_ncgen:fnn { \cs_to_str:N #1 }{ #2 }{ #3 }
625   }
626 \cs_new_protected:Nn \object_ncmember_generate_protected:NN
627   {
628     \__rawobjects_ncgen_pr:fN { \cs_to_str:N #1 } #2
629   }
630
631 \cs_new_protected:Nn \object_ncmember_generate_protected_inline:Nnn
632   {
633     \__rawobjects_ncgen_pr:fnn { \cs_to_str:N #1 }{ #2 }{ #3 }
634   }
635
```

(*End definition for* `\object_ncmember_generate:NN` *and others. These functions are documented on page 9.*)

`\object_rcmember_generate:NN`
`\object_rcmember_generate_inline:Nnn`
`\object_rcmember_generate_protected:NN`
`ject_rcmember_generate_protected_inline:Nnn`

Generate ncmember versions of specified functions.

```
636
637 \cs_new_protected:Nn \__rawobjects_rcgen:nN
638   {
639     \__rawobjects_save_fun:N #2
640     \cs_new:cpn { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
641       {
642         #2
643           {
644             \object_rcmember_adr:nnn{ ##1 }{ ##2 }{ ##3 }
645           }
646       }
647   }
648 \cs_new_protected:Nn \__rawobjects_rcgen_pr:nN
649   {
650     \__rawobjects_save_fun:N #2
651     \cs_new_protected:cpn
652       { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
653       {
654         #2
655           {
656             \object_rcmember_adr:nnn{ ##1 }{ ##2 }{ ##3 }
657           }
658       }
659   }
660
661 \cs_new_protected:Nn \__rawobjects_rcgen:nnn
662   {
```

```
663        \__rawobjects_save_dat:n { #3 }

664
665        \cs_new:cpn { __rawobjects_auxfun_#1 :nn } ##1##2
666          {
667            \use:c{ #2 : #3 }
668          }
669        \cs_generate_variant:cn { __rawobjects_auxfun_#1 :nn }{ nf }

670
671        \cs_new:cpn { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
672          {
673            \use:c { __rawobjects_auxfun_#1 :nf }
674              { ##3 }
675              {
676                \__rawobjects_scope_pfx_cl:n{ ##1 }
677              }
678              {
679                \object_rcmember_adr:nnn{ ##1 }{ ##2 }{ ##3 }
680              }
681          }
682      }
683  \cs_new_protected:Nn \__rawobjects_rcgen_pr:nnn
684    {
685        \__rawobjects_save_dat:n { #3 }

686
687        \cs_new:cpn { __rawobjects_auxfun_#1 :nn } ##1##2
688          {
689            \use:c{ #2 : #3 }
690          }
691        \cs_generate_variant:cn { __rawobjects_auxfun_#1 :nn }{ nf }

692
693        \cs_new_protected:cpn
694          { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
695          {
696            \use:c { __rawobjects_auxfun_#1 :nf }
697              { ##3 }
698              {
699                \__rawobjects_scope_pfx_cl:n{ ##1 }
700              }
701              {
702                \object_rcmember_adr:nnn{ ##1 }{ ##2 }{ ##3 }
703              }
704          }
705      }

706
707  \cs_generate_variant:Nn \__rawobjects_rcgen:nN { fN }
708  \cs_generate_variant:Nn \__rawobjects_rcgen:nnn { fnn }
709  \cs_generate_variant:Nn \__rawobjects_rcgen_pr:nN { fN }
710  \cs_generate_variant:Nn \__rawobjects_rcgen_pr:nnn { fnn }

711
712  \cs_new_protected:Nn \object_rcmember_generate:NN
713    {
714      \__rawobjects_rcgen:fN { \cs_to_str:N #1 } #2
715    }

716
```

```
717 \cs_new_protected:Nn \object_rcmember_generate_inline:Nnn
718   {
719     \__rawobjects_rcgen:fnn { \cs_to_str:N #1 }{ #2 }{ #3 }
720   }
721 \cs_new_protected:Nn \object_rcmember_generate_protected:NN
722   {
723     \__rawobjects_rcgen_pr:fN { \cs_to_str:N #1 } #2
724   }
725
726 \cs_new_protected:Nn \object_rcmember_generate_protected_inline:Nnn
727   {
728     \__rawobjects_rcgen_pr:fnn { \cs_to_str:N #1 }{ #2 }{ #3 }
729   }
730
```

(*End definition for* `\object_rcmember_generate:NN` *and others. These functions are documented on page* <span style="color:red">*9*</span>.)

Auxilary functions

```
731
732 \cs_generate_variant:Nn \cs_generate_variant:Nn { cx }
733
734 \cs_new_protected:Nn \__rawobjects_genmem_int:nnn
735   {
736     \__rawobjects_mgen:nnn { #1 }{ #2 }{ #3 }
737     \cs_generate_variant:cx
738       { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str }
739       { Vnn \str_use:N \l__rawobjects_tmp_fa_str, nnv \str_use:N \l__rawobjects_tmp_fa_str }
740     \cs_generate_variant:cx
741       { #1 : nn \str_use:N \l__rawobjects_tmp_fa_str }
742       { Vn \str_use:N \l__rawobjects_tmp_fa_str }
743   }
744 \cs_new_protected:Nn \__rawobjects_genmem_pr_int:nnn
745   {
746     \__rawobjects_mgen_pr:nnn { #1 }{ #2 }{ #3 }
747     \cs_generate_variant:cx
748       { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str }
749       { Vnn \str_use:N \l__rawobjects_tmp_fa_str, nnv \str_use:N \l__rawobjects_tmp_fa_str }
750     \cs_generate_variant:cx
751       { #1 : nn \str_use:N \l__rawobjects_tmp_fa_str }
752       { Vn \str_use:N \l__rawobjects_tmp_fa_str }
753   }
754
755 \cs_new_protected:Nn \__rawobjects_genncm_int:nnn
756   {
757     \__rawobjects_ncgen:nnn { #1 }{ #2 }{ #3 }
758     \cs_generate_variant:cx
759       { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str }
760       { Vnn \str_use:N \l__rawobjects_tmp_fa_str }
761   }
762 \cs_new_protected:Nn \__rawobjects_genncm_pr_int:nnn
763   {
764     \__rawobjects_ncgen_pr:nnn { #1 }{ #2 }{ #3 }
765     \cs_generate_variant:cx
766       { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str }
```

```
767        { Vnn \str_use:N \l__rawobjects_tmp_fa_str }
768      }
769
770   \cs_new_protected:Nn \__rawobjects_genrcm_int:nnn
771     {
772        \__rawobjects_rcgen:nnn { #1 }{ #2 }{ #3 }
773        \cs_generate_variant:cx
774           { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str }
775           { Vnn \str_use:N \l__rawobjects_tmp_fa_str }
776     }
777   \cs_new_protected:Nn \__rawobjects_genrcm_pr_int:nnn
778     {
779        \__rawobjects_rcgen_pr:nnn { #1 }{ #2 }{ #3 }
780        \cs_generate_variant:cx
781           { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str }
782           { Vnn \str_use:N \l__rawobjects_tmp_fa_str }
783     }
784
785
786   \msg_new:nnnn { rawobjects }{ noerr }{ Unspecified ~ scope }
787     {
788        Object ~ #1 ~ hasn't ~ a ~ scope ~ variable
789     }
790
```

`\object_new_member:nnn`
`\object_new_member_tracked:nnn`

Creates a new member variable

```
791
792   \__rawobjects_genmem_pr_int:nnn { object_new_member }{ #1 _ new }{ c }
793
794   \cs_new_protected:Nn \object_new_member_tracked:nnn
795     {
796        \object_new_member:nnn { #1 }{ #2 }{ #3 }
797
798        \str_const:cn
799           {
800              \object_ncmember_adr:nnn
801                 {
802                    \object_embedded_adr:nn { #1 }{ /_T_/ }
803                 }
804                 { #2 _ type }{ str }
805           }
806           { #3 }
807     }
808
809   \cs_generate_variant:Nn \object_new_member_tracked:nnn { Vnn, nnv }
810
```

(*End definition for* `\object_new_member:nnn` *and* `\object_new_member_tracked:nnn`. *These functions are documented on page* *7.*)

`\object_member_use:nnn`
`\object_member_use:nn`

Uses a member variable

```
811
812   \__rawobjects_genmem_int:nnn {object_member_use}{ #1_use }{c}
813
```

```
814 \cs_generate_variant:Nn \object_member_use:nnn {vnn}
815
```

(*End definition for* `\object_member_use:nnn` *and* `\object_member_use:nn`. *These functions are documented on page* *8*.)

`\object_member_set:nnnn`
`\object_member_set:nnn`

Set the value a member.

```
816
817 \__rawobjects_genmem_pr_int:nnn {object_member_set}{ #1_#2 set }{ cn }
818
```

(*End definition for* `\object_member_set:nnnn` *and* `\object_member_set:nnn`. *These functions are documented on page* *8*.)

`\object_member_set_eq:nnnN`
`\object_member_set_eq:nnN`

Make a member equal to another variable.

```
819
820 \__rawobjects_genmem_pr_int:nnn { object_member_set_eq }{ #1 _ #2 set_eq }{ cN }
821
822 \cs_generate_variant:Nn \object_member_set_eq:nnnN { nnnc, Vnnc }
823
824 \cs_generate_variant:Nn \object_member_set_eq:nnN { nnc, Vnc }
825
```

(*End definition for* `\object_member_set_eq:nnnN` *and* `\object_member_set_eq:nnN`. *These functions are documented on page* *8*.)

`\object_ncmember_adr:nnn`

Get address of near constant

```
826
827 \cs_new:Nn \object_ncmember_adr:nnn
828   {
829     \tl_to_str:n{ c _ } #1 \tl_to_str:n { _ CONST _ #2 _ #3 }
830   }
831
832 \cs_generate_variant:Nn \object_ncmember_adr:nnn { Vnn, vnn }
833
```

(*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.

```
834
835 \cs_new:Nn \object_rcmember_adr:nnn
836   {
837     \object_ncmember_adr:vnn
838       {
839         \object_ncmember_adr:nnn
840           {
841             \object_embedded_adr:nn{ #1 }{ /_I_/ }
842           }
843           { P }{ str }
844       }
845       { #2 }{ #3 }
846   }
847
848 \cs_generate_variant:Nn \object_rcmember_adr:nnn { Vnn }
```

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

`\object_ncmember_if_exist_p:nnn`
`\object_ncmember_if_exist:nnnTF`
`\object_rcmember_if_exist_p:nnn`
`\object_rcmember_if_exist:nnnTF`

Tests if the specified member constant exists.

```
849
850 \prg_new_conditional:Nnn \object_ncmember_if_exist:nnn {p, T, F, TF }
851   {
852     \cs_if_exist:cTF
853       {
854         \object_ncmember_adr:nnn { #1 }{ #2 }{ #3 }
855       }
856       {
857         \prg_return_true:
858       }
859       {
860         \prg_return_false:
861       }
862   }
863
864 \prg_new_conditional:Nnn \object_rcmember_if_exist:nnn {p, T, F, TF }
865   {
866     \cs_if_exist:cTF
867       {
868         \object_rcmember_adr:nnn { #1 }{ #2 }{ #3 }
869       }
870       {
871         \prg_return_true:
872       }
873       {
874         \prg_return_false:
875       }
876   }
877
878 \prg_generate_conditional_variant:Nnn \object_ncmember_if_exist:nnn
879   { Vnn }{ p, T, F, TF }
880 \prg_generate_conditional_variant:Nnn \object_rcmember_if_exist:nnn
881   { Vnn }{ p, T, F, TF }
882
```

(*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`
`\object_rcmember_use:nnn`

Uses a near/remote constant.

```
883
884 \__rawobjects_genncm_int:nnn { object_ncmember_use }{ #1_use}{ c }
885
886 \__rawobjects_genrcm_int:nnn { object_rcmember_use }{ #1_use}{ c }
887
```

(*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

```
888
889 \__rawobjects_genncm_pr_int:nnn { object_newconst }{ #1 _ const }{ cn }
890
```

(*End definition for* `\object_newconst:nnnn`. *This function is documented on page 11.*)

35

Create constants

```
891
892 \cs_new_protected:Nn \object_newconst_tl:nnn
893   {
894     \object_newconst:nnnn { #1 }{ #2 }{ tl }{ #3 }
895   }
896 \cs_new_protected:Nn \object_newconst_str:nnn
897   {
898     \object_newconst:nnnn { #1 }{ #2 }{ str }{ #3 }
899   }
900 \cs_new_protected:Nn \object_newconst_int:nnn
901   {
902     \object_newconst:nnnn { #1 }{ #2 }{ int }{ #3 }
903   }
904 \cs_new_protected:Nn \object_newconst_clist:nnn
905   {
906     \object_newconst:nnnn { #1 }{ #2 }{ clist }{ #3 }
907   }
908 \cs_new_protected:Nn \object_newconst_dim:nnn
909   {
910     \object_newconst:nnnn { #1 }{ #2 }{ dim }{ #3 }
911   }
912 \cs_new_protected:Nn \object_newconst_skip:nnn
913   {
914     \object_newconst:nnnn { #1 }{ #2 }{ skip }{ #3 }
915   }
916 \cs_new_protected:Nn \object_newconst_fp:nnn
917   {
918     \object_newconst:nnnn { #1 }{ #2 }{ fp }{ #3 }
919   }
920
921 \cs_generate_variant:Nn \object_newconst_tl:nnn { Vnn }
922 \cs_generate_variant:Nn \object_newconst_str:nnn { Vnn }
923 \cs_generate_variant:Nn \object_newconst_int:nnn { Vnn }
924 \cs_generate_variant:Nn \object_newconst_clist:nnn { Vnn }
925 \cs_generate_variant:Nn \object_newconst_dim:nnn { Vnn }
926 \cs_generate_variant:Nn \object_newconst_skip:nnn { Vnn }
927 \cs_generate_variant:Nn \object_newconst_fp:nnn { Vnn }
928
929
930 \cs_generate_variant:Nn \object_newconst_str:nnn { nnx }
931 \cs_generate_variant:Nn \object_newconst_str:nnn { nnV }
932
```

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

Creates a `seq` constant.

```
933
934 \cs_new_protected:Nn \object_newconst_seq_from_clist:nnn
935   {
936     \seq_const_from_clist:cn
937       {
938         \object_ncmember_adr:nnn { #1 }{ #2 }{ seq }
939       }
```

```
940        { #3 }
941    }
942
943  \cs_generate_variant:Nn \object_newconst_seq_from_clist:nnn { Vnn }
944
```

(*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.

```
945
946  \cs_new_protected:Nn \object_newconst_prop_from_keyval:nnn
947    {
948      \prop_const_from_keyval:cn
949        {
950          \object_ncmember_adr:nnn { #1 }{ #2 }{ prop }
951        }
952        { #3 }
953    }
954
955  \cs_generate_variant:Nn \object_newconst_prop_from_keyval:nnn { Vnn }
956
```

(*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

```
957
958  \cs_new:Nn \object_ncmethod_adr:nnn
959    {
960      #1 \tl_to_str:n { _ CMETHOD _ #2 : #3 }
961    }
962
963  \cs_generate_variant:Nn \object_ncmethod_adr:nnn { Vnn , vnn }
964
965  \cs_new:Nn \object_rcmethod_adr:nnn
966    {
967      \object_ncmethod_adr:vnn
968        {
969          \object_ncmember_adr:nnn
970            {
971              \object_embedded_adr:nn{ #1 }{ /_I_/ }
972            }
973            { P }{ str }
974        }
975        { #2 }{ #3 }
976    }
977
978  \cs_generate_variant:Nn \object_ncmethod_adr:nnn { Vnn , vnn }
979  \cs_generate_variant:Nn \object_rcmethod_adr:nnn { Vnn }
980
```

(*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`
`\object_ncmethod_if_exist:nnn` _TF_
`\object_rcmethod_if_exist_p:nnn`
`\object_rcmethod_if_exist:nnn` _TF_

Tests if the specified member constant exists.

```
981
982 \prg_new_conditional:Nnn \object_ncmethod_if_exist:nnn {p, T, F, TF }
983   {
984     \cs_if_exist:cTF
985       {
986         \object_ncmethod_adr:nnn { #1 }{ #2 }{ #3 }
987       }
988       {
989         \prg_return_true:
990       }
991       {
992         \prg_return_false:
993       }
994   }
995
996 \prg_new_conditional:Nnn \object_rcmethod_if_exist:nnn {p, T, F, TF }
997   {
998     \cs_if_exist:cTF
999       {
1000        \object_rcmethodr_adr:nnn { #1 }{ #2 }{ #3 }
1001      }
1002      {
1003        \prg_return_true:
1004      }
1005      {
1006        \prg_return_false:
1007      }
1008  }
1009
1010 \prg_generate_conditional_variant:Nnn \object_ncmethod_if_exist:nnn
1011   { Vnn }{ p, T, F, TF }
1012 \prg_generate_conditional_variant:Nnn \object_rcmethod_if_exist:nnn
1013   { Vnn }{ p, T, F, TF }
1014
```

(*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

```
1015
1016 \cs_new_protected:Nn \object_new_cmethod:nnnn
1017   {
1018     \cs_new:cn
1019   {
1020     \object_ncmethod_adr:nnn { #1 }{ #2 }{ #3 }
1021   }
1022   { #4 }
1023   }
1024
1025 \cs_generate_variant:Nn \object_new_cmethod:nnnn { Vnnn }
1026
```

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

`\object_ncmethod_call:nnn`
`\object_rcmethod_call:nnn`

Calls the specified method.

```
1027
1028 \cs_new:Nn \object_ncmethod_call:nnn
1029   {
1030     \use:c
1031   {
1032     \object_ncmethod_adr:nnn { #1 }{ #2 }{ #3 }
1033   }
1034   }
1035
1036 \cs_new:Nn \object_rcmethod_call:nnn
1037   {
1038     \use:c
1039   {
1040     \object_rcmethod_adr:nnn { #1 }{ #2 }{ #3 }
1041   }
1042   }
1043
1044 \cs_generate_variant:Nn \object_ncmethod_call:nnn { Vnn }
1045 \cs_generate_variant:Nn \object_rcmethod_call:nnn { Vnn }
1046
```

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

```
1047
1048 \cs_new_protected:Nn \__rawobjects_initproxy:nnn
1049   {
1050     \object_newconst:nnnn
1051       {
1052         \object_embedded_adr:nn{ #3 }{ /_I_/ }
1053       }
1054     { ifprox }{ bool }{ \c_true_bool }
1055   }
1056 \cs_generate_variant:Nn \__rawobjects_initproxy:nnn { VnV }
1057
```

`\object_if_proxy_p:n`
`\object_if_proxy:n`*TF*

Test if an object is a proxy.

```
1058
1059 \cs_new:Nn \__rawobjects_bol_com:N
1060   {
1061     \cs_if_exist_p:N #1 && \bool_if_p:N #1
1062   }
1063
1064 \cs_generate_variant:Nn \__rawobjects_bol_com:N { c }
1065
1066 \prg_new_conditional:Nnn \object_if_proxy:n {p, T, F, TF}
1067   {
1068     \cs_if_exist:cTF
1069       {
1070         \object_ncmember_adr:nnn
1071           {
1072             \object_embedded_adr:nn{ #1 }{ /_I_/ }
1073           }
1074         { ifprox }{ bool }
```

39

```
1075            }
1076          {
1077            \bool_if:cTF
1078              {
1079                \object_ncmember_adr:nnn
1080                  {
1081                    \object_embedded_adr:nn{ #1 }{ /_I_/ }
1082                  }
1083                  { ifprox }{ bool }
1084              }
1085              {
1086                \prg_return_true:
1087              }
1088              {
1089                \prg_return_false:
1090              }
1091          }
1092          {
1093            \prg_return_false:
1094          }
1095    }
1096
```

(*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.

```
1097
1098  \prg_generate_conditional_variant:Nnn \str_if_eq:nn { ve }{ TF }
1099
1100  \prg_new_conditional:Nnn \object_test_proxy:nn {p, T, F, TF}
1101    {
1102      \str_if_eq:veTF
1103        {
1104          \object_ncmember_adr:nnn
1105            {
1106              \object_embedded_adr:nn{ #1 }{ /_I_/ }
1107            }
1108            { P }{ str }
1109        }
1110  { #2 }
1111        {
1112          \prg_return_true:
1113        }
1114        {
1115          \prg_return_false:
1116        }
1117    }
1118
1119  \prg_new_conditional:Nnn \object_test_proxy:nN {p, T, F, TF}
1120    {
1121      \str_if_eq:cNTF
1122        {
1123          \object_ncmember_adr:nnn
1124            {
```

```
1125              \object_embedded_adr:nn{ #1 }{ /_I_/ }
1126            }
1127          { P }{ str }
1128        }
1129    #2
1130        {
1131          \prg_return_true:
1132        }
1133        {
1134          \prg_return_false:
1135        }
1136    }
1137
1138 \prg_generate_conditional_variant:Nnn \object_test_proxy:nn
1139   { Vn }{p, T, F, TF}
1140 \prg_generate_conditional_variant:Nnn \object_test_proxy:nN
1141   { VN }{p, T, F, TF}
1142
```

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

`\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`

Creates an object from a proxy.

```
1143
1144 \msg_new:nnnn { rawobjects }{ notproxy }{ Fake ~ proxy }
1145   {
1146     Object ~ #1 ~ is ~ not ~ a ~ proxy.
1147   }
1148
1149 \cs_new_protected:Nn \__rawobjects_force_proxy:n
1150   {
1151     \object_if_proxy:nF { #1 }
1152       {
1153         \msg_error:nnn { rawobjects }{ notproxy }{ #1 }
1154       }
1155   }
1156
1157 \cs_new_protected:Nn \__rawobjects_create_anon:nnnNN
1158   {
1159     \tl_if_empty:nF{ #1 }
1160       {
1161
1162     \__rawobjects_force_proxy:n { #1 }
1163
1164
1165     \object_newconst_str:nnn
1166       {
1167         \object_embedded_adr:nn{ #3 }{ /_I_/ }
1168       }
1169       { M }{ #2 }
1170     \object_newconst_str:nnn
1171       {
1172         \object_embedded_adr:nn{ #3 }{ /_I_/ }
1173       }
```

```
1174        { P }{ #1 }
1175      \object_newconst_str:nnV
1176        {
1177          \object_embedded_adr:nn{ #3 }{ /_I_/ }
1178        }
1179        { S } #4
1180      \object_newconst_str:nnV
1181        {
1182          \object_embedded_adr:nn{ #3 }{ /_I_/ }
1183        }
1184        { V } #5
1185
1186      \seq_map_inline:cn
1187        {
1188          \object_member_adr:nnn { #1 }{ varlist }{ seq }
1189        }
1190        {
1191          \object_new_member:nnv { #3 }{ ##1 }
1192            {
1193              \object_ncmember_adr:nnn { #1 }{ ##1 _ type }{ str }
1194            }
1195        }
1196
1197      \seq_map_inline:cn
1198        {
1199          \object_member_adr:nnn { #1 }{ objlist }{ seq }
1200        }
1201        {
1202          \embedded_create:nvn
1203            { #3 }
1204            {
1205              \object_ncmember_adr:nnn { #1 }{ ##1 _ proxy }{ str }
1206            }
1207            { ##1 }
1208        }
1209
1210      \tl_map_inline:cn
1211        {
1212          \object_member_adr:nnn { #1 }{ init }{ tl }
1213        }
1214        {
1215          ##1 { #1 }{ #2 }{ #3 }
1216        }
1217
1218    }
1219  }
1220
1221 \cs_generate_variant:Nn \__rawobjects_create_anon:nnnNN { xnxNN, xvxcc }
1222
1223 \cs_new_protected:Nn \object_create:nnnNN
1224  {
1225    \__rawobjects_create_anon:xnxNN { #1 }{ #2 }
1226      { \object_address:nn { #2 }{ #3 } }
1227      #4 #5
```

```
1228    }

1229
1230  \cs_generate_variant:Nn \object_create:nnnNN { VnnNN }

1231
1232  \cs_new_protected:Nn \object_create_set:NnnnNN
1233    {
1234      \object_create:nnnNN { #2 }{ #3 }{ #4 } #5 #6
1235      \str_set:Nx #1 { \object_address:nn { #3 }{ #4 } }
1236    }

1237
1238  \cs_new_protected:Nn \object_create_gset:NnnnNN
1239    {
1240      \object_create:nnnNN { #2 }{ #3 }{ #4 } #5 #6
1241      \str_gset:Nx #1 { \object_address:nn { #3 }{ #4 } }
1242    }

1243
1244  \cs_generate_variant:Nn \object_create_set:NnnnNN { NVnnNN, NnnfNN }
1245  \cs_generate_variant:Nn \object_create_gset:NnnnNN { NVnnNN, NnnfNN }

1246

1247

1248
1249  \cs_new_protected:Nn \object_create:nnnN
1250    {
1251      \object_create:nnnNN { #1 }{ #2 }{ #3 } #4 \c_object_public_str
1252    }

1253
1254  \cs_generate_variant:Nn \object_create:nnnN { VnnN }

1255
1256  \cs_new_protected:Nn \object_create_set:NnnnN
1257    {
1258      \object_create_set:NnnnNN #1 { #2 }{ #3 }{ #4 } #5 \c_object_public_str
1259    }

1260
1261  \cs_new_protected:Nn \object_create_gset:NnnnN
1262    {
1263      \object_create_gset:NnnnNN #1 { #2 }{ #3 }{ #4 } #5 \c_object_public_str
1264    }

1265
1266  \cs_generate_variant:Nn \object_create_set:NnnnN { NVnnN }
1267  \cs_generate_variant:Nn \object_create_gset:NnnnN { NVnnN }

1268
1269  \cs_new_protected:Nn \object_create:nnn
1270    {
1271      \object_create:nnnNN { #1 }{ #2 }{ #3 }
1272        \c_object_global_str \c_object_public_str
1273    }

1274
1275  \cs_generate_variant:Nn \object_create:nnn { Vnn }

1276
1277  \cs_new_protected:Nn \object_create_set:Nnnn
1278    {
1279      \object_create_set:NnnnNN #1 { #2 }{ #3 }{ #4 }
1280        \c_object_global_str \c_object_public_str
1281    }
```

```
1282
1283  \cs_new_protected:Nn \object_create_gset:Nnnn
1284    {
1285      \object_create_gset:NnnnNN #1 { #2 }{ #3 }{ #4 }
1286        \c_object_global_str \c_object_public_str
1287    }
1288
1289  \cs_generate_variant:Nn \object_create_set:Nnnn { NVnn }
1290  \cs_generate_variant:Nn \object_create_gset:Nnnn { NVnn }
1291
1292
1293
1294
1295  \cs_new_protected:Nn \embedded_create:nnn
1296    {
1297      \__rawobjects_create_anon:xvxcc { #2 }
1298    {
1299        \object_ncmember_adr:nnn
1300          {
1301            \object_embedded_adr:nn{ #1 }{ /_I_/ }
1302          }
1303          { M }{ str }
1304      }
1305      {
1306        \object_embedded_adr:nn
1307          { #1 }{ #3 }
1308      }
1309      {
1310        \object_ncmember_adr:nnn
1311          {
1312            \object_embedded_adr:nn{ #1 }{ /_I_/ }
1313          }
1314          { S }{ str }
1315      }
1316      {
1317        \object_ncmember_adr:nnn
1318          {
1319            \object_embedded_adr:nn{ #1 }{ /_I_/ }
1320          }
1321          { V }{ str }
1322      }
1323    }
1324
1325  \cs_generate_variant:Nn \embedded_create:nnn { nvn, Vnn }
1326
```

(*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

```
1327
1328  \cs_new_protected:Nn \proxy_create:nn
1329    {
1330      \object_create:VnnNN \c_proxy_address_str { #1 }{ #2 }
1331        \c_object_global_str \c_object_public_str
```

```
1332      }
1333
1334  \cs_new_protected:Nn \proxy_create_set:Nnn
1335    {
1336      \object_create_set:NVnnNN #1 \c_proxy_address_str { #2 }{ #3 }
1337        \c_object_global_str \c_object_public_str
1338    }
1339
1340  \cs_new_protected:Nn \proxy_create_gset:Nnn
1341    {
1342      \object_create_gset:NVnnNN #1 \c_proxy_address_str { #2 }{ #3 }
1343        \c_object_global_str \c_object_public_str
1344    }
1345
1346
1347
1348  \cs_new_protected:Nn \proxy_create:nnN
1349    {
1350      \__rawobjects_launch_deprecate:NN \proxy_create:nnN \proxy_create:nn
1351      \object_create:VnnNN \c_proxy_address_str { #1 }{ #2 }
1352        \c_object_global_str #3
1353    }
1354
1355  \cs_new_protected:Nn \proxy_create_set:NnnN
1356    {
1357      \__rawobjects_launch_deprecate:NN \proxy_create_set:NnnN \proxy_create_set:Nnn
1358      \object_create_set:NVnnNN #1 \c_proxy_address_str { #2 }{ #3 }
1359        \c_object_global_str #4
1360    }
1361
1362  \cs_new_protected:Nn \proxy_create_gset:NnnN
1363    {
1364      \__rawobjects_launch_deprecate:NN \proxy_create_gset:NnnN \proxy_create_gset:Nnn
1365      \object_create_gset:NVnnNN #1 \c_proxy_address_str { #2 }{ #3 }
1366        \c_object_global_str #4
1367    }
1368
```

(*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.

```
1369
1370  \cs_new_protected:Nn \proxy_push_member:nnn
1371    {
1372      \object_newconst_str:nnn { #1 }{ #2 _ type }{ #3 }
1373      \seq_gput_left:cn
1374        {
1375          \object_member_adr:nnn { #1 }{ varlist }{ seq }
1376        }
1377        { #2 }
1378    }
1379
1380  \cs_generate_variant:Nn \proxy_push_member:nnn { Vnn }
1381
```

*(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.

```
1382
1383 \cs_new_protected:Nn \proxy_push_embedded:nnn
1384   {
1385     \object_newconst_str:nnx { #1 }{ #2 _ proxy }{ #3 }
1386     \seq_gput_left:cn
1387       {
1388         \object_member_adr:nnn { #1 }{ objlist }{ seq }
1389       }
1390       { #2 }
1391   }
1392
1393 \cs_generate_variant:Nn \proxy_push_embedded:nnn { Vnn }
1394
```

*(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.

```
1395
1396 \cs_new_protected:Nn \proxy_add_initializer:nN
1397   {
1398     \tl_gput_right:cn
1399       {
1400         \object_member_adr:nnn { #1 }{ init }{ tl }
1401       }
1402       { #2 }
1403   }
1404
1405 \cs_generate_variant:Nn \proxy_add_initializer:nN { VN }
1406
```

*(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.

```
1407
1408 \str_const:Nx \c_proxy_address_str
1409   { \object_address:nn { rawobjects }{ proxy } }
1410
1411 \object_newconst_str:nnn
1412   {
1413     \object_embedded_adr:Vn \c_proxy_address_str { /_I_/ }
1414   }
1415   { M }{ rawobjects }
1416
1417 \object_newconst_str:nnV
1418   {
1419     \object_embedded_adr:Vn \c_proxy_address_str { /_I_/ }
1420   }
1421   { P } \c_proxy_address_str
1422
1423 \object_newconst_str:nnV
1424   {
```

```
1425        \object_embedded_adr:Vn \c_proxy_address_str { /_I_/ }
1426    }
1427    { S } \c_object_global_str
1428
1429 \object_newconst_str:nnV
1430    {
1431        \object_embedded_adr:Vn \c_proxy_address_str { /_I_/ }
1432    }
1433    { V } \c_object_public_str
1434
1435
1436 \__rawobjects_initproxy:VnV \c_proxy_address_str { rawobjects } \c_proxy_address_str
1437
1438 \object_new_member:Vnn \c_proxy_address_str { init }{ tl }
1439
1440 \object_new_member:Vnn \c_proxy_address_str { varlist }{ seq }
1441
1442 \object_new_member:Vnn \c_proxy_address_str { objlist }{ seq }
1443
1444 \proxy_push_member:Vnn \c_proxy_address_str
1445    { init }{ tl }
1446 \proxy_push_member:Vnn \c_proxy_address_str
1447    { varlist }{ seq }
1448 \proxy_push_member:Vnn \c_proxy_address_str
1449    { objlist }{ seq }
1450
1451 \proxy_add_initializer:VN \c_proxy_address_str
1452    \__rawobjects_initproxy:nnn
1453
```

(*End definition for* `\c_proxy_address_str`. *This variable is documented on page 12.*)

**\object_allocate_incr:NNnnNN**
\object_gallocate_incr:NNnnNN
\object_allocate_gincr:NNnnNN
\object_gallocate_gincr:NNnnNN

Create an address and use it to instantiate an object

```
1454
1455 \cs_new:Nn \__rawobjects_combine_aux:nnn
1456    {
1457        anon . #3 . #2 . #1
1458    }
1459
1460 \cs_generate_variant:Nn \__rawobjects_combine_aux:nnn { Vnf }
1461
1462 \cs_new:Nn \__rawobjects_combine:Nn
1463    {
1464        \__rawobjects_combine_aux:Vnf #1 { #2 }
1465    {
1466        \cs_to_str:N #1
1467    }
1468    }
1469
1470 \cs_new_protected:Nn \object_allocate_incr:NNnnNN
1471    {
1472        \object_create_set:NnnfNN #1 { #3 }{ #4 }
1473            {
1474                \__rawobjects_combine:Nn #2 { #3 }
```

```
1475          }
1476         #5 #6

1477
1478        \int_incr:N #2
1479      }

1480
1481  \cs_new_protected:Nn \object_gallocate_incr:NNnnNN
1482    {
1483       \object_create_gset:NnnfNN #1 { #3 }{ #4 }
1484         {
1485           \__rawobjects_combine:Nn #2 { #3 }
1486         }
1487         #5 #6

1488
1489        \int_incr:N #2
1490      }

1491
1492  \cs_generate_variant:Nn \object_allocate_incr:NNnnNN { NNVnNN }

1493
1494  \cs_generate_variant:Nn \object_gallocate_incr:NNnnNN { NNVnNN }

1495
1496  \cs_new_protected:Nn \object_allocate_gincr:NNnnNN
1497    {
1498       \object_create_set:NnnfNN #1 { #3 }{ #4 }
1499         {
1500           \__rawobjects_combine:Nn #2 { #3 }
1501         }
1502         #5 #6

1503
1504        \int_gincr:N #2
1505      }

1506
1507  \cs_new_protected:Nn \object_gallocate_gincr:NNnnNN
1508    {
1509       \object_create_gset:NnnfNN #1 { #3 }{ #4 }
1510         {
1511           \__rawobjects_combine:Nn #2 { #3 }
1512         }
1513         #5 #6

1514
1515        \int_gincr:N #2
1516      }

1517
1518  \cs_generate_variant:Nn \object_allocate_gincr:NNnnNN { NNVnNN }

1519
1520  \cs_generate_variant:Nn \object_gallocate_gincr:NNnnNN { NNVnNN }

1521
```

(*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.

```
1522  \cs_new_protected:Nn \object_assign:nn
1523    {
```

```
1524      \seq_map_inline:cn
1525        {
1526          \object_member_adr:vnn
1527            {
1528              \object_ncmember_adr:nnn
1529                {
1530                  \object_embedded_adr:nn{ #1 }{ /_I_/ }
1531                }
1532                { P }{ str }
1533            }
1534            { varlist }{ seq }
1535        }
1536        {
1537          \object_member_set_eq:nnc { #1 }{ ##1 }
1538            {
1539              \object_member_adr:nn{ #2 }{ ##1 }
1540            }
1541        }
1542    }
1543
1544 \cs_generate_variant:Nn \object_assign:nn { nV, Vn, VV }
```

(*End definition for* \object_assign:nn. *This function is documented on page 14.*)

```
1545 ⟨/package⟩
```