The lt3rawobjects package

Paolo De Donato

Released on 2022/12/27 Version 2.3-beta-2

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 Instead changes introduced in minor and patch version updates are always backward compatible, and any withdrawn function is declared deprecated instead of being removed.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 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 `\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 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 ⟨*module*⟩ and an ⟨*identifier*⟩ as explained before. The use of underscore character in objects identifiers is reserved. You can retrive 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 identidied by a pure address. Here an item could be one of the following entities:

- a LaTeX3 variable, in which case the item is called *member*;
- a LaTeX3 constant, in which case the item is called just *constant*;
- a LaTeX3 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 LaTeX3 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.

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

4.2 Methods

Methods are LaTeX3 functions that can't be changed once they're created. Like constant, methods could be near or remote.

4.3 Members

Members are just mutable LaTeX3 variables. Members can be manually created inside objects or can be automatically created during object creation if their definitions are stored inside the generating proxy through the `\proxy_push_member` function. These members automatically created by proxies are called *tracked* since the generator knows about them. Moreover, you don't need to specify the type of a tracked member since it can be inferred from the generator and the `assign` functions automatically copy all the tracked members in the new object.

If the object is local/global then all its members are automatically local/global.

5 Put objects inside objects

Sometimes it's necessary to include other objects inside an object, and since objects are structured data types you can't put them directly inside a variable. However lt3rawobjects provides some workarounds that allows you to include objects inside other objects, each with its own advantages and disadvantages.

In the following examples we're in module `mymod` and we want to put inside object `A` another object created with proxy `prx`.

5.1 Put a pointer variable

A simple solution is creating that object outside `A` with `\object_create`

```
\object_create:nnnNN
  { \object_address:nn{ mymod }{ prx } }{ mymod }{ B } ....
```

and then creating a pointer variable inside `A` (usually of type `tl` or `str`) holding the newly created address:

```
\object_new_member:nnn
  {
    \object_address:nn{ mymod }{ A }
  }{ pointer }{ tl }

\tl_(g)set:cn
```

```
{
  \object_new_member:nnn
    {
      \object_address:nn{ mymod }{ A }
    }{ pointer }{ tl }
}
{
  \object_address:nn{ mymod }{ B }
}
```

you can the access the pointed object by calling `\object_member_use` on `pointer` member.

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;
- creating objects also creates additional hidden variables, taking so (little) additional space.

5.2 Clone the inner structure

Instead of referring a complete object you can just clone the inner structure of `prx` and put inside `A`. For example if `prx` declares member `x` of type `str` and member `y` of type `int` then you can do

```
\object_new_member:nnn
  {
    \object_address:nn{ mymod }{ A }
  }{ prx-x }{ str }
\object_new_member:nnn
  {
    \object_address:nn{ mymod }{ A }
  }{ prx-y }{ int }
```

and then use `prx-x`, `prx-y` as normal members of `A`.

Advantages

- Simple and no additional function needed to create and manage included objects;
- you can put these specifications inside a proxy so that every object created with it will have the required members/methods;
- no hidden variable created, lowest overhead among the proposed solutions.

Disadvantages

- Cloning the inner structure doesn't create any object, so you don't have any object address nor you can share the included "object" unless you share the container object too.

5.3 Embedded objects

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

```
\embedded_create:nnn
  {
    \object_address:nn{  mymod }{ A }
  }{ prx }{ B }
```

and addresses of emmbedded 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

`\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.

From: 1.0

\object_address_set:Nnn
\object_address_gset:Nnn

\object_address_set:nn ⟨*str var*⟩ {⟨*module*⟩} {⟨*id*⟩}

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

From: 1.1

\object_embedded_adr:nn ☆
\object_embedded_adr:Vn ☆

\object_embedded_adr:nn {⟨*address*⟩} {⟨*id*⟩}

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

\object_if_exist_p:n ⋆
\object_if_exist_p:V ⋆
\object_if_exist:nTF ⋆
\object_if_exist:VTF ⋆

\object_if_exist_p:n {⟨*address*⟩}
\object_if_exist:nTF {⟨*address*⟩} {⟨*true code*⟩} {⟨*false code*⟩}

Tests if an object was instantiated at the specified address.

From: 1.0

\object_get_module:n ⋆
\object_get_module:V ⋆
\object_get_proxy_adr:n ⋆
\object_get_proxy_adr:V ⋆

\object_get_module:n {⟨*address*⟩}
\object_get_proxy_adr:n {⟨*address*⟩}

Get the object module and its generator.

From: 1.0

\object_if_local_p:n ⋆
\object_if_local_p:V ⋆
\object_if_local:nTF ⋆
\object_if_local:VTF ⋆
\object_if_global_p:n ⋆
\object_if_global_p:V ⋆
\object_if_global:nTF ⋆
\object_if_global:VTF ⋆

\object_if_local_p:n {⟨*address*⟩}
\object_if_local:nTF {⟨*address*⟩} {⟨*true code*⟩} {⟨*false code*⟩}

Tests if the object is local or global.

From: 1.0

\object_if_public_p:n ⋆
\object_if_public_p:V ⋆
\object_if_public:nTF ⋆
\object_if_public:VTF ⋆
\object_if_private_p:n ⋆
\object_if_private_p:V ⋆
\object_if_private:nTF ⋆
\object_if_private:VTF ⋆

\object_if_public_p:n {⟨*address*⟩}
\object_if_public:nTF {⟨*address*⟩} {⟨*true code*⟩} {⟨*false code*⟩}

Tests if the object is public or private.

From: 1.0

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 type is not specified it'll be retrieved from the generator proxy, but only if member is specified in the generator.

From: 1.0

\object_member_if_exist_p:nnn ⋆ \object_member_if_exist_p:nnn {⟨address⟩} {⟨member name⟩} {⟨member type⟩}
\object_member_if_exist_p:Vnn ⋆
\object_member_if_exist:nnnTF ⋆ \object_member_if_exist:nnnTF {⟨address⟩} {⟨member name⟩} {⟨member type⟩} {⟨true code⟩} {⟨false code⟩}
\object_member_if_exist:VnnTF ⋆
\object_member_if_exist_p:nn ⋆ \object_member_if_exist_p:nn {⟨address⟩} {⟨member name⟩}
\object_member_if_exist_p:Vn ⋆ \object_member_if_exist:nnTF {⟨address⟩} {⟨member name⟩} {⟨true code⟩} {⟨false code⟩}
\object_member_if_exist:nnTF ⋆
\object_member_if_exist:VnTF ⋆

Tests if the specified member exist.

From: 2.0

\object_member_type:nn ⋆ \object_member_type:nn {⟨address⟩} {⟨member name⟩}
\object_member_type:Vn ⋆

Fully expands to the type of member ⟨*member name*⟩. Use this function only with member variables specified in the generator proxy, not with other member variables.

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 variable with specified name and type. You can't retrieve the type of these variables with \object_member_type functions.

From: 1.0

\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⟩} {⟨value⟩}
\object_member_set:(nnvn|Vnnn)
\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*⟩} {⟨*member type*⟩} ⟨*variable*⟩
\object_member_set_eq:(nnvN|VnnN|nnnc|Vnnc)	
\object_member_set_eq:nnN	\object_member_set_eq:nnN {⟨*address*⟩} {⟨*member name*⟩} ⟨*variable*⟩
\object_member_set_eq:(VnN|nnc|Vnc)	

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

From: 1.0

\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 type*⟩}
\object_ncmember_if_exist_p:Vnn	⋆	
\object_ncmember_if_exist:nnn*TF*	⋆	\object_ncmember_if_exist:nnnTF {⟨*address*⟩} {⟨*member name*⟩} {⟨*member type*⟩} {⟨*true code*⟩} {⟨*false code*⟩}
\object_ncmember_if_exist:Vnn*TF*	⋆	
\object_rcmember_if_exist_p:nnn	⋆	
\object_rcmember_if_exist_p:Vnn	⋆	
\object_rcmember_if_exist:nnn*TF*	⋆	
\object_rcmember_if_exist:Vnn*TF*	⋆	

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	⋆	
\object_rcmember_use:Vnn	⋆	

Uses the specified near/remote constant member.

From: 2.0

6.4 Methods

Currentlu only constant methods (near and remote) are implemented in lt3rawobjects as explained before.

\object_ncmethod_adr:nnn	☆	\object_ncmethod_adr:nnn {⟨*address*⟩} {⟨*method name*⟩} {⟨*method variant*⟩}
\object_ncmethod_adr:(Vnn|vnn)	☆	
\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:Vnn ⋆
\object_ncmethod_if_exist:nnnTF ⋆
\object_ncmethod_if_exist:VnnTF ⋆
\object_rcmethod_if_exist_p:nnn ⋆
\object_rcmethod_if_exist_p:Vnn ⋆
\object_rcmethod_if_exist:nnnTF ⋆
\object_rcmethod_if_exist:VnnTF ⋆

\object_ncmethod_if_exist_p:nnn {⟨*address*⟩} {⟨*method name*⟩} {⟨*method variant*⟩}
\object_ncmethod_if_exist:nnnTF {⟨*address*⟩} {⟨*method name*⟩} {⟨*method variant*⟩} {⟨*true code*⟩} {⟨*false code*⟩}

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:Vnn ⋆
\object_rcmethod_call:nnn ⋆
\object_rcmethod_call:Vnn ⋆

\object_ncmethod_call:nnn {⟨*address*⟩} {⟨*method name*⟩} {⟨*method variant*⟩}

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

From: 2.0

6.5 Constant member creation

Unlike normal variables, constant variables in LaTeX3 are created in different ways depending on the specified type. So we dedicate a new section only to collect some of these fuinctions readapted for near constants (remote constants are simply near constants created on the generator proxy).

\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:Vnn

```
\object_newconst_seq_from_clist:nnn {⟨address⟩} {⟨constant name⟩}
{⟨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:Vnn

```
\object_newconst_prop_from_keyval:nnn {⟨address⟩} {⟨constant
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⟩}
```

Expands to `\⟨type⟩_const:cn {⟨address⟩} {⟨value⟩}`, use it if you need to create simple constants with custom types.

From: 2.1

6.6 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.

6.7 Proxy utilities and object creation

\object_if_proxy_p:n ⋆
\object_if_proxy_p:V ⋆
\object_if_proxy:nTF ⋆
\object_if_proxy:VTF ⋆

```
\object_if_proxy_p:n {⟨address⟩}
\object_if_proxy:nTF {⟨address⟩} {⟨true code⟩} {⟨false code⟩}
```

Test if the specified object is a proxy object.

From: 1.0

\object_test_proxy_p:nn ⋆
\object_test_proxy_p:Vn ⋆
\object_test_proxy:nnTF ⋆
\object_test_proxy:VnTF ⋆

\object_test_proxy_p:nn {⟨*object address*⟩} {⟨*proxy address*⟩}
\object_test_proxy:nnTF {⟨*object address*⟩} {⟨*proxy address*⟩} {⟨*true code*⟩} {⟨*false code*⟩}

Test if the specified object is generated by the selected proxy, where ⟨*proxy variable*⟩ is a string variable holding the proxy address.

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`.

From: 2.0

\object_test_proxy_p:nN ⋆
\object_test_proxy_p:VN ⋆
\object_test_proxy:nNTF ⋆
\object_test_proxy:VNTF ⋆

\object_test_proxy_p:nN {⟨*object address*⟩} ⟨*proxy variable*⟩
\object_test_proxy:nNTF {⟨*object address*⟩} ⟨*proxy variable*⟩ {⟨*true code*⟩} {⟨*false code*⟩}

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 safetly used with older compilers.

From: 2.0

\c_proxy_address_str

The address of the `proxy` object in the `rawobjects` module.

From: 1.0

\object_create:nnnNN
\object_create:VnnNN

\object_create:nnnNN {⟨*proxy address*⟩} {⟨*module*⟩} {⟨*id*⟩} ⟨*scope*⟩ ⟨*visibility*⟩

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`).

From: 1.0

\object_create:nnnN
\object_create:VnnN
\object_create:nnn
\object_create:Vnn

\object_create:nnnN {⟨*proxy address*⟩} {⟨*module*⟩} {⟨*id*⟩} ⟨*scope*⟩
\object_create:nnn {⟨*proxy address*⟩} {⟨*module*⟩} {⟨*id*⟩}

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.

From: 2.3

\embedded_create:nnn
\embedded_create:(Vnn|nvn)

\embedded_create:nnn {⟨*parent object*⟩} {⟨*proxy address*⟩} {⟨*id*⟩}

Creates an embedded object with name ⟨*id*⟩ inside ⟨*parent object*⟩.

From: 2.2

\c_object_local_str
\c_object_global_str

Possible values for ⟨*scope*⟩ parameter.

From: 1.0

\c_object_public_str
\c_object_private_str

Possible values for ⟨*visibility*⟩ parameter.

From: 1.0

\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*⟩.

From: 1.0

\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.

From: 1.1

\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`.

From: 1.0
Deprecated in: 2.3

\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.

From: 2.3

\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.

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 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`.

```
\str_new:N \g_myproxy_str
\proxy_create_gset:Nnn \g_myproxy_str { example }{ myproxy }
\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.

```
\str_new:N \g_myobj_str
\object_create_gset:NVnn \g_myobj_str \g_myproxy_str
  { example }{ myobj }
\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 $

If you don't want to specify an object identifier you can also do

```
\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:

```
\object_member_set:nnn
  {
    \object_embedded_adr:Vn \g_EXTobj_str { emb }
  }{ var }{ Hi }
\object_member_use:nn
  {
    \object_embedded_adr:Vn \g_EXTobj_str { emb }
  }{ var }
```

Output: Hi

8 Implementation

```
⟨*package⟩
⟨@@=rawobjects⟩
```

Deprecation message

```
\msg_new:nnn { rawobjects }{ deprecate }
  {
    Command ~ #1 ~ is ~ deprecated. ~ Use ~ instead ~ #2
  }

\cs_new_protected:Nn \__rawobjects_launch_deprecate:NN
  {
    \msg_warning:nnnn{ rawobjects }{ deprecate }{ #1 }{ #2 }
  }
```

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

```
\cs_new:Nn \rwobj_address_f:n
  {
    \exp_args:Nc \cs_to_str:N { #1 }
  }
```

(*End definition for* `\rwobj_address_f:n`*. This function is documented on page* 6.)

`\c_object_local_str`
`\c_object_global_str`
`\c_object_public_str`
`\c_object_private_str`

```
\str_const:Nn \c_object_local_str {l}
\str_const:Nn \c_object_global_str {g}
\str_const:Nn \c_object_public_str {_}
\str_const:Nn \c_object_private_str {__}


\cs_new:Nn \__rawobjects_scope:N
  {
    \str_use:N #1
  }

\cs_new:Nn \__rawobjects_scope_pfx:N
  {
    \str_if_eq:NNF #1 \c_object_local_str
      { g }
  }

\cs_generate_variant:Nn \__rawobjects_scope_pfx:N { c }

\cs_new:Nn \__rawobjects_scope_pfx_cl:n
  {
    \__rawobjects_scope_pfx:c{
  \object_ncmember_adr:nnn
    {
  \object_embedded_adr:nn { #1  }{ /_I_/ }
}
```

```
{ S }{ str }
}
  }

\cs_new:Nn \__rawobjects_vis_var:N
  {
    \str_use:N #1
  }

\cs_new:Nn \__rawobjects_vis_fun:N
  {
    \str_if_eq:NNT #1 \c_object_private_str
      {
        __
      }
  }

```

(*End definition for* `\c_object_local_str` *and others. These variables are documented on page* 12.)

`\object_address:nn` Get address of an object

```
\cs_new:Nn \object_address:nn {
  \tl_to_str:n { #1  _  #2 }
}
```

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

`\object_embedded_adr:nn` Address of embedded object

```

\cs_new:Nn \object_embedded_adr:nn
  {
    #1 \tl_to_str:n{ _SUB_ #2 }
  }

\cs_generate_variant:Nn \object_embedded_adr:nn{ Vn }

```

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

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

```

\cs_new_protected:Nn \object_address_set:Nnn {
  \str_set:Nn #1 { #2  _  #3 }
}

\cs_new_protected:Nn \object_address_gset:Nnn {
  \str_gset:Nn #1 { #2  _  #3 }
}

```

(*End definition for* `\object_address_set:Nnn` *and* `\object_address_gset:Nnn`*. These functions are documented on page* 7.)

\object_if_exist_p:n
\object_if_exist:nTF

Tests if object exists.

```
\prg_new_conditional:Nnn \object_if_exist:n { p, T, F, TF }
  {
    \cs_if_exist:cTF
      {
        \object_ncmember_adr:nnn
          {
            \object_embedded_adr:nn{ #1 }{ /_I_/ }
          }
          { S }{ str }
      }
      {
        \prg_return_true:
      }
      {
        \prg_return_false:
      }
  }

\prg_generate_conditional_variant:Nnn \object_if_exist:n { V }
  { p, T, F, TF }
```

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

\object_get_module:n
\object_get_proxy_adr:n

Retrieve the name, module and generating proxy of an object

```
\cs_new:Nn \object_get_module:n {
  \object_ncmember_use:nnn
  {
    \object_embedded_adr:nn{ #1 }{ /_I_/ }
  }
  { M }{ str }
}
\cs_new:Nn \object_get_proxy_adr:n {
  \object_ncmember_use:nnn
  {
    \object_embedded_adr:nn{ #1 }{ /_I_/ }
  }
  { P }{ str }
}

\cs_generate_variant:Nn \object_get_module:n { V }
\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
\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.

```
\prg_new_conditional:Nnn \object_if_local:n {p, T, F, TF}
{
  \str_if_eq:cNTF
    {
      \object_ncmember_adr:nnn
```

```
        {
          \object_embedded_adr:nn{ #1 }{ /_I_/ }
        }
        { S }{ str }
    }
    \c_object_local_str
    {
      \prg_return_true:
    }
    {
      \prg_return_false:
    }
}

\prg_new_conditional:Nnn \object_if_global:n {p, T, F, TF}
{
  \str_if_eq:cNTF
    {
      \object_ncmember_adr:nnn
        {
          \object_embedded_adr:nn{ #1 }{ /_I_/ }
        }
        { S }{ str }
    }
    \c_object_global_str
    {
      \prg_return_true:
    }
    {
      \prg_return_false:
    }
}

\prg_new_conditional:Nnn \object_if_public:n {p, T, F, TF}
{
  \str_if_eq:cNTF
    {
      \object_ncmember_adr:nnn
        {
          \object_embedded_adr:nn{ #1 }{ /_I_/ }
        }
        { V }{ str }
    }
    \c_object_public_str
    {
      \prg_return_true:
    }
    {
      \prg_return_false:
    }
}

\prg_new_conditional:Nnn \object_if_private:n {p, T, F, TF}
{
```

```
  \str_if_eq:cNTF
    {
      \object_ncmember_adr:nnn
        {
          \object_embedded_adr:nn{ #1 }{ /_I_/ }
        }
        { V }{ str }
    }
    \c_object_private_str
    {
      \prg_return_true:
    }
    {
      \prg_return_false:
    }
}

\prg_generate_conditional_variant:Nnn \object_if_local:n { V }
  { p, T, F, TF }
\prg_generate_conditional_variant:Nnn \object_if_global:n { V }
  { p, T, F, TF }
\prg_generate_conditional_variant:Nnn \object_if_public:n { V }
  { p, T, F, TF }
\prg_generate_conditional_variant:Nnn \object_if_private:n { V }
  { p, T, F, TF }
```

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

`\object_macro_adr:nn`
`\object_macro_use:nn`

Generic macro address

```

\cs_new:Nn \object_macro_adr:nn
  {
    #1 \tl_to_str:n{ _MACRO_ #2 }
  }

\cs_generate_variant:Nn \object_macro_adr:nn{ Vn }

\cs_new:Nn \object_macro_use:nn
  {
    \use:c
      {
        \object_macro_adr:nn{ #1 }{ #2 }
      }
  }

\cs_generate_variant:Nn \object_macro_use:nn{ Vn }

```

(*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

```

\cs_new:Nn \__rawobjects_member_adr:nnnNN
  {
```

```
    \__rawobjects_scope:N #4
    \__rawobjects_vis_var:N #5
    #1 \tl_to_str:n { _ MEMBER _ #2 _ #3 }
  }

\cs_generate_variant:Nn \__rawobjects_member_adr:nnnNN { VnnNN, nnncc }
```

(*End definition for* `\__rawobjects_member_adr:nnnNN`.)

`\object_member_adr:nnn`
`\object_member_adr:nn`

Get the address of a member variable

```
\cs_new:Nn \object_member_adr:nnn
  {
    \__rawobjects_member_adr:nnncc { #1 }{ #2 }{ #3 }
      {
        \object_ncmember_adr:nnn
          {
            \object_embedded_adr:nn{ #1 }{ /_I_/ }
          }
          { S }{ str }
      }
      {
        \object_ncmember_adr:nnn
          {
            \object_embedded_adr:nn{ #1 }{ /_I_/ }
          }
          { V }{ str }
      }
  }

\cs_generate_variant:Nn \object_member_adr:nnn { Vnn, vnn, nnv }

\cs_new:Nn \object_member_adr:nn
  {
    \object_member_adr:nnv { #1 }{ #2 }
      {
        \object_rcmember_adr:nnn { #1 }
          { #2 _ type }{ str }
      }
  }

\cs_generate_variant:Nn \object_member_adr:nn { Vn }
```

(*End definition for* `\object_member_adr:nnn` *and* `\object_member_adr:nn`. *These functions are 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.

```
\cs_new_protected:Nn \__rawobjects_generator_mem:nnn
  {
```

```
    \cs_new:cn
      {
        rwobj-aux_ #1 : nn
      }
      {
        \use:c{ #2 : c #3 }
      }
    \cs_new:cpn {#1 : nnn #3} ##1##2##3
      {
        \use:c
          {
            rwobj-aux_ #1 : nn
          }
          { ##3 }
          {
            \__rawobjects_scope_pfx_cl:n{ ##1 }
          }
          {
            \object_member_adr:nnn{ ##1 }{ ##2 }{ ##3 }
          }
      }
    \cs_generate_variant:cn { #1 : nnn #3 }{ Vnn #3, nnv #3 }

    \cs_new:cpn { #1 : nn #3 } ##1##2
      {
        \use:c{ #1 : nnv #3 }
          { ##1 }{ ##2 }
          {
            \object_rcmember_adr:nnn
              { ##1 }{ ##2 _ type }{ str }
          }
      }

    \cs_generate_variant:cn { #1 : nn #3 }{ Vn #3 }
  }


\msg_new:nnn{ rawobjects }{ nonew }{ Unknown ~ function ~ #1 }

\cs_new_protected:Nn \__rawobjects_generator_mem_protected:nnn
  {
    \cs_new_protected:cn
      {
        rwobj-aux_ #1 : nn
      }
      {
        \cs_if_exist_use:cF{ #2 : c #3 }
        {
          \msg_error:nnx{ rawobjects }{ nonew }{ #2 :c #3 }
        }
      }
    \cs_new_protected:cpn {#1 : nnn #3} ##1##2##3
      {
        \use:c
```

```
        {
          rwobj-aux_ #1 : nn
        }
        { ##3 }
        {
          \__rawobjects_scope_pfx_cl:n{ ##1 }
        }
        {
          \object_member_adr:nnn{ ##1 }{ ##2 }{ ##3 }
        }
    }
  \cs_generate_variant:cn { #1 : nnn #3 }{ Vnn #3, nnv #3 }

  \cs_new_protected:cpn { #1 : nn #3 } ##1##2
    {
      \use:c{ #1 : nnv #3 }
        { ##1 }{ ##2 }
        {
          \object_rcmember_adr:nnn
            { ##1 }{ ##2 _ type }{ str }
        }
    }

  \cs_generate_variant:cn { #1 : nn #3 }{ Vn #3 }
}

```

\object_member_type:nn Deduce the member type from the generating proxy.

```
\cs_new:Nn \object_member_type:nn
  {
    \object_rcmember_use:nnn { #1 }
      { #2 _ type }{ str }
  }

```

(*End definition for* `\object_member_type:nn`. *This function is documented on page* 8.)

```
\msg_new:nnnn { rawobjects }{ noerr }{ Unspecified ~ scope }
  {
    Object ~ #1 ~ hasn't ~ a ~ scope ~ variable
  }

\msg_new:nnnn { rawobjects }{ scoperr }{ Nonstandard ~ scope }
  {
    Operation ~ not ~ permitted ~ on ~ object ~ #1 ~
    ~ since ~ it ~ wasn't ~ declared ~ local ~ or ~ global
  }

\cs_new_protected:Nn \__rawobjects_force_scope:n
  {
    \cs_if_exist:cTF
      {
        \object_ncmember_adr:nnn
```

```
        {
          \object_embedded_adr:nn{ #1 }{ /_I_/ }
        }
        { S }{ str }
    }
    {
      \bool_if:nF
        {
          \object_if_local_p:n { #1 } || \object_if_global_p:n { #1 }
        }
        {
          \msg_error:nnx { rawobjects }{ scoperr }{ #1 }
        }
    }
    {
      \msg_error:nnx { rawobjects }{ noerr }{ #1 }
    }
}

```

\object_member_if_exist_p:nnn
\object_member_if_exist:nnnTF
\object_member_if_exist_p:nn
\object_member_if_exist:nnTF

Tests if the specified member exists

```

\prg_new_conditional:Nnn \object_member_if_exist:nnn {p, T, F, TF }
  {
    \cs_if_exist:cTF
      {
        \object_member_adr:nnn { #1 }{ #2 }{ #3 }
      }
      {
        \prg_return_true:
      }
      {
        \prg_return_false:
      }
  }

\prg_new_conditional:Nnn \object_member_if_exist:nn {p, T, F, TF }
  {
    \cs_if_exist:cTF
      {
        \object_member_adr:nn { #1 }{ #2 }
      }
      {
        \prg_return_true:
      }
      {
        \prg_return_false:
      }
  }

\prg_generate_conditional_variant:Nnn \object_member_if_exist:nnn
  { Vnn }{ p, T, F, TF }
\prg_generate_conditional_variant:Nnn \object_member_if_exist:nn
  { Vn }{ p, T, F, TF }
```

```
426
```

(*End definition for* `\object_member_if_exist:nnnTF` *and* `\object_member_if_exist:nnTF`*. These functions are documented on page 8.*)

`\object_new_member:nnn` Creates a new member variable

```
427
428
429 \cs_new_protected:Nn \object_new_member:nnn
430   {
431     \cs_if_exist_use:cT { #3 _ new:c }
432       {
433         { \object_member_adr:nnn { #1 }{ #2 }{ #3 } }
434       }
435   }
436
437 \cs_generate_variant:Nn \object_new_member:nnn { Vnn, nnv }
438
```

(*End definition for* `\object_new_member:nnn`*. This function is documented on page 8.*)

`\object_member_use:nnn` `\object_member_use:nn` Uses a member variable

```
439
440 \__rawobjects_generator_mem:nnn {object_member_use}{ #1_use }{}
441
442 \cs_generate_variant:Nn \object_member_use:nnn {vnn}
443
```

(*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.

```
444
445 \__rawobjects_generator_mem:nnn {object_member_set}{ #1_#2 set }{n}
446
```

(*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.

```
447
448 \__rawobjects_generator_mem_protected:nnn { object_member_set_eq }{ #1 _ #2 set_eq }{ N }
449
450 \cs_generate_variant:Nn \object_member_set_eq:nnnN { nnnc, Vnnc }
451
452 \cs_generate_variant:Nn \object_member_set_eq:nnN { nnc, Vnc }
453
```

(*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

```
\cs_new:Nn \object_ncmember_adr:nnn
  {
    \tl_to_str:n{ c _ } #1 \tl_to_str:n { _ CONST _ #2 _ #3 }
  }

\cs_generate_variant:Nn \object_ncmember_adr:nnn { Vnn, vnn }
```

(*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.

```
\cs_new:Nn \object_rcmember_adr:nnn
  {
    \object_ncmember_adr:vnn
      {
        \object_ncmember_adr:nnn
          {
            \object_embedded_adr:nn{ #1 }{ /_I_/ }
          }
          { P }{ str }
      }
      { #2 }{ #3 }
  }

\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.

```
\cs_new_protected:Nn \__rawobjects_generator_ncmem:nnn
  {
    \cs_new:cn
      {
        rwobj-aux_ #1 : n
      }
      {
        \use:c{ #2 : c #3 }
      }
    \cs_new:cpn {#1 : nnn #3} ##1##2##3
      {
        \use:c
          {
            rwobj-aux_ #1 : n
          }
          { ##3 }
          {
            \object_ncmember_adr:nnn{ ##1 }{ ##2 }{ ##3 }
          }
      }
```

```
    \cs_generate_variant:cn { #1 : nnn #3 }{ Vnn #3 }
  }

\cs_new_protected:Nn \__rawobjects_generator_ncmem_protected:nnn
  {
    \cs_new_protected:cn
      {
        rwobj-aux_ #1 : n
      }
      {
        \cs_if_exist_use:cF{ #2 : c #3 }
          {
            \msg_error:nnx{ rawobjects }{ nonew }{ #2 :c #3 }
          }
      }
    \cs_new_protected:cpn {#1 : nnn #3} ##1##2##3
      {
        \use:c
          {
            rwobj-aux_ #1 : n
          }
          { ##3 }
          {
            \object_ncmember_adr:nnn{ ##1 }{ ##2 }{ ##3 }
          }
      }
    \cs_generate_variant:cn { #1 : nnn #3 }{ Vnn #3 }
  }

```

\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.

```

\prg_new_conditional:Nnn \object_ncmember_if_exist:nnn {p, T, F, TF }
  {
    \cs_if_exist:cTF
      {
        \object_ncmember_adr:nnn { #1 }{ #2 }{ #3 }
      }
      {
        \prg_return_true:
      }
      {
        \prg_return_false:
      }
  }

\prg_new_conditional:Nnn \object_rcmember_if_exist:nnn {p, T, F, TF }
  {
    \cs_if_exist:cTF
      {
        \object_rcmember_adr:nnn { #1 }{ #2 }{ #3 }
      }
      {
        \prg_return_true:
```

```
      }
      {
        \prg_return_false:
      }
  }

\prg_generate_conditional_variant:Nnn \object_ncmember_if_exist:nnn
  { Vnn }{ p, T, F, TF }
\prg_generate_conditional_variant:Nnn \object_rcmember_if_exist:nnn
  { Vnn }{ p, T, F, TF }
```

(*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.

```
\__rawobjects_generator_ncmem:nnn{ object_ncmember_use }{ #1_use}{}

\cs_new:Nn \object_rcmember_use:nnn
  {
    \cs_if_exist_use:cT { #3 _ use:c }
      {
        { \object_rcmember_adr:nnn { #1 }{ #2 }{ #3 } }
      }
  }

\cs_generate_variant:Nn \object_rcmember_use:nnn { Vnn }
```

(*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

```
\__rawobjects_generator_ncmem_protected:nnn { object_newconst }{ #1 _ const }{n}
```

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

`\object_newconst_tl:nnn` `\object_newconst_str:nnn` `\object_newconst_int:nnn` `\object_newconst_clist:nnn` `\object_newconst_dim:nnn` `\object_newconst_skip:nnn` `\object_newconst_fp:nnn` Create constants

```
\cs_new_protected:Nn \object_newconst_tl:nnn
  {
    \object_newconst:nnnn { #1 }{ #2 }{ tl }{ #3 }
  }
\cs_new_protected:Nn \object_newconst_str:nnn
  {
    \object_newconst:nnnn { #1 }{ #2 }{ str }{ #3 }
  }
\cs_new_protected:Nn \object_newconst_int:nnn
  {
    \object_newconst:nnnn { #1 }{ #2 }{ int }{ #3 }
  }
\cs_new_protected:Nn \object_newconst_clist:nnn
```

```
  {
    \object_newconst:nnnn { #1 }{ #2 }{ clist }{ #3 }
  }
\cs_new_protected:Nn \object_newconst_dim:nnn
  {
    \object_newconst:nnnn { #1 }{ #2 }{ dim }{ #3 }
  }
\cs_new_protected:Nn \object_newconst_skip:nnn
  {
    \object_newconst:nnnn { #1 }{ #2 }{ skip }{ #3 }
  }
\cs_new_protected:Nn \object_newconst_fp:nnn
  {
    \object_newconst:nnnn { #1 }{ #2 }{ fp }{ #3 }
  }

\cs_generate_variant:Nn \object_newconst_tl:nnn { Vnn }
\cs_generate_variant:Nn \object_newconst_str:nnn { Vnn }
\cs_generate_variant:Nn \object_newconst_int:nnn { Vnn }
\cs_generate_variant:Nn \object_newconst_clist:nnn { Vnn }
\cs_generate_variant:Nn \object_newconst_dim:nnn { Vnn }
\cs_generate_variant:Nn \object_newconst_skip:nnn { Vnn }
\cs_generate_variant:Nn \object_newconst_fp:nnn { Vnn }


\cs_generate_variant:Nn \object_newconst_str:nnn { nnx }
\cs_generate_variant:Nn \object_newconst_str:nnn { nnV }

```

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

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

```

\cs_new_protected:Nn \object_newconst_seq_from_clist:nnn
  {
    \seq_const_from_clist:cn
      {
        \object_ncmember_adr:nnn { #1 }{ #2 }{ seq }
      }
      { #3 }
  }

\cs_generate_variant:Nn \object_newconst_seq_from_clist:nnn { Vnn }

```

(*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.

```

\cs_new_protected:Nn \object_newconst_prop_from_keyval:nnn
  {
    \prop_const_from_keyval:cn
      {
        \object_ncmember_adr:nnn { #1 }{ #2 }{ prop }
      }
```

```
      { #3 }
  }

\cs_generate_variant:Nn \object_newconst_prop_from_keyval:nnn { Vnn }

```

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

`\object_ncmethod_adr:nnn`
`\object_rcmethod_adr:nnn`

Fully expands to the method address.

```

\cs_new:Nn \object_ncmethod_adr:nnn
  {
    #1 \tl_to_str:n { _ CMETHOD _ #2 : #3 }
  }

\cs_generate_variant:Nn \object_ncmethod_adr:nnn { Vnn , vnn }

\cs_new:Nn \object_rcmethod_adr:nnn
  {
    \object_ncmethod_adr:vnn
      {
        \object_ncmember_adr:nnn
          {
            \object_embedded_adr:nn{ #1 }{ /_I_/ }
          }
          { P }{ str }
      }
      { #2 }{ #3 }
  }

\cs_generate_variant:Nn \object_ncmethod_adr:nnn { Vnn , vnn }
\cs_generate_variant:Nn \object_rcmethod_adr:nnn { Vnn }

```

(*End definition for* `\object_ncmethod_adr:nnn` *and* `\object_rcmethod_adr:nnn`. *These functions are documented on page 9.*)

`\object_ncmethod_if_exist_p:nnn`
`\object_ncmethod_if_exist:nnnTF`
`\object_rcmethod_if_exist_p:nnn`
`\object_rcmethod_if_exist:nnnTF`

Tests if the specified member constant exists.

```

\prg_new_conditional:Nnn \object_ncmethod_if_exist:nnn {p, T, F, TF }
  {
    \cs_if_exist:cTF
      {
        \object_ncmethod_adr:nnn { #1 }{ #2 }{ #3 }
      }
      {
        \prg_return_true:
      }
      {
        \prg_return_false:
      }
  }

\prg_new_conditional:Nnn \object_rcmethod_if_exist:nnn {p, T, F, TF }
  {

```

```
    \cs_if_exist:cTF
      {
        \object_rcmethodr_adr:nnn { #1 }{ #2 }{ #3 }
      }
      {
        \prg_return_true:
      }
      {
        \prg_return_false:
      }
  }

\prg_generate_conditional_variant:Nnn \object_ncmethod_if_exist:nnn
  { Vnn }{ p, T, F, TF }
\prg_generate_conditional_variant:Nnn \object_rcmethod_if_exist:nnn
  { Vnn }{ p, T, F, TF }

```

(*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

```

\cs_new_protected:Nn \object_new_cmethod:nnnn
  {
    \cs_new:cn
  {
    \object_ncmethod_adr:nnn { #1 }{ #2 }{ #3 }
  }
  { #4 }
  }

\cs_generate_variant:Nn \object_new_cmethod:nnnn { Vnnn }

```

(*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.

```

\cs_new:Nn \object_ncmethod_call:nnn
  {
    \use:c
  {
    \object_ncmethod_adr:nnn { #1 }{ #2 }{ #3 }
  }
  }

\cs_new:Nn \object_rcmethod_call:nnn
  {
    \use:c
  {
    \object_rcmethod_adr:nnn { #1 }{ #2 }{ #3 }
  }
  }

```

```
\cs_generate_variant:Nn \object_ncmethod_call:nnn { Vnn }
\cs_generate_variant:Nn \object_rcmethod_call:nnn { Vnn }

```

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

```

\cs_new_protected:Nn \__rawobjects_initproxy:nnn
  {
    \object_newconst:nnnn
      {
        \object_embedded_adr:nn{ #3 }{ /_I_/ }
      }
      { ifprox }{ bool }{ \c_true_bool }
  }
\cs_generate_variant:Nn \__rawobjects_initproxy:nnn { VnV }

```

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

```

\cs_new:Nn \__rawobjects_bol_com:N
  {
    \cs_if_exist_p:N #1 && \bool_if_p:N #1
  }

\cs_generate_variant:Nn \__rawobjects_bol_com:N { c }

\prg_new_conditional:Nnn \object_if_proxy:n {p, T, F, TF}
  {
    \cs_if_exist:cTF
      {
        \object_ncmember_adr:nnn
          {
            \object_embedded_adr:nn{ #1 }{ /_I_/ }
          }
          { ifprox }{ bool }
      }
      {
        \bool_if:cTF
          {
            \object_ncmember_adr:nnn
              {
                \object_embedded_adr:nn{ #1 }{ /_I_/ }
              }
              { ifprox }{ bool }
          }
          {
            \prg_return_true:
          }
          {
            \prg_return_false:
          }
      }
      {
```

```
        \prg_return_false:
      }
  }
```

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

`\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.

```
\prg_generate_conditional_variant:Nnn \str_if_eq:nn { ve }{ TF }

\prg_new_conditional:Nnn \object_test_proxy:nn {p, T, F, TF}
  {
    \str_if_eq:veTF
      {
        \object_ncmember_adr:nnn
          {
            \object_embedded_adr:nn{ #1 }{ /_I_/ }
          }
          { P }{ str }
      }
  { #2 }
      {
        \prg_return_true:
      }
      {
        \prg_return_false:
      }
  }

\prg_new_conditional:Nnn \object_test_proxy:nN {p, T, F, TF}
  {
    \str_if_eq:cNTF
      {
        \object_ncmember_adr:nnn
          {
            \object_embedded_adr:nn{ #1 }{ /_I_/ }
          }
          { P }{ str }
      }
  #2
      {
        \prg_return_true:
      }
      {
        \prg_return_false:
      }
  }

\prg_generate_conditional_variant:Nnn \object_test_proxy:nn
  { Vn }{p, T, F, TF}
\prg_generate_conditional_variant:Nnn \object_test_proxy:nN
  { VN }{p, T, F, TF}
```

(*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.

```
\msg_new:nnnn { rawobjects }{ notproxy }{ Fake ~ proxy }
  {
    Object ~ #1 ~ is ~ not ~ a ~ proxy.
  }

\cs_new_protected:Nn \__rawobjects_force_proxy:n
  {
    \object_if_proxy:nF { #1 }
      {
        \msg_error:nnn { rawobjects }{ notproxy }{ #1 }
      }
  }

\cs_new_protected:Nn \__rawobjects_create_anon:nnnNN
  {
    \tl_if_empty:nF{ #1 }
    {

    \__rawobjects_force_proxy:n { #1 }


    \object_newconst_str:nnn
      {
        \object_embedded_adr:nn{ #3 }{ /_I_/ }
      }
      { M }{ #2 }
    \object_newconst_str:nnn
      {
        \object_embedded_adr:nn{ #3 }{ /_I_/ }
      }
      { P }{ #1 }
    \object_newconst_str:nnV
      {
        \object_embedded_adr:nn{ #3 }{ /_I_/ }
      }
      { S } #4
    \object_newconst_str:nnV
      {
        \object_embedded_adr:nn{ #3 }{ /_I_/ }
      }
      { V } #5

    \seq_map_inline:cn
      {
        \object_member_adr:nnn { #1 }{ varlist }{ seq }
      }
      {
        \object_new_member:nnv { #3 }{ ##1 }
          {
```

```
            \object_ncmember_adr:nnn { #1 }{ ##1 _ type }{ str }
          }
      }

    \seq_map_inline:cn
      {
        \object_member_adr:nnn { #1 }{ objlist }{ seq }
      }
      {
        \embedded_create:nvn
          { #3 }
          {
            \object_ncmember_adr:nnn { #1 }{ ##1 _ proxy }{ str }
          }
          { ##1 }
      }

    \tl_map_inline:cn
      {
        \object_member_adr:nnn { #1 }{ init }{ tl }
      }
      {
        ##1 { #1 }{ #2 }{ #3 }
      }

    }
  }

\cs_generate_variant:Nn \__rawobjects_create_anon:nnnNN { xnxNN, xvxcc }

\cs_new_protected:Nn \object_create:nnnNN
  {
    \__rawobjects_create_anon:xnxNN { #1 }{ #2 }
      { \object_address:nn { #2 }{ #3 } }
      #4 #5
  }

\cs_generate_variant:Nn \object_create:nnnNN { VnnNN }

\cs_new_protected:Nn \object_create_set:NnnnNN
  {
    \object_create:nnnNN { #2 }{ #3 }{ #4 } #5 #6
    \str_set:Nx #1 { \object_address:nn { #3 }{ #4 } }
  }

\cs_new_protected:Nn \object_create_gset:NnnnNN
  {
    \object_create:nnnNN { #2 }{ #3 }{ #4 } #5 #6
    \str_gset:Nx #1 { \object_address:nn { #3 }{ #4 } }
  }

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

```

```
933
934
935 \cs_new_protected:Nn \object_create:nnnN
936   {
937     \object_create:nnnNN { #1 }{ #2 }{ #3 } #4 \c_object_public_str
938   }
939
940 \cs_generate_variant:Nn \object_create:nnnN { VnnN }
941
942 \cs_new_protected:Nn \object_create_set:NnnnN
943   {
944     \object_create_set:NnnnNN #1 { #2 }{ #3 }{ #4 } #5 \c_object_public_str
945   }
946
947 \cs_new_protected:Nn \object_create_gset:NnnnN
948   {
949     \object_create_gset:NnnnNN #1 { #2 }{ #3 }{ #4 } #5 \c_object_public_str
950   }
951
952 \cs_generate_variant:Nn \object_create_set:NnnnN { NVnnN }
953 \cs_generate_variant:Nn \object_create_gset:NnnnN { NVnnN }
954
955 \cs_new_protected:Nn \object_create:nnn
956   {
957     \object_create:nnnNN { #1 }{ #2 }{ #3 }
958       \c_object_global_str \c_object_public_str
959   }
960
961 \cs_generate_variant:Nn \object_create:nnn { Vnn }
962
963 \cs_new_protected:Nn \object_create_set:Nnnn
964   {
965     \object_create_set:NnnnNN #1 { #2 }{ #3 }{ #4 }
966       \c_object_global_str \c_object_public_str
967   }
968
969 \cs_new_protected:Nn \object_create_gset:Nnnn
970   {
971     \object_create_gset:NnnnNN #1 { #2 }{ #3 }{ #4 }
972       \c_object_global_str \c_object_public_str
973   }
974
975 \cs_generate_variant:Nn \object_create_set:Nnnn { NVnn }
976 \cs_generate_variant:Nn \object_create_gset:Nnnn { NVnn }
977
978
979
980
981 \cs_new_protected:Nn \embedded_create:nnn
982   {
983     \__rawobjects_create_anon:xvxcc { #2 }
984   {
985         \object_ncmember_adr:nnn
986           {
```

```
            \object_embedded_adr:nn{ #1 }{ /_I_/ }
          }
          { M }{ str }
      }
      {
        \object_embedded_adr:nn
          { #1 }{ #3 }
      }
      {
        \object_ncmember_adr:nnn
          {
            \object_embedded_adr:nn{ #1 }{ /_I_/ }
          }
          { S }{ str }
      }
      {
        \object_ncmember_adr:nnn
          {
            \object_embedded_adr:nn{ #1 }{ /_I_/ }
          }
          { V }{ str }
      }
  }

\cs_generate_variant:Nn \embedded_create:nnn { nvn, Vnn }

```

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

`\proxy_create:nn`
`\proxy_create_set:Nnn`
`\proxy_create_gset:Nnn`

Creates a new proxy object

```

\cs_new_protected:Nn \proxy_create:nn
  {
    \object_create:VnnNN \c_proxy_address_str { #1 }{ #2 }
      \c_object_global_str \c_object_public_str
  }

\cs_new_protected:Nn \proxy_create_set:Nnn
  {
    \object_create_set:NVnnNN #1 \c_proxy_address_str { #2 }{ #3 }
      \c_object_global_str \c_object_public_str
  }

\cs_new_protected:Nn \proxy_create_gset:Nnn
  {
    \object_create_gset:NVnnNN #1 \c_proxy_address_str { #2 }{ #3 }
      \c_object_global_str \c_object_public_str
  }



\cs_new_protected:Nn \proxy_create:nnN
  {
    \__rawobjects_launch_deprecate:NN \proxy_create:nnN \proxy_create:nn
```

```
    \object_create:VnnNN \c_proxy_address_str { #1 }{ #2 }
      \c_object_global_str #3
  }

\cs_new_protected:Nn \proxy_create_set:NnnN
  {
    \__rawobjects_launch_deprecate:NN \proxy_create_set:NnnN \proxy_create_set:Nnn
    \object_create_set:NVnnNN #1 \c_proxy_address_str { #2 }{ #3 }
      \c_object_global_str #4
  }

\cs_new_protected:Nn \proxy_create_gset:NnnN
  {
    \__rawobjects_launch_deprecate:NN \proxy_create_gset:NnnN \proxy_create_gset:Nnn
    \object_create_gset:NVnnNN #1 \c_proxy_address_str { #2 }{ #3 }
      \c_object_global_str #4
  }

```

(*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.

```

\cs_new_protected:Nn \proxy_push_member:nnn
  {
    \object_newconst_str:nnn { #1 }{ #2 _ type }{ #3 }
    \seq_gput_left:cn
      {
        \object_member_adr:nnn { #1 }{ varlist }{ seq }
      }
      { #2 }
  }

\cs_generate_variant:Nn \proxy_push_member:nnn { Vnn }

```

(*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.

```

\cs_new_protected:Nn \proxy_push_embedded:nnn
  {
    \object_newconst_str:nnx { #1 }{ #2 _ proxy }{ #3 }
    \seq_gput_left:cn
      {
        \object_member_adr:nnn { #1 }{ objlist }{ seq }
      }
      { #2 }
  }

\cs_generate_variant:Nn \proxy_push_embedded:nnn { Vnn }

```

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

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

```
\cs_new_protected:Nn \proxy_add_initializer:nN
  {
    \tl_gput_right:cn
      {
        \object_member_adr:nnn { #1 }{ init }{ tl }
      }
      { #2 }
  }

\cs_generate_variant:Nn \proxy_add_initializer:nN { VN }
```

(*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.

```
\str_const:Nx \c_proxy_address_str
  { \object_address:nn { rawobjects }{ proxy } }

\object_newconst_str:nnn
  {
    \object_embedded_adr:Vn \c_proxy_address_str { /_I_/ }
  }
  { M }{ rawobjects }

\object_newconst_str:nnV
  {
    \object_embedded_adr:Vn \c_proxy_address_str { /_I_/ }
  }
  { P } \c_proxy_address_str

\object_newconst_str:nnV
  {
    \object_embedded_adr:Vn \c_proxy_address_str { /_I_/ }
  }
  { S } \c_object_global_str

\object_newconst_str:nnV
  {
    \object_embedded_adr:Vn \c_proxy_address_str { /_I_/ }
  }
  { V } \c_object_public_str


\__rawobjects_initproxy:VnV \c_proxy_address_str { rawobjects } \c_proxy_address_str

\object_new_member:Vnn \c_proxy_address_str { init }{ tl }

\object_new_member:Vnn \c_proxy_address_str { varlist }{ seq }

\object_new_member:Vnn \c_proxy_address_str { objlist }{ seq }
```

```
\proxy_push_member:Vnn \c_proxy_address_str
  { init }{ tl }
\proxy_push_member:Vnn \c_proxy_address_str
  { varlist }{ seq }
\proxy_push_member:Vnn \c_proxy_address_str
  { objlist }{ seq }

\proxy_add_initializer:VN \c_proxy_address_str
  \__rawobjects_initproxy:nnn

```

(*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

```

\cs_new:Nn \__rawobjects_combine_aux:nnn
  {
    anon . #3 . #2 . #1
  }

\cs_generate_variant:Nn \__rawobjects_combine_aux:nnn { Vnf }

\cs_new:Nn \__rawobjects_combine:Nn
  {
    \__rawobjects_combine_aux:Vnf #1 { #2 }
  {
    \cs_to_str:N #1
  }
  }

\cs_new_protected:Nn \object_allocate_incr:NNnnNN
  {
    \object_create_set:NnnfNN #1 { #3 }{ #4 }
      {
        \__rawobjects_combine:Nn #2 { #3 }
      }
      #5 #6

      \int_incr:N #2
  }

\cs_new_protected:Nn \object_gallocate_incr:NNnnNN
  {
    \object_create_gset:NnnfNN #1 { #3 }{ #4 }
      {
        \__rawobjects_combine:Nn #2 { #3 }
      }
      #5 #6

      \int_incr:N #2
  }

\cs_generate_variant:Nn \object_allocate_incr:NNnnNN { NNVnNN }

```

```
\cs_generate_variant:Nn \object_gallocate_incr:NNnnNN { NNVnNN }

\cs_new_protected:Nn \object_allocate_gincr:NNnnNN
  {
    \object_create_set:NnnfNN #1 { #3 }{ #4 }
      {
        \__rawobjects_combine:Nn #2 { #3 }
      }
      #5 #6

      \int_gincr:N #2
  }

\cs_new_protected:Nn \object_gallocate_gincr:NNnnNN
  {
    \object_create_gset:NnnfNN #1 { #3 }{ #4 }
      {
        \__rawobjects_combine:Nn #2 { #3 }
      }
      #5 #6

      \int_gincr:N #2
  }

\cs_generate_variant:Nn \object_allocate_gincr:NNnnNN { NNVnNN }

\cs_generate_variant:Nn \object_gallocate_gincr:NNnnNN { NNVnNN }
```

(*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.

```
\cs_new_protected:Nn \object_assign:nn
  {
    \seq_map_inline:cn
      {
        \object_member_adr:vnn
          {
            \object_ncmember_adr:nnn
              {
                \object_embedded_adr:nn{ #1 }{ /_I_/ }
              }
              { P }{ str }
          }
          { varlist }{ seq }
      }
      {
        \object_member_set_eq:nnc { #1 }{ ##1 }
          {
            \object_member_adr:nn{ #2 }{ ##1 }
          }
      }
  }
```

```
\cs_generate_variant:Nn \object_assign:nn { nV, Vn, VV }
```

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

⟨/package⟩