

The lt3rawobjects package

Paolo De Donato

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

Contents

1	Introduction	2
2	Addresses	2
3	Objects	3
4	Items	3
4.1	Constants	4
4.2	Methods	4
4.3	Members	4
5	Object members	4
5.1	Create a pointer member	4
5.2	Clone the inner structure	5
5.3	Embedded objects	6
6	Library functions	7
6.1	Common functions	7
6.2	Base object functions	7
6.3	Members	8
6.4	Constants	10
6.5	Methods	10
6.6	Creation of constants	11
6.7	Macros	12
6.8	Proxies and object creation	12
7	Examples	15
8	Implementation	18

1 Introduction

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

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 Instead changes introduced in minor and patch version updates are always backward compatible, and any withdrawn function is declared deprecated instead of being removed.

2 Addresses

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

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

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

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

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

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

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

A *pointer* is just a L^AT_EX3 string variable that holds a pure address. We don't enforce to use `str` or any special suffix to denote pointers so you're free to use `str` or a custom $\langle type \rangle$ as suffix for your pointers in order to distinguish between them according to their type.

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

3 Objects

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

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

4 Items

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

- a L^AT_EX3 variable, in which case the item is called *member*;
- a L^AT_EX3 constant, in which case the item is called just *constant*;
- a L^AT_EX3 function, in which case the item is called *method*;
- generic control sequences, in which case the item is called simply *macro*;
- an entire object, in which case the item is called *embedded object*.

Objects could be declared *local* or *global*. The only difference between a local and a global object is the scope of their members (that are L^AT_EX3 variables). You should always create global object unless you specifically need local members.

4.1 Constants

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

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

4.2 Methods

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

4.3 Members

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

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

5 Object members

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

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

5.1 Create a pointer member

We first create a new object from `PRX`

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

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

```

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

```

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

Advantages

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

Disadvantages

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

5.2 Clone the inner structure

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

```

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

```

```

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

```

Advantages

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

Disadvantages

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

5.3 Embedded objects

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

```

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

```

and addresses of 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

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

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

6.2 Base object functions

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

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

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

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

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

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

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

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

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

Get the object module and its generator.
From: 1.0

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

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

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

6.3 Members

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

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

From: 1.0

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

Tests if the specified member exist.

From: 2.0

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

Tests if the specified member exist and is tracked.

From: 2.3

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

From: 1.0

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

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

From: 1.0

<code>\object_new_member_tracked:nnn</code>		<code>\object_new_member_tracked:nnn {<address>} {<member name>} {<member type>}</code>
<code>\object_new_member_tracked:Vnn</code>		<code>type)}</code>

Creates a new tracked member.

From: 2.3

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

Uses the specified member variable.

From: 1.0

<code>\object_member_set:nnnn</code>	<code>\object_member_set:nnnn {⟨address⟩} {⟨member name⟩} {⟨member type⟩}</code>
<code>\object_member_set:(nnvn Vnnn)</code>	<code>{⟨value⟩}</code>
<code>\object_member_set:nnn</code>	<code>\object_member_set:nnn {⟨address⟩} {⟨member name⟩} {⟨value⟩}</code>
<code>\object_member_set:Vnn</code>	

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

From: 2.1

<code>\object_member_set_eq:nnnN</code>	<code>\object_member_set_eq:nnnN {⟨address⟩} {⟨member name⟩}</code>
<code>\object_member_set_eq:(nnvN VnnN nnnc Vnnc)</code>	<code>{⟨member type⟩} {⟨variable⟩}</code>
<code>\object_member_set_eq:nnN</code>	<code>\object_member_set_eq:nnN {⟨address⟩} {⟨member name⟩}</code>
<code>\object_member_set_eq:(VnN nnnc Vnc)</code>	<code>{⟨variable⟩}</code>

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

From: 1.0

<code>\object_member_generate:NN</code>	<code>\object_member_generate:NN \⟨name₁⟩ \⟨name₂⟩:⟨arg1⟩⟨args⟩</code>
<code>\object_member_generate_protected:NN</code>	

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

<code>\object_member_generate_inline:Nnn</code>	<code>\object_member_generate_inline:Nnn \⟨name₁⟩ {⟨name₂⟩}</code>
<code>\object_member_generate_protected_inline:Nnn</code>	<code>{⟨arg1⟩⟨args⟩}</code>

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

<code>\object_ncmper_adr:nnn</code>	☆	<code>\object_ncmper_adr:nnn {<address>} {<member name>} {<member type>}</code>
<code>\object_ncmper_adr:(Vnn vnn)</code>	☆	
<code>\object_rcmember_adr:nnn</code>	☆	
<code>\object_rcmember_adr:Vnn</code>	☆	

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

From: 2.0

<code>\object_ncmper_if_exist_p:nnn</code>	★	<code>\object_ncmper_if_exist_p:nnn {<address>} {<member name>} {<member type>}</code>
<code>\object_ncmper_if_exist_p:Vnn</code>	★	
<code>\object_ncmper_if_exist:nnnTF</code>	★	<code>\object_ncmper_if_exist:nnnTF {<address>} {<member name>} {<member type>}</code>
<code>\object_ncmper_if_exist:VnnTF</code>	★	<code>{<true code>} {<false code>}</code>
<code>\object_rcmember_if_exist_p:nnn</code>	★	
<code>\object_rcmember_if_exist_p:Vnn</code>	★	
<code>\object_rcmember_if_exist:nnnTF</code>	★	
<code>\object_rcmember_if_exist:VnnTF</code>	★	

Tests if the specified member constant exist.

From: 2.0

<code>\object_ncmper_use:nnn</code>	★	<code>\object_ncmper_use:nnn {<address>} {<member name>} {<member type>}</code>
<code>\object_ncmper_use:Vnn</code>	★	
<code>\object_rcmember_use:nnn</code>	★	Uses the specified near/remote constant member.
<code>\object_rcmember_use:Vnn</code>	★	

From: 2.0

<code>\object_ncmper_generate:NN</code>	<code>\object_ncmper_generate:NN \<name₁> \name₂:<arg1><args></code>
<code>\object_ncmper_protected_generate:NN</code>	

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

From: 2.3

<code>\object_ncmper_generate_inline:Nnn</code>	<code>\object_ncmper_generate_inline:Nnn \<name₁> {<name₂>}</code>
<code>\object_ncmper_protected_generate_inline:Nnn</code>	<code>{<arg1><args>}</code>

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

From: 2.3

6.5 Methods

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

Fully expands to the address of the specified

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

From: 2.0

<code>\object_ncmethod_if_exist_p:nnn</code>	<code>*</code>	<code>\object_ncmethod_if_exist_p:nnn</code>	<code>{\langle address \rangle}</code>	<code>{\langle method name \rangle}</code>	<code>{\langle method variant \rangle}</code>
<code>\object_ncmethod_if_exist_p:Vnn</code>	<code>*</code>				
<code>\object_ncmethod_if_exist:nnnTF</code>	<code>*</code>	<code>\object_ncmethod_if_exist:nnnTF</code>	<code>{\langle address \rangle}</code>	<code>{\langle method name \rangle}</code>	<code>{\langle method variant \rangle}</code>
<code>\object_ncmethod_if_exist:VnnTF</code>	<code>*</code>				
<code>\object_rcmethod_if_exist_p:nnn</code>	<code>*</code>				
<code>\object_rcmethod_if_exist_p:Vnn</code>	<code>*</code>				
<code>\object_rcmethod_if_exist:nnnTF</code>	<code>*</code>				
<code>\object_rcmethod_if_exist:VnnTF</code>	<code>*</code>				

Tests if the specified method constant exist.

From: 2.0

<code>\object_new_cmethod:nnnn</code>	<code>\object_new_cmethod:nnnn</code>	<code>{\langle address \rangle}</code>	<code>{\langle method name \rangle}</code>	<code>{\langle method arguments \rangle}</code>	<code>{\langle code \rangle}</code>
<code>\object_new_cmethod:Vnnn</code>					

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

From: 2.0

<code>\object_ncmethod_call:nnn</code>	<code>*</code>	<code>\object_ncmethod_call:nnn</code>	<code>{\langle address \rangle}</code>	<code>{\langle method name \rangle}</code>	<code>{\langle method variant \rangle}</code>
<code>\object_ncmethod_call:Vnn</code>	<code>*</code>				
<code>\object_rcmethod_call:nnn</code>	<code>*</code>				
<code>\object_rcmethod_call:Vnn</code>	<code>*</code>				

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

From: 2.0

6.6 Creation of constants

<code>\object_newconst_tl:nnn</code>	<code>\object_newconst_{\langle type \rangle}:nnn</code>	<code>{\langle address \rangle}</code>	<code>{\langle constant name \rangle}</code>	<code>{\langle value \rangle}</code>
<code>\object_newconst_tl:Vnn</code>				

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

From: 1.1

<code>\object_newconst_str:nnn</code>				
<code>\object_newconst_str:Vnn</code>				
<code>\object_newconst_int:nnn</code>				
<code>\object_newconst_int:Vnn</code>				
<code>\object_newconst_clist:nnn</code>				
<code>\object_newconst_clist:Vnn</code>				
<code>\object_newconst_dim:nnn</code>				
<code>\object_newconst_dim:Vnn</code>				
<code>\object_newconst_skip:nnn</code>				
<code>\object_newconst_skip:Vnn</code>				
<code>\object_newconst_fp:nnn</code>				
<code>\object_newconst_fp:Vnn</code>				

<code>\object_newconst_seq_from_clist:nnn</code>	<code>\object_newconst_seq_from_clist:nnn</code>	<code>{\langle address \rangle}</code>	<code>{\langle constant name \rangle}</code>
<code>\object_newconst_seq_from_clist:Vnn</code>		<code>{\langle comma-list \rangle}</code>	

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

From: 1.1

<hr/>	
<code>\object_newconst_prop_from_keyval:nnn</code>	<code>\object_newconst_prop_from_keyval:nnn {⟨address⟩} {⟨constant name⟩}</code>
<code>\object_newconst_prop_from_keyval:Vnn</code>	<code>{</code> <code> ⟨key⟩ = ⟨value⟩, ...</code> <code>}</code>

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

From: 1.1

<hr/>	
<code>\object_newconst:nnnn</code>	<code>\object_newconst:nnnn {⟨address⟩} {⟨constant name⟩} {⟨type⟩} {⟨value⟩}</code>

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

From: 2.1

6.7 Macros

<hr/>	
<code>\object_macro_adr:nn</code> ☆	<code>\object_macro_adr:nn {⟨address⟩} {⟨macro name⟩}</code>

<code>\object_macro_adr:Vn</code> ☆	Address of specified macro.
-------------------------------------	-----------------------------

From: 2.2

<hr/>	
<code>\object_macro_use:nn</code> ☆	<code>\object_macro_use:nn {⟨address⟩} {⟨macro name⟩}</code>

<code>\object_macro_use:Vn</code> ☆	Uses the specified macro. This function is expandable if and only if the specified macro is it.
-------------------------------------	---

From: 2.2

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

6.8 Proxies and object creation

<hr/>	
<code>\object_if_proxy_p:n</code> ☆	<code>\object_if_proxy_p:n {⟨address⟩}</code>

<code>\object_if_proxy_p:V</code> ☆	<code>\object_if_proxy:nTF {⟨address⟩} {⟨true code⟩} {⟨false code⟩}</code>
-------------------------------------	--

<code>\object_if_proxy:nTF</code> ☆	Test if the specified object is a proxy object.
-------------------------------------	---

<code>\object_if_proxy:VTF</code> ☆	From: 1.0
-------------------------------------	-----------

<hr/>	
<code>\object_test_proxy_p:nn</code> ☆	<code>\object_test_proxy_p:nn {⟨object address⟩} {⟨proxy address⟩}</code>

<code>\object_test_proxy_p:Vn</code> ☆	<code>\object_test_proxy:nnTF {⟨object address⟩} {⟨proxy address⟩} {⟨true code⟩} {⟨false code⟩}</code>
--	--

<code>\object_test_proxy:nnTF</code> ☆	
--	--

<code>\object_test_proxy:VnTF</code> ☆	Test if the specified object is generated by the selected proxy, where <i>⟨proxy variable⟩</i> is a string variable holding the proxy address.
--	--

TeXhackers note: Remember that this command uses internally an **e** expansion so in older engines (any different from Lua^ATeX before 2019) it'll require slow processing. Don't use it in speed critical parts, instead use `\object_test_proxy:nN`.

From: 2.0

<code>\object_test_proxy_p:nN</code> *	<code>\object_test_proxy_p:nN</code> { <i><object address></i> } <i><proxy variable></i>
<code>\object_test_proxy_p:VN</code> *	<code>\object_test_proxy:nNTF</code> { <i><object address></i> } <i><proxy variable></i> { <i><true code></i> } { <i><false code></i> }
<code>\object_test_proxy:nNTF</code> *	
<code>\object_test_proxy:VNTF</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 safely used with older compilers.

From: 2.0

<code>\c_proxy_address_str</code>	The address of the proxy object in the <code>rawobjects</code> module.
-----------------------------------	--

From: 1.0

<code>\object_create:nnnNN</code> <code>\object_create:VnnNN</code>	<code>\object_create:nnnNN</code> { <i><proxy address></i> } { <i><module></i> } { <i><id></i> } <i><scope></i> <i><visibility></i> Creates an object by using the proxy at <i><proxy address></i> and the specified parameters. Use this function only if you need to create private objects (at present private objects are functionally equivalent to public objects) or if you need to compile your project with an old version of this library (< 2.3).
--	---

From: 1.0

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

From: 2.3

<code>\embedded_create:nnn</code> <code>\embedded_create:(Vnn nvn)</code>	<code>\embedded_create:nnn</code> { <i><parent object></i> } { <i><proxy address></i> } { <i><id></i> } Creates an embedded object with name <i><id></i> inside <i><parent object></i> .
--	---

From: 2.2

<code>\c_object_local_str</code> <code>\c_object_global_str</code>	Possible values for <i><scope></i> parameter. From: 1.0
---	--

<code>\c_object_public_str</code> <code>\c_object_private_str</code>	Possible values for <i><visibility></i> parameter. From: 1.0
---	---

<code>\object_create_set:NnnnNN</code> <code>\object_create_set:(NVnnNN NnnfNN)</code> <code>\object_create_gset:NnnnNN</code> <code>\object_create_gset:(NVnnNN NnnfNN)</code>	<code>\object_create_set:NnnnNN</code> <i><str var></i> { <i><proxy address></i> } { <i><module></i> } { <i><id></i> } <i><scope></i> <i><visibility></i> Creates an object and sets its fully expanded address inside <i><str var></i> . From: 1.0
--	---

<code>\object_allocate_incr:NNnnNN</code>	<code>\object_allocate_incr:NNnnNN <str var> <int var> {<proxy address>}</code>
<code>\object_allocate_incr:NNVnNN</code>	<code>{<module>} <scope> <visibility></code>
<code>\object_gallocate_incr:NNnnNN</code>	
<code>\object_gallocate_incr:NNVnNN</code>	
<code>\object_allocate_gincr:NNnnNN</code>	
<code>\object_allocate_gincr:NNVnNN</code>	
<code>\object_gallocate_gincr:NNnnNN</code>	
<code>\object_gallocate_gincr:NNVnNN</code>	

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

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

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

From: 1.0

Deprecated in: 2.3

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

Creates a global public proxy object.

From: 2.3

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

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

From: 1.0

<code>\proxy_push_embedded:nnn</code>	<code>\proxy_push_embedded:nnn {<proxy address>} {<embedded object name>} {<embedded object proxy>}</code>
<code>\proxy_push_embedded:Vnn</code>	

Updates a proxy object with a new embedded object specification.

From: 2.2

<code>\proxy_add_initializer:nN</code>	<code>\proxy_add_initializer:nN {<proxy address>} <initializer></code>
<code>\proxy_add_initializer:VN</code>	

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 {\langle to address \rangle} {\langle from address \rangle}
```

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

From: 1.0

7 Examples

Example 1

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

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

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

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

Output: \$ dollar \$

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

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

Output: \$ dollar \$

Example 2

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

Internal proxy

```
1 \proxy_create:nn { mymod }{ INT }
2 \proxy_push_member:nnn
3 {
4   \object_address:nn { mymod }{ INT }
5 }{ var }{ t1 }
```

Container proxy

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

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

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

and use the embedded object in the following way:

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

Output: Hi

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 practical since we should write a lot of code each time. We can then use `\object_member_generate:NN` to define an auxiliary 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 value 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 occurrences of `#1` with `int`, thus it will call `\int_use:c`.

8 Implementation

```

1 <*package>
2 <@@=rawobjects>
3
4 Deprecation message
5
6 \msg_new:nnn { rawobjects }{ deprecate }
7 {
8   Command ~ #1 ~ is ~ deprecated. ~ Use ~ instead ~ #2
9 }
10
11 \cs_new_protected:Nn \__rawobjects_launch_deprecate:NN
12 {
13   \msg_warning:nnnn{ rawobjects }{ deprecate }{ #1 }{ #2 }
14 }
15
16 \rwobj_address_f:n It just performs a c expansion before passing it to \cs_to_str:N.
17
18 \cs_new:Nn \rwobj_address_f:n
19 {
20   \exp_args:Nc \cs_to_str:N { #1 }
21 }
22
23 (End definition for \rwobj_address_f:n. This function is documented on page 7.)
24
25
26 \c_object_local_str
27 \c_object_global_str
28 \c_object_public_str
29 \c_object_private_str
30
31 \str_const:Nn \c_object_local_str {l}
32 \str_const:Nn \c_object_global_str {g}
33 \str_const:Nn \c_object_public_str {_}
34 \str_const:Nn \c_object_private_str {__}
35
36
37 \cs_new:Nn \__rawobjects_scope:N
38 {
39   \str_use:N #1
40 }
41
42 \cs_new:Nn \__rawobjects_scope_pfx:N
43 {
44   \str_if_eq:NNF #1 \c_object_local_str
45   { g }
46 }
47
48 \cs_generate_variant:Nn \__rawobjects_scope_pfx:N { c }
49
50 \cs_new:Nn \__rawobjects_scope_pfx_cl:n
51 {
52   \__rawobjects_scope_pfx:c{
53     \object_ncmember_adr:nnn
54     {
55       \object_embedded_adr:nn { #1 }{ /_I_/ }
56     }
57   }
58 }

```

```

46 { S }{ str }
47 }
48 }
49
50 \cs_new:Nn \__rawobjects_vis_var:N
51 {
52   \str_use:N #1
53 }
54
55 \cs_new:Nn \__rawobjects_vis_fun:N
56 {
57   \str_if_eq:NNT #1 \c_object_private_str
58   {
59     --
60   }
61 }
62

```

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

`\object_address:nn` Get address of an object

```

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

```

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

`\object_embedded_adr:nn` Address of embedded object

```

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

```

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

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

`\object_address_gset:Nnn`

```

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

```

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

`\object_if_exist_p:n` Tests if object exists.

`\object_if_exist:nTF`

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

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

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

`\object_get_proxy_adr:n`

```
105 \cs_new:Nn \object_get_module:n {
106   \object_ncmember_use:nnn
107   {
108     \object_embedded_adr:nn{ #1 }{ /_I_/ }
109   }
110   { M }{ str }
111 }
112 \cs_new:Nn \object_get_proxy_adr:n {
113   \object_ncmember_use:nnn
114   {
115     \object_embedded_adr:nn{ #1 }{ /_I_/ }
116   }
117   { P }{ str }
118 }
119
120 \cs_generate_variant:Nn \object_get_module:n { V }
121 \cs_generate_variant:Nn \object_get_proxy_adr:n { V }
```

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

`\object_if_local_p:n` Test the specified parameters.

`\object_if_local:nTF`

`\object_if_global_p:n`

`\object_if_global:nTF`

`\object_if_public_p:n`

`\object_if_public:nTF`

`\object_if_private_p:n`

`\object_if_private:nTF`

```
122 \prg_new_conditional:Nnn \object_if_local:n {p, T, F, TF}
123 {
124   \str_if_eq:cNTF
125   {
126     \object_ncmember_adr:nnn
```

```

127         {
128             \object_embedded_adr:nn{ #1 }{ /_I_/ }
129         }
130         { S }{ str }
131     }
132     \c_object_local_str
133     {
134         \prg_return_true:
135     }
136     {
137         \prg_return_false:
138     }
139 }
140
141 \prg_new_conditional:Nnn \object_if_global:n {p, T, F, TF}
142 {
143     \str_if_eq:cNTF
144     {
145         \object_ncmember_adr:nnn
146         {
147             \object_embedded_adr:nn{ #1 }{ /_I_/ }
148         }
149         { S }{ str }
150     }
151     \c_object_global_str
152     {
153         \prg_return_true:
154     }
155     {
156         \prg_return_false:
157     }
158 }
159
160 \prg_new_conditional:Nnn \object_if_public:n {p, T, F, TF}
161 {
162     \str_if_eq:cNTF
163     {
164         \object_ncmember_adr:nnn
165         {
166             \object_embedded_adr:nn{ #1 }{ /_I_/ }
167         }
168         { V }{ str }
169     }
170     \c_object_public_str
171     {
172         \prg_return_true:
173     }
174     {
175         \prg_return_false:
176     }
177 }
178
179 \prg_new_conditional:Nnn \object_if_private:n {p, T, F, TF}
180 {

```

```

181  \str_if_eq:cNTF
182  {
183      \object_ncmember_adr:nnn
184      {
185          \object_embedded_adr:nn{ #1 }{ /_I_/ }
186      }
187      { V }{ str }
188  }
189  \c_object_private_str
190  {
191      \prg_return_true:
192  }
193  {
194      \prg_return_false:
195  }
196 }
197
198 \prg_generate_conditional_variant:Nnn \object_if_local:n { V }
199 { p, T, F, TF }
200 \prg_generate_conditional_variant:Nnn \object_if_global:n { V }
201 { p, T, F, TF }
202 \prg_generate_conditional_variant:Nnn \object_if_public:n { V }
203 { p, T, F, TF }
204 \prg_generate_conditional_variant:Nnn \object_if_private:n { V }
205 { p, T, F, TF }

```

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

`\object_macro_adr:nn` Generic macro address

```

\object_macro_use:nn
206
207 \cs_new:Nn \object_macro_adr:nn
208 {
209     #1 \tl_to_str:n{ _MACRO_ #2 }
210 }
211
212 \cs_generate_variant:Nn \object_macro_adr:nn{ Vn }
213
214 \cs_new:Nn \object_macro_use:nn
215 {
216     \use:c
217     {
218         \object_macro_adr:nn{ #1 }{ #2 }
219     }
220 }
221
222 \cs_generate_variant:Nn \object_macro_use:nn{ Vn }
223

```

(End definition for `\object_macro_adr:nn` and `\object_macro_use:nn`. These functions are documented on page 12.)

`__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, nnnf }
255

```

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

\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:nnTF`. This function is documented on page 8.)

```

\object_member_if_tracked_p:nn Tests if the member is tracked.
\object_member_if_tracked:nnTF
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 8.)

`\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 8.)

`\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 8.)

`\object_member_generate:NN` Generate member versions of specified functions.

```

\object_member_generate_inline:Nnn
\object_member_generate_protected:NN
\object_member_generate_protected_inline:Nnn
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_protected:Nn \__rawobjects_mgen:nN
400 {
401     \__rawobjects_save_fun:N #2
402     \cs_new:cpn { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
403     {
404         #2
405         {
406             \object_member_adr:nnn{ ##1 }{ ##2 }{ ##3 }
407         }
408     }
409     \cs_new:cpn { #1 : nn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2
410     {
411         #2
412         {
413             \object_member_adr:nn{ ##1 }{ ##2 }
414         }
415     }
416 }
417 \cs_new_protected:Nn \__rawobjects_mgen_pr:nN
418 {
419     \__rawobjects_save_fun:N #2
420     \cs_new_protected:cpn
421         { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3

```

```

422     {
423         #2
424         {
425             \object_member_adr:nnn{ ##1 }{ ##2 }{ ##3 }
426         }
427     }
428     \cs_new_protected:cpn
429     { #1 : nn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2
430     {
431         #2
432         {
433             \object_member_adr:nn{ ##1 }{ ##2 }
434         }
435     }
436 }
437
438 \cs_new_protected:Nn \__rawobjects_mgen:nnn
439 {
440     \__rawobjects_save_dat:n { #3 }
441
442     \cs_new:cpn { __rawobjects_auxfun_#1 :nn } ##1##2
443     {
444         \use:c{ #2 : #3 }
445     }
446     \cs_generate_variant:cn { __rawobjects_auxfun_#1 :nn }{ nf, ff }
447
448     \cs_new:cpn { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
449     {
450         \use:c { __rawobjects_auxfun_#1 :nf }
451         { ##3 }
452         {
453             \__rawobjects_scope_pfx_cl:n{ ##1 }
454         }
455         {
456             \object_member_adr:nnn{ ##1 }{ ##2 }{ ##3 }
457         }
458     }
459     \cs_new:cpn { #1 : nn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2
460     {
461         \use:c { __rawobjects_auxfun_#1 :ff }
462         {
463             \object_member_type:nn { ##1 }{ ##2 }
464         }
465         {
466             \__rawobjects_scope_pfx_cl:n{ ##1 }
467         }
468         {
469             \object_member_adr:nn{ ##1 }{ ##2 }
470         }
471     }
472 }
473 \cs_new_protected:Nn \__rawobjects_mgen_pr:nnn
474 {
475     \__rawobjects_save_dat:n { #3 }

```

```

476
477 \cs_new:cpn { __rawobjects_auxfun_#1 :nn } ##1##2
478 {
479   \use:c{ #2 : #3 }
480 }
481 \cs_generate_variant:cn { __rawobjects_auxfun_#1 :nn }{ nf, ff }
482
483 \cs_new_protected:cpn
484 { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
485 {
486   \use:c { __rawobjects_auxfun_#1 :nf }
487   { ##3 }
488   {
489     \__rawobjects_scope_pfx_cl:n{ ##1 }
490   }
491   {
492     \object_member_adr:nnn{ ##1 }{ ##2 }{ ##3 }
493   }
494 }
495 \cs_new_protected:cpn
496 { #1 : nn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2
497 {
498   \use:c { __rawobjects_auxfun_#1 :ff }
499   {
500     \object_member_type:nn { ##1 }{ ##2 }
501   }
502   {
503     \__rawobjects_scope_pfx_cl:n{ ##1 }
504   }
505   {
506     \object_member_adr:nn{ ##1 }{ ##2 }
507   }
508 }
509 }
510
511 \cs_generate_variant:Nn \__rawobjects_mgen:nN { fN }
512 \cs_generate_variant:Nn \__rawobjects_mgen:nnn { fnn }
513 \cs_generate_variant:Nn \__rawobjects_mgen_pr:nN { fN }
514 \cs_generate_variant:Nn \__rawobjects_mgen_pr:nnn { fnn }
515
516 \cs_new_protected:Nn \object_member_generate:NN
517 {
518   \__rawobjects_mgen:fN { \cs_to_str:N #1 } #2
519 }
520
521 \cs_new_protected:Nn \object_member_generate_inline:Nnn
522 {
523   \__rawobjects_mgen:fnn { \cs_to_str:N #1 }{ #2 }{ #3 }
524 }
525 \cs_new_protected:Nn \object_member_generate_protected:NN
526 {
527   \__rawobjects_mgen_pr:fN { \cs_to_str:N #1 } #2
528 }
529

```

```

530 \cs_new_protected:Nn \object_member_generate_protected_inline:Nnn
531 {
532   \__rawobjects_mgen_pr:fnn { \cs_to_str:N #1 }{ #2 }{ #3 }
533 }
534

```

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

\object_ncmember_generate:NN Generate ncmember versions of specified functions.

```

\object_ncmember_generate_inline:Nnn
\object_ncmember_generate_protected:NN
\object_ncmember_generate_protected_inline:Nnn
535
536 \cs_new_protected:Nn \__rawobjects_ncgen:nN
537 {
538   \__rawobjects_save_fun:N #2
539   \cs_new:cpn { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
540   {
541     #2
542     {
543       \object_ncmember_adr:nnn{ ##1 }{ ##2 }{ ##3 }
544     }
545   }
546 }
547 \cs_new_protected:Nn \__rawobjects_ncgen_pr:nN
548 {
549   \__rawobjects_save_fun:N #2
550   \cs_new_protected:cpn
551   { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
552   {
553     #2
554     {
555       \object_ncmember_adr:nnn{ ##1 }{ ##2 }{ ##3 }
556     }
557   }
558 }
559
560 \cs_new_protected:Nn \__rawobjects_ncgen:nnn
561 {
562   \__rawobjects_save_dat:n { #3 }
563
564   \cs_new:cpn { __rawobjects_auxfun_#1 :nn } ##1##2
565   {
566     \use:c{ #2 : #3 }
567   }
568   \cs_generate_variant:cn { __rawobjects_auxfun_#1 :nn }{ nf }
569
570   \cs_new:cpn { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
571   {
572     \use:c { __rawobjects_auxfun_#1 :nf }
573     { ##3 }
574     {
575       \__rawobjects_scope_pfx_cl:n{ ##1 }
576     }
577     {
578       \object_ncmember_adr:nnn{ ##1 }{ ##2 }{ ##3 }

```

```

579     }
580   }
581 }
582 \cs_new_protected:Nn \__rawobjects_ncgen_pr:nnn
583 {
584   \__rawobjects_save_dat:n { #3 }
585
586   \cs_new:cpn { __rawobjects_auxfun_#1 :nn } ##1##2
587   {
588     \use:c{ #2 : #3 }
589   }
590   \cs_generate_variant:cn { __rawobjects_auxfun_#1 :nn }{ nf }
591
592   \cs_new_protected:cpn
593   { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
594   {
595     \use:c { __rawobjects_auxfun_#1 :nf }
596     { ##3 }
597     {
598       \__rawobjects_scope_pfx_cl:n{ ##1 }
599     }
600     {
601       \object_ncmember_adr:nnn{ ##1 }{ ##2 }{ ##3 }
602     }
603   }
604 }
605
606 \cs_generate_variant:Nn \__rawobjects_ncgen:nN { fN }
607 \cs_generate_variant:Nn \__rawobjects_ncgen:nnn { fnn }
608 \cs_generate_variant:Nn \__rawobjects_ncgen_pr:nN { fN }
609 \cs_generate_variant:Nn \__rawobjects_ncgen_pr:nnn { fnn }
610
611 \cs_new_protected:Nn \object_ncmember_generate:NN
612 {
613   \__rawobjects_ncgen:fN { \cs_to_str:N #1 } #2
614 }
615
616 \cs_new_protected:Nn \object_ncmember_generate_inline:Nnn
617 {
618   \__rawobjects_ncgen:fnn { \cs_to_str:N #1 }{ #2 }{ #3 }
619 }
620 \cs_new_protected:Nn \object_ncmember_generate_protected:NN
621 {
622   \__rawobjects_ncgen_pr:fN { \cs_to_str:N #1 } #2
623 }
624
625 \cs_new_protected:Nn \object_ncmember_generate_protected_inline:Nnn
626 {
627   \__rawobjects_ncgen_pr:fnn { \cs_to_str:N #1 }{ #2 }{ #3 }
628 }
629

```

(End definition for \object_ncmember_generate:NN and others. These functions are documented on page 10.)

Auxiliary functions

```

630
631 \cs_generate_variant:Nn \cs_generate_variant:Nn { cx }
632
633 \cs_new_protected:Nn \__rawobjects_genmem_int:nnn
634 {
635   \__rawobjects_mgen:nnn { #1 }{ #2 }{ #3 }
636   \cs_generate_variant:cx
637     { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str }
638     { Vnn \str_use:N \l__rawobjects_tmp_fa_str, nnv \str_use:N \l__rawobjects_tmp_fa_str }
639   \cs_generate_variant:cx
640     { #1 : nn \str_use:N \l__rawobjects_tmp_fa_str }
641     { Vn \str_use:N \l__rawobjects_tmp_fa_str }
642 }
643 \cs_new_protected:Nn \__rawobjects_genmem_pr_int:nnn
644 {
645   \__rawobjects_mgen_pr:nnn { #1 }{ #2 }{ #3 }
646   \cs_generate_variant:cx
647     { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str }
648     { Vnn \str_use:N \l__rawobjects_tmp_fa_str, nnv \str_use:N \l__rawobjects_tmp_fa_str }
649   \cs_generate_variant:cx
650     { #1 : nn \str_use:N \l__rawobjects_tmp_fa_str }
651     { Vn \str_use:N \l__rawobjects_tmp_fa_str }
652 }
653
654 \cs_new_protected:Nn \__rawobjects_genncm_int:nnn
655 {
656   \__rawobjects_ncgen:nnn { #1 }{ #2 }{ #3 }
657   \cs_generate_variant:cx
658     { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str }
659     { Vnn \str_use:N \l__rawobjects_tmp_fa_str }
660 }
661 \cs_new_protected:Nn \__rawobjects_genncm_pr_int:nnn
662 {
663   \__rawobjects_ncgen_pr:nnn { #1 }{ #2 }{ #3 }
664   \cs_generate_variant:cx
665     { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str }
666     { Vnn \str_use:N \l__rawobjects_tmp_fa_str }
667 }
668
669
670 \msg_new:nnnn { rawobjects }{ noerr }{ Unspecified ~ scope }
671 {
672   Object ~ #1 ~ hasn't ~ a ~ scope ~ variable
673 }
674
675 \msg_new:nnnn { rawobjects }{ scoperr }{ Nonstandard ~ scope }
676 {
677   Operation ~ not ~ permitted ~ on ~ object ~ #1 ~
678   ~ since ~ it ~ wasn't ~ declared ~ local ~ or ~ global
679 }
680
681 \cs_new_protected:Nn \__rawobjects_force_scope:n

```

```

682 {
683   \cs_if_exist:cTF
684   {
685     \object_ncmember_adr:nnn
686     {
687       \object_embedded_adr:nn{ #1 }{ /_I_/ }
688     }
689     { S }{ str }
690   }
691   {
692     \bool_if:nF
693     {
694       \object_if_local_p:n { #1 } || \object_if_global_p:n { #1 }
695     }
696     {
697       \msg_error:nnx { rawobjects }{ scoperr }{ #1 }
698     }
699   }
700   {
701     \msg_error:nnx { rawobjects }{ noerr }{ #1 }
702   }
703 }
704

```

\object_new_member:nnn Creates a new member variable

\object_new_member_tracked:nnn

```

705
706 \cs_new_protected:Nn \object_new_member:nnn
707 {
708   \cs_if_exist_use:cT { #3 _ new:c }
709   {
710     { \object_member_adr:nnn { #1 }{ #2 }{ #3 } }
711   }
712 }
713
714 \cs_generate_variant:Nn \object_new_member:nnn { Vnn, nnv }
715
716
717 \cs_new_protected:Nn \object_new_member_tracked:nnn
718 {
719   \object_new_member:nnn { #1 }{ #2 }{ #3 }
720
721   \str_const:cn
722   {
723     \object_ncmember_adr:nnn
724     {
725       \object_embedded_adr:nn { #1 }{ /_T_/ }
726     }
727     { #2 _ type }{ str }
728   }
729   { #3 }
730 }
731
732 \cs_generate_variant:Nn \object_new_member_tracked:nnn { Vnn, nnv }
733

```


(End definition for `\object_new_member:nnn` and `\object_new_member_tracked:nnn`. These functions are documented on page 8.)

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

```
734
735 \__rawobjects_genmem_int:nnn {object_member_use}{ #1_use }{c}
736
737 \cs_generate_variant:Nn \object_member_use:nnn {vnn}
738
```

(End definition for `\object_member_use:nnn` and `\object_member_use:nn`. These functions are documented on page 9.)

`\object_member_set:nnnn` Set the value a member.
`\object_member_set:nnn`

```
739
740 \__rawobjects_genmem_pr_int:nnn {object_member_set}{ #1_#2 set }{ cn }
741
```

(End definition for `\object_member_set:nnnn` and `\object_member_set:nnn`. These functions are documented on page 9.)

`\object_member_set_eq:nnnN` Make a member equal to another variable.
`\object_member_set_eq:nnN`

```
742
743 \__rawobjects_genmem_pr_int:nnn { object_member_set_eq }{ #1 _ #2 set_eq }{ cN }
744
745 \cs_generate_variant:Nn \object_member_set_eq:nnnN { nnnnc, Vnnnc }
746
747 \cs_generate_variant:Nn \object_member_set_eq:nnN { nnc, Vnc }
748
```

(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

```
749
750 \cs_new:Nn \object_ncmember_adr:nnn
751 {
752   \tl_to_str:n{ c _ } #1 \tl_to_str:n { _ CONST _ #2 _ #3 }
753 }
754
755 \cs_generate_variant:Nn \object_ncmember_adr:nnn { Vnn, vnn }
756
```

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

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

```
757
758 \cs_new:Nn \object_rcmember_adr:nnn
759 {
760   \object_ncmember_adr:vnn
761   {
762     \object_ncmember_adr:nnn
763     {
764       \object_embedded_adr:nn{ #1 }{ /_I/ }
765     }
766   }
767 }
```

```

766         { P }{ str }
767     }
768     { #2 }{ #3 }
769 }
770
771 \cs_generate_variant:Nn \object_rcmember_adr:nnn { Vnn }

```

(End definition for \object_rcmember_adr:nnn. This function is documented on page 10.)

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

```

\object_ncmember_if_exist:nnnTF
\object_rcmember_if_exist_p:nnn
\object_rcmember_if_exist:nnnTF
772
773 \prg_new_conditional:Nnn \object_ncmember_if_exist:nnn {p, T, F, TF }
774 {
775     \cs_if_exist:cTF
776     {
777         \object_ncmember_adr:nnn { #1 }{ #2 }{ #3 }
778     }
779     {
780         \prg_return_true:
781     }
782     {
783         \prg_return_false:
784     }
785 }
786
787 \prg_new_conditional:Nnn \object_rcmember_if_exist:nnn {p, T, F, TF }
788 {
789     \cs_if_exist:cTF
790     {
791         \object_rcmember_adr:nnn { #1 }{ #2 }{ #3 }
792     }
793     {
794         \prg_return_true:
795     }
796     {
797         \prg_return_false:
798     }
799 }
800
801 \prg_generate_conditional_variant:Nnn \object_ncmember_if_exist:nnn
802 { Vnn }{ p, T, F, TF }
803 \prg_generate_conditional_variant:Nnn \object_rcmember_if_exist:nnn
804 { Vnn }{ p, T, F, TF }
805

```

(End definition for \object_ncmember_if_exist:nnnTF and \object_rcmember_if_exist:nnnTF. These functions are documented on page 10.)

`\object_ncmember_use:nnn` Uses a near/remote constant.

```

\object_rcmember_use:nnn
806
807 \__rawobjects_genncm_int:nnn { object_ncmember_use }{ #1_use}{ c }
808
809 \cs_new:Nn \object_rcmember_use:nnn
810 {
811     \cs_if_exist_use:cT { #3 _ use:c }

```

```

812     {
813       { \object_rcmember_adr:nnn { #1 } { #2 } { #3 } }
814     }
815   }
816
817 \cs_generate_variant:Nn \object_rcmember_use:nnn { Vnn }
818

```

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

`\object_newconst:nnnn` Creates a constant variable, use with caution

```

819
820 \__rawobjects_genncm_pr_int:nnn { object_newconst } { #1 _ const } { cn }
821

```

(End definition for `\object_newconst:nnnn`. This function is documented on page 12.)

`\object_newconst_tl:nnn` Create constants

`\object_newconst_str:nnn`

`\object_newconst_int:nnn`

`\object_newconst_clist:nnn`

`\object_newconst_dim:nnn`

`\object_newconst_skip:nnn`

`\object_newconst_fp:nnn`

```

822
823 \cs_new_protected:Nn \object_newconst_tl:nnn
824 {
825   \object_newconst:nnnn { #1 } { #2 } { tl } { #3 }
826 }
827 \cs_new_protected:Nn \object_newconst_str:nnn
828 {
829   \object_newconst:nnnn { #1 } { #2 } { str } { #3 }
830 }
831 \cs_new_protected:Nn \object_newconst_int:nnn
832 {
833   \object_newconst:nnnn { #1 } { #2 } { int } { #3 }
834 }
835 \cs_new_protected:Nn \object_newconst_clist:nnn
836 {
837   \object_newconst:nnnn { #1 } { #2 } { clist } { #3 }
838 }
839 \cs_new_protected:Nn \object_newconst_dim:nnn
840 {
841   \object_newconst:nnnn { #1 } { #2 } { dim } { #3 }
842 }
843 \cs_new_protected:Nn \object_newconst_skip:nnn
844 {
845   \object_newconst:nnnn { #1 } { #2 } { skip } { #3 }
846 }
847 \cs_new_protected:Nn \object_newconst_fp:nnn
848 {
849   \object_newconst:nnnn { #1 } { #2 } { fp } { #3 }
850 }
851
852 \cs_generate_variant:Nn \object_newconst_tl:nnn { Vnn }
853 \cs_generate_variant:Nn \object_newconst_str:nnn { Vnn }
854 \cs_generate_variant:Nn \object_newconst_int:nnn { Vnn }
855 \cs_generate_variant:Nn \object_newconst_clist:nnn { Vnn }
856 \cs_generate_variant:Nn \object_newconst_dim:nnn { Vnn }
857 \cs_generate_variant:Nn \object_newconst_skip:nnn { Vnn }

```

```

858 \cs_generate_variant:Nn \object_newconst_fp:nnn { Vnn }
859
860
861 \cs_generate_variant:Nn \object_newconst_str:nnn { nnx }
862 \cs_generate_variant:Nn \object_newconst_str:nnn { nnV }
863

```

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

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

```

864
865 \cs_new_protected:Nn \object_newconst_seq_from_clist:nnn
866 {
867   \seq_const_from_clist:cn
868   {
869     \object_ncmember_adr:nnn { #1 }{ #2 }{ seq }
870   }
871   { #3 }
872 }
873
874 \cs_generate_variant:Nn \object_newconst_seq_from_clist:nnn { Vnn }
875

```

(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.

```

876
877 \cs_new_protected:Nn \object_newconst_prop_from_keyval:nnn
878 {
879   \prop_const_from_keyval:cn
880   {
881     \object_ncmember_adr:nnn { #1 }{ #2 }{ prop }
882   }
883   { #3 }
884 }
885
886 \cs_generate_variant:Nn \object_newconst_prop_from_keyval:nnn { Vnn }
887

```

(End definition for `\object_newconst_prop_from_keyval:nnn`. This function is documented on page 12.)

`\object_ncmethod_adr:nnn` Fully expands to the method address.

`\object_rcmethod_adr:nnn`

```

888
889 \cs_new:Nn \object_ncmethod_adr:nnn
890 {
891   #1 \tl_to_str:n { _ CMETHOD _ #2 : #3 }
892 }
893
894 \cs_generate_variant:Nn \object_ncmethod_adr:nnn { Vnn , vnn }
895
896 \cs_new:Nn \object_rcmethod_adr:nnn
897 {
898   \object_ncmethod_adr:vnn
899   {
900     \object_ncmember_adr:nnn

```

```

901         {
902             \object_embedded_adr:nn{ #1 }{ /_I_/ }
903         }
904         { P }{ str }
905     }
906     { #2 }{ #3 }
907 }
908
909 \cs_generate_variant:Nn \object_ncmethod_adr:nnn { Vnn , vnn }
910 \cs_generate_variant:Nn \object_rcmethod_adr:nnn { Vnn }
911

```

(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:nnnTF`
`\object_rcmethod_if_exist:p:nnn`
`\object_rcmethod_if_exist:nnnTF`

Tests if the specified member constant exists.

```

912
913 \prg_new_conditional:Nnn \object_ncmethod_if_exist:nnn {p, T, F, TF }
914 {
915     \cs_if_exist:cTF
916     {
917         \object_ncmethod_adr:nnn { #1 }{ #2 }{ #3 }
918     }
919     {
920         \prg_return_true:
921     }
922     {
923         \prg_return_false:
924     }
925 }
926
927 \prg_new_conditional:Nnn \object_rcmethod_if_exist:nnn {p, T, F, TF }
928 {
929     \cs_if_exist:cTF
930     {
931         \object_rcmethodr_adr:nnn { #1 }{ #2 }{ #3 }
932     }
933     {
934         \prg_return_true:
935     }
936     {
937         \prg_return_false:
938     }
939 }
940
941 \prg_generate_conditional_variant:Nnn \object_ncmethod_if_exist:nnn
942 { Vnn }{ p, T, F, TF }
943 \prg_generate_conditional_variant:Nnn \object_rcmethod_if_exist:nnn
944 { Vnn }{ p, T, F, TF }
945

```

(End definition for `\object_ncmethod_if_exist:nnnTF` and `\object_rcmethod_if_exist:nnnTF`. These functions are documented on page 11.)

`\object_new_cmethod:nnnn` Creates a new method

```

946
947 \cs_new_protected:Nn \object_new_cmethod:nnnn
948 {
949   \cs_new:cn
950   {
951     \object_ncmethod_adr:nnn { #1 }{ #2 }{ #3 }
952   }
953   { #4 }
954 }
955
956 \cs_generate_variant:Nn \object_new_cmethod:nnnn { Vnnn }
957

```

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

`\object_ncmethod_call:nnn` Calls the specified method.

`\object_rcmethod_call:nnn`

```

958
959 \cs_new:Nn \object_ncmethod_call:nnn
960 {
961   \use:c
962   {
963     \object_ncmethod_adr:nnn { #1 }{ #2 }{ #3 }
964   }
965 }
966
967 \cs_new:Nn \object_rcmethod_call:nnn
968 {
969   \use:c
970   {
971     \object_rcmethod_adr:nnn { #1 }{ #2 }{ #3 }
972   }
973 }
974
975 \cs_generate_variant:Nn \object_ncmethod_call:nnn { Vnn }
976 \cs_generate_variant:Nn \object_rcmethod_call:nnn { Vnn }
977

```

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

```

978
979 \cs_new_protected:Nn \__rawobjects_initproxy:nnn
980 {
981   \object_newconst:nnnn
982   {
983     \object_embedded_adr:nn{ #3 }{ /_I_/ }
984   }
985   { ifprox }{ bool }{ \c_true_bool }
986 }
987 \cs_generate_variant:Nn \__rawobjects_initproxy:nnn { VnV }
988

```

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

`\object_if_proxy:nTF`

```

989
990 \cs_new:Nn \__rawobjects_bol_com:N

```

```

991 {
992     \cs_if_exist_p:N #1 && \bool_if_p:N #1
993 }
994
995 \cs_generate_variant:Nn \__rawobjects_bol_com:N { c }
996
997 \prg_new_conditional:Nnn \object_if_proxy:n {p, T, F, TF}
998 {
999     \cs_if_exist:cTF
1000     {
1001         \object_ncmember_adr:nnn
1002         {
1003             \object_embedded_adr:nn{ #1 }{ /_I_/ }
1004         }
1005         { ifprox }{ bool }
1006     }
1007     {
1008         \bool_if:cTF
1009         {
1010             \object_ncmember_adr:nnn
1011             {
1012                 \object_embedded_adr:nn{ #1 }{ /_I_/ }
1013             }
1014             { ifprox }{ bool }
1015         }
1016         {
1017             \prg_return_true:
1018         }
1019         {
1020             \prg_return_false:
1021         }
1022     }
1023     {
1024         \prg_return_false:
1025     }
1026 }
1027

```

(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.

```

1028
1029 \prg_generate_conditional_variant:Nnn \str_if_eq:nn { ve }{ TF }
1030
1031 \prg_new_conditional:Nnn \object_test_proxy:nn {p, T, F, TF}
1032 {
1033     \str_if_eq:veTF
1034     {
1035         \object_ncmember_adr:nnn
1036         {
1037             \object_embedded_adr:nn{ #1 }{ /_I_/ }
1038         }
1039         { P }{ str }
1040     }

```

```

1041 { #2 }
1042 {
1043   \prg_return_true:
1044 }
1045 {
1046   \prg_return_false:
1047 }
1048 }
1049
1050 \prg_new_conditional:Nnn \object_test_proxy:nN {p, T, F, TF}
1051 {
1052   \str_if_eq:cNTF
1053   {
1054     \object_ncmember_adr:nnn
1055     {
1056       \object_embedded_adr:nn{ #1 }{ /_I_/ }
1057     }
1058     { P }{ str }
1059   }
1060   #2
1061   {
1062     \prg_return_true:
1063   }
1064   {
1065     \prg_return_false:
1066   }
1067 }
1068
1069 \prg_generate_conditional_variant:Nnn \object_test_proxy:nn
1070 { Vn }{p, T, F, TF}
1071 \prg_generate_conditional_variant:Nnn \object_test_proxy:nN
1072 { VN }{p, T, F, TF}
1073

```

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

```

\object_create:nnnNN Creates an object from a proxy.
\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
1074
1075 \msg_new:nnnn { rawobjects }{ notproxy }{ Fake ~ proxy }
1076 {
1077   Object ~ #1 ~ is ~ not ~ a ~ proxy.
1078 }
1079
1080 \cs_new_protected:Nn \__rawobjects_force_proxy:n
1081 {
1082   \object_if_proxy:nF { #1 }
1083   {
1084     \msg_error:nnn { rawobjects }{ notproxy }{ #1 }
1085   }
1086 }
1087
1088 \cs_new_protected:Nn \__rawobjects_create_anon:nnnNN
1089 {

```



```

1090 \tl_if_empty:nF{ #1 }
1091 {
1092
1093 \__rawobjects_force_proxy:n { #1 }
1094
1095
1096 \object_newconst_str:nnn
1097 {
1098   \object_embedded_adr:nn{ #3 }{ /_I_/ }
1099 }
1100 { M }{ #2 }
1101 \object_newconst_str:nnn
1102 {
1103   \object_embedded_adr:nn{ #3 }{ /_I_/ }
1104 }
1105 { P }{ #1 }
1106 \object_newconst_str:nnV
1107 {
1108   \object_embedded_adr:nn{ #3 }{ /_I_/ }
1109 }
1110 { S } #4
1111 \object_newconst_str:nnV
1112 {
1113   \object_embedded_adr:nn{ #3 }{ /_I_/ }
1114 }
1115 { V } #5
1116
1117 \seq_map_inline:cn
1118 {
1119   \object_member_adr:nnn { #1 }{ varlist }{ seq }
1120 }
1121 {
1122   \object_new_member:nnv { #3 }{ ##1 }
1123   {
1124     \object_ncmember_adr:nnn { #1 }{ ##1 _ type }{ str }
1125   }
1126 }
1127
1128 \seq_map_inline:cn
1129 {
1130   \object_member_adr:nnn { #1 }{ objlist }{ seq }
1131 }
1132 {
1133   \embedded_create:nvn
1134   { #3 }
1135   {
1136     \object_ncmember_adr:nnn { #1 }{ ##1 _ proxy }{ str }
1137   }
1138   { ##1 }
1139 }
1140
1141 \tl_map_inline:cn
1142 {
1143   \object_member_adr:nnn { #1 }{ init }{ tl }

```

```

1144     }
1145     {
1146         ##1 { #1 }{ #2 }{ #3 }
1147     }
1148
1149     }
1150 }
1151
1152 \cs_generate_variant:Nn \__rawobjects_create_anon:nnnNN { xnxNN, xvxcc }
1153
1154 \cs_new_protected:Nn \object_create:nnnNN
1155 {
1156     \__rawobjects_create_anon:xnxNN { #1 }{ #2 }
1157     { \object_address:nn { #2 }{ #3 } }
1158     #4 #5
1159 }
1160
1161 \cs_generate_variant:Nn \object_create:nnnNN { VnnNN }
1162
1163 \cs_new_protected:Nn \object_create_set:NnnnNN
1164 {
1165     \object_create:nnnNN { #2 }{ #3 }{ #4 } #5 #6
1166     \str_set:Nx #1 { \object_address:nn { #3 }{ #4 } }
1167 }
1168
1169 \cs_new_protected:Nn \object_create_gset:NnnnNN
1170 {
1171     \object_create:nnnNN { #2 }{ #3 }{ #4 } #5 #6
1172     \str_gset:Nx #1 { \object_address:nn { #3 }{ #4 } }
1173 }
1174
1175 \cs_generate_variant:Nn \object_create_set:NnnnNN { NVnnNN, NnnfNN }
1176 \cs_generate_variant:Nn \object_create_gset:NnnnNN { NVnnNN, NnnfNN }
1177
1178
1179
1180 \cs_new_protected:Nn \object_create:nnnN
1181 {
1182     \object_create:nnnNN { #1 }{ #2 }{ #3 } #4 \c_object_public_str
1183 }
1184
1185 \cs_generate_variant:Nn \object_create:nnnN { VnnN }
1186
1187 \cs_new_protected:Nn \object_create_set:NnnnN
1188 {
1189     \object_create_set:NnnnNN #1 { #2 }{ #3 }{ #4 } #5 \c_object_public_str
1190 }
1191
1192 \cs_new_protected:Nn \object_create_gset:NnnnN
1193 {
1194     \object_create_gset:NnnnNN #1 { #2 }{ #3 }{ #4 } #5 \c_object_public_str
1195 }
1196
1197 \cs_generate_variant:Nn \object_create_set:NnnnN { NVnnN }

```

```

1198 \cs_generate_variant:Nn \object_create_gset:NnnnN { NVnnN }
1199
1200 \cs_new_protected:Nn \object_create:nnn
1201 {
1202   \object_create:nnnNN { #1 }{ #2 }{ #3 }
1203   \c_object_global_str \c_object_public_str
1204 }
1205
1206 \cs_generate_variant:Nn \object_create:nnn { Vnn }
1207
1208 \cs_new_protected:Nn \object_create_set:Nnnn
1209 {
1210   \object_create_set:NnnnNN #1 { #2 }{ #3 }{ #4 }
1211   \c_object_global_str \c_object_public_str
1212 }
1213
1214 \cs_new_protected:Nn \object_create_gset:Nnnn
1215 {
1216   \object_create_gset:NnnnNN #1 { #2 }{ #3 }{ #4 }
1217   \c_object_global_str \c_object_public_str
1218 }
1219
1220 \cs_generate_variant:Nn \object_create_set:Nnnn { NVnn }
1221 \cs_generate_variant:Nn \object_create_gset:Nnnn { NVnn }
1222
1223
1224
1225
1226 \cs_new_protected:Nn \embedded_create:nnn
1227 {
1228   \__rawobjects_create_anon:xvxc { #2 }
1229   {
1230     \object_ncmember_adr:nnn
1231     {
1232       \object_embedded_adr:nn{ #1 }{ /_I_/ }
1233     }
1234     { M }{ str }
1235   }
1236   {
1237     \object_embedded_adr:nn
1238     { #1 }{ #3 }
1239   }
1240   {
1241     \object_ncmember_adr:nnn
1242     {
1243       \object_embedded_adr:nn{ #1 }{ /_I_/ }
1244     }
1245     { S }{ str }
1246   }
1247   {
1248     \object_ncmember_adr:nnn
1249     {
1250       \object_embedded_adr:nn{ #1 }{ /_I_/ }
1251     }
1252   }

```

```

1252         { V }{ str }
1253     }
1254 }
1255
1256 \cs_generate_variant:Nn \embedded_create:nnn { nvn, Vnn }
1257

```

(End definition for \object_create:nnnNN and others. These functions are documented on page 13.)

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

Creates a new proxy object

```

1258
1259 \cs_new_protected:Nn \proxy_create:nn
1260 {
1261   \object_create:VnnNN \c_proxy_address_str { #1 }{ #2 }
1262   \c_object_global_str \c_object_public_str
1263 }
1264
1265 \cs_new_protected:Nn \proxy_create_set:Nnn
1266 {
1267   \object_create_set:NVnnNN #1 \c_proxy_address_str { #2 }{ #3 }
1268   \c_object_global_str \c_object_public_str
1269 }
1270
1271 \cs_new_protected:Nn \proxy_create_gset:Nnn
1272 {
1273   \object_create_gset:NVnnNN #1 \c_proxy_address_str { #2 }{ #3 }
1274   \c_object_global_str \c_object_public_str
1275 }
1276
1277
1278
1279 \cs_new_protected:Nn \proxy_create:nnN
1280 {
1281   \__rawobjects_launch_deprecate:NN \proxy_create:nnN \proxy_create:nn
1282   \object_create:VnnNN \c_proxy_address_str { #1 }{ #2 }
1283   \c_object_global_str #3
1284 }
1285
1286 \cs_new_protected:Nn \proxy_create_set:NnnN
1287 {
1288   \__rawobjects_launch_deprecate:NN \proxy_create_set:NnnN \proxy_create_set:Nnn
1289   \object_create_set:NVnnNN #1 \c_proxy_address_str { #2 }{ #3 }
1290   \c_object_global_str #4
1291 }
1292
1293 \cs_new_protected:Nn \proxy_create_gset:NnnN
1294 {
1295   \__rawobjects_launch_deprecate:NN \proxy_create_gset:NnnN \proxy_create_gset:Nnn
1296   \object_create_gset:NVnnNN #1 \c_proxy_address_str { #2 }{ #3 }
1297   \c_object_global_str #4
1298 }
1299

```

(End definition for \proxy_create:nn, \proxy_create_set:Nnn, and \proxy_create_gset:Nnn. These functions are documented on page 14.)

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

```
1300
1301 \cs_new_protected:Nn \proxy_push_member:nnn
1302 {
1303   \object_newconst_str:nnn { #1 }{ #2 _ type }{ #3 }
1304   \seq_gput_left:cn
1305   {
1306     \object_member_adr:nnn { #1 }{ varlist }{ seq }
1307   }
1308   { #2 }
1309 }
1310
1311 \cs_generate_variant:Nn \proxy_push_member:nnn { Vnn }
1312
```

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

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

```
1313
1314 \cs_new_protected:Nn \proxy_push_embedded:nnn
1315 {
1316   \object_newconst_str:nnx { #1 }{ #2 _ proxy }{ #3 }
1317   \seq_gput_left:cn
1318   {
1319     \object_member_adr:nnn { #1 }{ objlist }{ seq }
1320   }
1321   { #2 }
1322 }
1323
1324 \cs_generate_variant:Nn \proxy_push_embedded:nnn { Vnn }
1325
```

(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.

```
1326
1327 \cs_new_protected:Nn \proxy_add_initializer:nN
1328 {
1329   \tl_gput_right:cn
1330   {
1331     \object_member_adr:nnn { #1 }{ init }{ tl }
1332   }
1333   { #2 }
1334 }
1335
1336 \cs_generate_variant:Nn \proxy_add_initializer:nN { VN }
1337
```

(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.

```
1338
1339 \str_const:Nx \c_proxy_address_str
1340 { \object_address:nn { rawobjects }{ proxy } }
```

```

1341
1342 \object_newconst_str:nnn
1343 {
1344     \object_embedded_adr:Vn \c_proxy_address_str { /_I_/ }
1345 }
1346 { M }{ rawobjects }
1347
1348 \object_newconst_str:nnV
1349 {
1350     \object_embedded_adr:Vn \c_proxy_address_str { /_I_/ }
1351 }
1352 { P } \c_proxy_address_str
1353
1354 \object_newconst_str:nnV
1355 {
1356     \object_embedded_adr:Vn \c_proxy_address_str { /_I_/ }
1357 }
1358 { S } \c_object_global_str
1359
1360 \object_newconst_str:nnV
1361 {
1362     \object_embedded_adr:Vn \c_proxy_address_str { /_I_/ }
1363 }
1364 { V } \c_object_public_str
1365
1366
1367 \_rawobjects_initproxy:VnV \c_proxy_address_str { rawobjects } \c_proxy_address_str
1368
1369 \object_new_member:Vnn \c_proxy_address_str { init }{ t1 }
1370
1371 \object_new_member:Vnn \c_proxy_address_str { varlist }{ seq }
1372
1373 \object_new_member:Vnn \c_proxy_address_str { objlist }{ seq }
1374
1375 \proxy_push_member:Vnn \c_proxy_address_str
1376 { init }{ t1 }
1377 \proxy_push_member:Vnn \c_proxy_address_str
1378 { varlist }{ seq }
1379 \proxy_push_member:Vnn \c_proxy_address_str
1380 { objlist }{ seq }
1381
1382 \proxy_add_initializer:VN \c_proxy_address_str
1383 \_rawobjects_initproxy:nnn
1384

```

(End definition for \c_proxy_address_str. This variable is documented on page 13.)

\object_allocate_incr:NNnnNN Create an address and use it to instantiate an object

```

\object_gallocate_incr:NNnnNN
\object_allocate_gincr:NNnnNN
\object_gallocate_gincr:NNnnNN
1385
1386 \cs_new:Nn \_rawobjects_combine_aux:nnn
1387 {
1388     anon . #3 . #2 . #1
1389 }
1390

```

```

1391 \cs_generate_variant:Nn \__rawobjects_combine_aux:nnn { Vnf }
1392
1393 \cs_new:Nn \__rawobjects_combine:Nn
1394 {
1395   \__rawobjects_combine_aux:Vnf #1 { #2 }
1396   {
1397     \cs_to_str:N #1
1398   }
1399 }
1400
1401 \cs_new_protected:Nn \object_allocate_incr:NNnnNN
1402 {
1403   \object_create_set:NnnfNN #1 { #3 }{ #4 }
1404   {
1405     \__rawobjects_combine:Nn #2 { #3 }
1406   }
1407   #5 #6
1408
1409   \int_incr:N #2
1410 }
1411
1412 \cs_new_protected:Nn \object_gallocate_incr:NNnnNN
1413 {
1414   \object_create_gset:NnnfNN #1 { #3 }{ #4 }
1415   {
1416     \__rawobjects_combine:Nn #2 { #3 }
1417   }
1418   #5 #6
1419
1420   \int_incr:N #2
1421 }
1422
1423 \cs_generate_variant:Nn \object_allocate_incr:NNnnNN { NNVnNN }
1424
1425 \cs_generate_variant:Nn \object_gallocate_incr:NNnnNN { NNVnNN }
1426
1427 \cs_new_protected:Nn \object_allocate_gincr:NNnnNN
1428 {
1429   \object_create_set:NnnfNN #1 { #3 }{ #4 }
1430   {
1431     \__rawobjects_combine:Nn #2 { #3 }
1432   }
1433   #5 #6
1434
1435   \int_gincr:N #2
1436 }
1437
1438 \cs_new_protected:Nn \object_gallocate_gincr:NNnnNN
1439 {
1440   \object_create_gset:NnnfNN #1 { #3 }{ #4 }
1441   {
1442     \__rawobjects_combine:Nn #2 { #3 }
1443   }
1444   #5 #6

```

```

1445
1446     \int_gincr:N #2
1447   }
1448
1449   \cs_generate_variant:Nn \object_allocate_gincr:NNnnNN { NNvNNN }
1450
1451   \cs_generate_variant:Nn \object_gallocate_gincr:NNnnNN { NNvNNN }
1452

```

(End definition for `\object_allocate_incr:NNnnNN` and others. These functions are documented on page 14.)

`\object_assign:nn` Copy an object to another one.

```

1453   \cs_new_protected:Nn \object_assign:nn
1454   {
1455     \seq_map_inline:cn
1456     {
1457       \object_member_adr:vnn
1458       {
1459         \object_ncmember_adr:nnn
1460         {
1461           \object_embedded_adr:nn{ #1 }{ /_I_/ }
1462         }
1463         { P }{ str }
1464       }
1465       { varlist }{ seq }
1466     }
1467     {
1468       \object_member_set_eq:nnc { #1 }{ ##1 }
1469       {
1470         \object_member_adr:nn{ #2 }{ ##1 }
1471       }
1472     }
1473   }
1474
1475   \cs_generate_variant:Nn \object_assign:nn { nV, Vn, VV }

```

(End definition for `\object_assign:nn`. This function is documented on page 15.)

```

1476 \endpackage

```