The \texttt{lt3rawobjects} package

Paolo De Donato

Released on 2023/03/17 Version 2.3

Contents

1 Introduction \hspace{1cm} 2
2 Addresses \hspace{1cm} 2
3 Objects \hspace{1cm} 3
4 Fields \hspace{1cm} 3
\hspace{1cm} 4.1 Constants \hspace{1cm} 4
\hspace{1cm} 4.2 Methods \hspace{1cm} 4
\hspace{1cm} 4.3 Members \hspace{1cm} 4
5 Object members \hspace{1cm} 4
\hspace{1cm} 5.1 Create a pointer member \hspace{1cm} 4
\hspace{1cm} 5.2 Clone the inner structure \hspace{1cm} 5
\hspace{1cm} 5.3 Embedded objects \hspace{1cm} 6
6 Library functions \hspace{1cm} 7
\hspace{1cm} 6.1 Common functions \hspace{1cm} 7
\hspace{1cm} 6.2 Base object functions \hspace{1cm} 7
\hspace{1cm} 6.3 Members \hspace{1cm} 8
\hspace{1cm} 6.4 Constants \hspace{1cm} 10
\hspace{1cm} 6.5 Methods \hspace{1cm} 11
\hspace{1cm} 6.6 Creation of constants \hspace{1cm} 12
\hspace{1cm} 6.7 Macros \hspace{1cm} 12
\hspace{1cm} 6.8 Proxies and object creation \hspace{1cm} 13
7 Examples \hspace{1cm} 15
8 Implementation \hspace{1cm} 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 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 Instead changes introduced in minor and patch version updates are always backward compatible, and any withdrawn function is declared deprecated instead of being removed.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 if the name of a control sequence \( \langle \text{name} \rangle \) that can obtained by full expanding \( \text{\texttt{cs\_to\_str:N}} \ \langle \text{name} \rangle \). 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 \( \text{\texttt{tl\_to\_str:n}} \) 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 \( \text{\texttt{exp\_not:n}} \) (because they leave unexpanded control sequences after an \( x \) or \( e \) expansion, and expanded addresses can’t have control sequences inside them). In particular, \( \text{\texttt{tl\_to\_str:n}} \ \langle \#\# \rangle \) 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 \( c \) expanded 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 \( \text{\texttt{rwobj\_address\_f:n}} \) and pass them to your function. For example,

\[
\text{your\_function:f}{ \text{\texttt{rwobj\_address\_f:n}} \ \langle \text{your \ address} \rangle }
\]

Remember that \( \text{\texttt{rwobj\_address\_f:n}} \) 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 LATEX 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 fields. 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 fields 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 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 Fields

Remember that objects are just a collection of different fields uniquely identifidied by a pure address. Here an field could be one of the following entities:

- a LATEX variable, in which case the field is called member;
- a LATEX constant, in which case the field is called just constant;
- a LATEX function, in which case the field is called method;
- generic control sequences, in which case the field is called simply macro;
- an entire object, in which case the field 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 LATEX 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 LATEX3 functions that can’t be changed once they’re created. Like constant, methods could be near or remote. Moreover, functions in this library dealing with near methods contain `ncmethod` whereas those dealing with remote methods contain `rcmethod` in their names.

4.3 Members

Members are just mutable LATEX3 variables. You can manually create new members in already existing objects or you can put the definition of a new member directly in a proxy with the `\proxy_push_member` functions. In this way all the objects created with that proxy will have a member according to such definition. If the object is local/global then all its members are automatically local/global.

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

5 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 LATEX3 variable you can’t just put it inside a member or constant. However, there are some very easy workarounds to insert object instances as fields of other objects.

For example, we’re in module `MOD` and we have an object with id `PAR`. We want to provide `PAR` with a field 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`

```latex
\object_create:nnn
\{ \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.

```
\object_new_member:nnn
 |
| \object_address:nn \{ MOD \} \{ PAR \}
| \{ \pointer \} \{ \str \}
\object_member_set:nnnx
 |
| \object_address:nn \{ MOD \} \{ PAR \}
| \{ \pointer \} \{ \str \}
| \{ \object_address:nn \{ MOD \} \{ INST \} \}
```

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: \(\langle \type \rangle_{\new:c}\), \(\langle \type \rangle_{\\langle \(g\rangle\set:c}\) and \(\langle \type \rangle_{\use:c}\).

**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

```
\object_new_member:nnn
 |
| \object_address:nn \{ MOD \} \{ PAR \}
| \{ \prx-x \} \{ \str \}
\object_new_member:nnn
 |
| \{ }
```

...
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 field to it. Don’t use this method if you later need to retrieve the stored object entirely and not only its fields.

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 { MOD }{ PAR }

  ){ prx-y }{ int }
}
```

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

Advantages

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

Disadvantages

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

6.1 Common functions

\texttt{\textbackslash rwobj\_address\_f:not} \{\texttt{\textbackslash address}\}\{

Fully expand an address in an f-type argument.

From: 2.3

6.2 Base object functions

\texttt{\textbackslash object\_address:n} \{\texttt{\textbackslash module}\}\{\texttt{id}\}\{

Composes the address of object in module \texttt{\langle module\rangle} with identifier \texttt{\langle id\rangle} and places it in the input stream. Notice that both \texttt{\langle module\rangle} and \texttt{\langle id\rangle} are converted to strings before composing them in the address, so they shouldn’t contain any command inside.

From: 1.0

\texttt{\textbackslash object\_address\_set:n} \{\texttt{\textbackslash str \textbackslash var}\}\{\texttt{\textbackslash module}\}\{\texttt{id}\}\{

Stores the address of selected object inside the string variable \texttt{\langle str var\rangle}.

From: 1.1

\texttt{\textbackslash object\_embedded\_adr:n} \{\texttt{\textbackslash address}\}\{\texttt{id}\}\{

Compose the address of embedded object with name \texttt{\langle id\rangle} inside the parent object with address \texttt{\langle address\rangle}. 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

\texttt{\textbackslash object\_if\_exist\_p:n} \{\texttt{\textbackslash address}\}\{

Tests if an object was instantiated at the specified address.

From: 1.0

\texttt{\textbackslash object\_if\_local\_p:n} \{\texttt{\textbackslash address}\}\{

Tests if the object is local or global.

From: 1.0

\texttt{\textbackslash object\_get\_module:n} \{\texttt{\textbackslash address}\}\{

Get the object module and its generator.

From: 1.0
6.3 Members

\texttt{\textbackslash object\_member\_adr:nnn} \star \texttt{\textbackslash object\_member\_adr:nn \{(address)\} \{(member name)\} \{(member type)\}}
\texttt{\textbackslash object\_member\_adr:(Vnn|nnv) \star \texttt{\textbackslash object\_member\_adr:nn \{(address)\} \{(member name)\}}}
\texttt{\textbackslash object\_member\_adr:nn} \star \texttt{\textbackslash object\_member\_adr:Vn} \star \texttt{\textbackslash object\_member\_adr:Vn}

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

From: 1.0

\texttt{\textbackslash object\_member\_if\_exist\_p:nnn} \star \texttt{\textbackslash object\_member\_if\_exist\_p:nnn \{(address)\} \{(member name)\} \{(member type)\}}
\texttt{\textbackslash object\_member\_if\_exist\_p:Vnn \star \texttt{\textbackslash object\_member\_if\_exist\_p:nn \{(address)\} \{(member name)\}}}
\texttt{\textbackslash object\_member\_if\_exist\_p:nn} \star \texttt{\textbackslash object\_member\_if\_exist\_p:Vnn} \star \texttt{\textbackslash object\_member\_if\_exist\_p:nn}

Tests if the specified member exist.

From: 2.0

\texttt{\textbackslash object\_member\_if\_tracked\_p:nn} \star \texttt{\textbackslash object\_member\_if\_tracked\_p:nn \{(address)\} \{(member name)\}}
\texttt{\textbackslash object\_member\_if\_tracked\_p:Vn} \star \texttt{\textbackslash object\_member\_if\_tracked\_p:nn \{(address)\} \{(member name)\} \{(true \ code)\} \{(false \ code)\}}
\texttt{\textbackslash object\_member\_if\_tracked\_p:nn \star \texttt{\textbackslash object\_member\_if\_tracked\_p:Vn \{(true \ code)\} \{(false \ code)\}}}

Tests if the specified member exist and is tracked.

From: 2.3

\texttt{\textbackslash object\_member\_type:nn} \star \texttt{\textbackslash object\_member\_type:nn \{(address)\} \{(member name)\}}
\texttt{\textbackslash object\_member\_type:Vn} \star \texttt{\textbackslash object\_member\_type:nn \{(address)\} \{(member name)\} \{(member type)\}}

Fully expands to the type of specified tracked member.

From: 1.0

\texttt{\textbackslash object\_new\_member:nnn} \texttt{\textbackslash object\_new\_member:nnn \{(address)\} \{(member name)\} \{(member type)\}}
\texttt{\textbackslash object\_new\_member:(Vnn|nnv) \textbackslash object\_new\_member:nn}

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

From: 1.0

\texttt{\textbackslash object\_new\_member\_tracked:nnn} \texttt{\textbackslash object\_new\_member\_tracked:nnn \{(address)\} \{(member name)\} \{(member type)\}}
\texttt{\textbackslash object\_new\_member\_tracked:\textbackslash Vnn \textbackslash object\_new\_member\_tracked:nn}

Creates a new tracked member.

From: 2.3
Uses the specified member variable.
From: 1.0

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

Sets the value of specified member equal to the value of \{variable\}.
From: 1.0

Define the new functions \{name\}:nnn\langle Targs\rangle and \{name\}:nn\langle Targs\rangle that pass to \{name\}:\langle arg1\rangle\langle args\rangle the specified member address as the first argument. \langle Targs\rangle is a list of argument specifications obtained by transforming each element of \langle args\rangle to \n, \N, \w, \T or \F.

The first three parameters of \{name\}:nn\langle args\rangle should be in the following order:

1. an object address;
2. a member name;
3. the type of specified member.

Function \{name\}:nn\langle args\rangle only accepts the first two parameters and works only with tracked members. Notice that \langle arg1\rangle must be only one of the following: \n, \c, \v, \x, \f, \e, \o.
From: 2.3

Works as \object_member_generate:NN, however in \{name\} 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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>From</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\object_ncmember_adr:nnn</code></td>
<td>▶ <code>\object_ncmember_adr:nnn {&lt;address&gt;} {&lt;member name&gt;} {&lt;member type&gt;}</code></td>
<td>2.0</td>
</tr>
<tr>
<td>`\object_ncmember_adr:(Vnn</td>
<td>vnn)`</td>
<td>▶ <code>\object_ncmember_adr:nnn {&lt;address&gt;} {&lt;member name&gt;} {&lt;member type&gt;}</code></td>
</tr>
<tr>
<td><code>\object_rcmember_adr:nnn</code></td>
<td>▶ <code>\object_ncmember_adr:nnn {&lt;address&gt;} {&lt;member name&gt;} {&lt;member type&gt;}</code></td>
<td></td>
</tr>
<tr>
<td><code>\object_rcmember_adr:Vnn</code></td>
<td>▶ <code>\object_rcmember_adr:nnn {&lt;address&gt;} {&lt;member name&gt;} {&lt;member type&gt;}</code></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>From</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\object_ncmember_if_exist_p:nnn</code></td>
<td>▶ <code>\object_ncmember_if_exist_p:nnn {&lt;address&gt;} {&lt;member name&gt;} {&lt;member type&gt;}</code></td>
<td>2.0</td>
</tr>
<tr>
<td><code>\object_ncmember_if_exist_p:Vnn</code></td>
<td>▶ <code>\object_ncmember_if_exist_p:nnn {&lt;address&gt;} {&lt;member name&gt;} {&lt;member type&gt;}</code></td>
<td></td>
</tr>
<tr>
<td><code>\object_ncmember_if_exist:nnn</code></td>
<td>▶ <code>\object_ncmember_if_exist:nnn {&lt;true code&gt;} {&lt;false code&gt;}</code></td>
<td>2.0</td>
</tr>
<tr>
<td><code>\object_rcmember_if_exist_p:nnn</code></td>
<td>▶ <code>\object_rcmember_if_exist_p:nnn {&lt;true code&gt;} {&lt;false code&gt;}</code></td>
<td></td>
</tr>
<tr>
<td><code>\object_rcmember_if_exist:nnn</code></td>
<td>▶ <code>\object_rcmember_if_exist:nnn {&lt;true code&gt;} {&lt;false code&gt;}</code></td>
<td></td>
</tr>
<tr>
<td><code>\object_rcmember_if_exist:Vnn</code></td>
<td>▶ <code>\object_rcmember_if_exist:nnn {&lt;true code&gt;} {&lt;false code&gt;}</code></td>
<td></td>
</tr>
</tbody>
</table>

Tests if the specified member constant exist.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>From</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\object_ncmember_use:nnn</code></td>
<td>▶ <code>\object_ncmember_use:nnn {&lt;address&gt;} {&lt;member name&gt;} {&lt;member type&gt;}</code></td>
<td>2.0</td>
</tr>
<tr>
<td><code>\object_ncmember_use:Vnn</code></td>
<td>▶ <code>\object_ncmember_use:nnn {&lt;address&gt;} {&lt;member name&gt;} {&lt;member type&gt;}</code></td>
<td></td>
</tr>
<tr>
<td><code>\object_rcmember_use:nnn</code></td>
<td>▶ <code>\object_rcmember_use:nnn {&lt;address&gt;} {&lt;member name&gt;} {&lt;member type&gt;}</code></td>
<td></td>
</tr>
<tr>
<td><code>\object_rcmember_use:Vnn</code></td>
<td>▶ <code>\object_rcmember_use:nnn {&lt;address&gt;} {&lt;member name&gt;} {&lt;member type&gt;}</code></td>
<td></td>
</tr>
</tbody>
</table>

Uses the specified near/remote constant member.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>From</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\object_ncmember_generate:NN</code></td>
<td>▶ <code>\object_ncmember_generate:NN \{&lt;name&gt;\} \{name1\} \{arg1\} \{args\}</code></td>
<td>2.3</td>
</tr>
<tr>
<td><code>\object_ncmember_protected_generate:NN</code></td>
<td>▶ <code>\object_ncmember_protected_generate:NN \{&lt;name&gt;\} \{name1\} \{arg1\} \{args\}</code></td>
<td></td>
</tr>
<tr>
<td><code>\object_rcmember_generate:NN</code></td>
<td>▶ <code>\object_rcmember_generate:NN \{&lt;name&gt;\} \{name1\} \{arg1\} \{args\}</code></td>
<td></td>
</tr>
<tr>
<td><code>\object_rcmember_protected_generate:NN</code></td>
<td>▶ <code>\object_rcmember_protected_generate:NN \{&lt;name&gt;\} \{name1\} \{arg1\} \{args\}</code></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>From</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\object_ncmember_generate_inline:Nnn</code></td>
<td>▶ <code>\object_ncmember_generate_inline:Nnn \{&lt;name&gt;\} \{name2\} \{arg1\} \{args\}</code></td>
<td>2.3</td>
</tr>
<tr>
<td><code>\object_ncmember_protected_generate_inline:Nnn</code></td>
<td>▶ <code>\object_ncmember_protected_generate_inline:Nnn \{&lt;name&gt;\} \{name2\} \{arg1\} \{args\}</code></td>
<td></td>
</tr>
<tr>
<td><code>\object_rcmember_generate_inline:Nnn</code></td>
<td>▶ <code>\object_rcmember_generate_inline:Nnn \{&lt;name&gt;\} \{name2\} \{arg1\} \{args\}</code></td>
<td></td>
</tr>
<tr>
<td><code>\object_rcmember_protected_generate_inline:Nnn</code></td>
<td>▶ <code>\object_rcmember_protected_generate_inline:Nnn \{&lt;name&gt;\} \{name2\} \{arg1\} \{args\}</code></td>
<td></td>
</tr>
</tbody>
</table>

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

From: 2.3
6.5 Methods

\object_ncmethod_adr:nnn \object_ncmethod_adr:nnn \object_ncmethod_adr:nnn \object_rcmethod_adr:nnn \object_ncmethod_if_exist_p:nnn \object_ncmethod_if_exist_p:nnn \object_ncmethod_if_exist_p:nnn \object_ncmethod_if_exist:nnnTF \object_ncmethod_if_exist:nnnTF \object_ncmethod_if_exist:nnnTF \object_ncmethod_if_exist:Vnn \object_rcmethod_if_exist_p:nnn \object_rcmethod_if_exist_p:nnn \object_rcmethod_if_exist_p:nnn \object_rcmethod_if_exist_p:Vnn \object_rcmethod_if_exist_p:Vnn \object_rcmethod_if_exist:nnnTF \object_rcmethod_if_exist:nnnTF \object_rcmethod_if_exist:Vnn \object_rcmethod_if_exist:Vnn

\object_new_cmethod:nnnn \object_new_cmethod:nnnn \object_new_cmethod:nnnn \object_new_cmethod:Vnn \object_new_cmethod:Vnn

\object_ncmethod_call:nnn \object_ncmethod_call:nnn \object_ncmethod_call:nnn \object_ncmethod_call:nnn \object_ncmethod_call:Vnn \object_ncmethod_call: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.

\object_ncmethod_if_exist_p:nnn \object_ncmethod_if_exist_p:nnn \object_ncmethod_if_exist_p:nnn \object_ncmethod_if_exist:nnnTF \object_ncmethod_if_exist:nnnTF \object_ncmethod_if_exist:nnnTF \object_ncmethod_if_exist:Vnn \object_rcmethod_if_exist_p:nnn \object_rcmethod_if_exist_p:nnn \object_rcmethod_if_exist_p:nnn \object_rcmethod_if_exist_p:Vnn

Tests if the specified method constant exist.

\object_new_cmethod:nnnn \object_new_cmethod:nnnn \object_new_cmethod:nnnn \object_new_cmethod:Vnn \object_new_cmethod:Vnn

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 \texttt{cs_new:Nn}.

\object_ncmethod_call:nnn \object_ncmethod_call:nnn \object_ncmethod_call:nnn \object_ncmethod_call:nnn \object_ncmethod_call:Vnn \object_ncmethod_call:Vnn

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

From: 2.0
6.6 Creation of constants

\texttt{\object_newconst\_\text{(type)}:nnn \{(address)\} \{(constant name)\} \{(value)\}}

Creates a constant variable with type \texttt{(type)} and sets its value to \texttt{(value)}.

From: 1.1

\texttt{\object_newconst\_\text{seq\-from\-clist}:nnn \{(address)\} \{(constant name)\} \{(comma-list)\}}

Creates a \texttt{seq} constant which is set to contain all the items in \texttt{(comma-list)}.

From: 1.1

\texttt{\object_newconst\_\text{prop\-from\-keyval}:nnn \{(address)\} \{(constant name)\} \{(key)\} \{(value)\}, ...}

Creates a \texttt{prop} constant which is set to contain all the specified key-value pairs.

From: 1.1

\texttt{\object_newconst:nnnn \{(address)\} \{(constant name)\} \{(type)\} \{(value)\}}

Invokes \texttt{(type)\_const:cn} to create the specified constant.

From: 2.1

6.7 Macros

\texttt{\object_macro\_adr:nn \{(macro name)\}}

Address of specified macro.

From: 2.2

\texttt{\object_macro\_use:nn \{(macro name)\}}

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

From: 2.2

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

\object_if_proxy_p:n \{\object_if_proxy_p:n \{address\}\} \{\object_if_proxy:nTF \{\object_if_proxy:nTF \{address\}\} \{\true\} \{\false\}\}\} \{\false\} \{\true\} \{\false\}\*

Test if the specified object is a proxy object.

From: 1.0

\object_if_proxy_p:n \{\object_if_proxy_p:n \{proxy\}\} \{\object_if_proxy:nTF \{\object_if_proxy:nTF \{\object_if_proxy:nTF \{address\}\}\} \{\true\} \{\false\}\}\} \{\false\} \{\true\} \{\false\}\*

Test if the specified object is generated by the selected proxy, where \textit{proxy variable} is a string variable holding the proxy address.

\textbf{\textsc{\texttt{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 \texttt{\object_test_proxy:nN}.

From: 2.0

\texttt{\object_test_proxy_p:nN} \{\object_test_proxy_p:nN \{\object_test_proxy_p:nN \{object address\}\} \{\object_test_proxy:nN \{\object_test_proxy:nN \{proxy address\}\}\} \{\true\} \{\false\}\}\} \{\false\} \{\true\} \{\false\}\*

Test if the specified object is generated by the selected proxy, where \textit{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

\texttt{\c_proxy_address_str} \{\c_proxy_address_str\} \{\texttt{proxy object in the rawobjects module.}\}

From: 1.0

\object_create:nnnNN \{\object_create:nnnNN \{\object_create:nnnNN \{proxy address\}\} \{\object_create:nnnNN \{\object_create:nnnNN \{\object_create:nnnNN \{proxy address\}\}\} \{\object_create:nnnNN \{\object_create:nnnNN \{\object_create:nnnNN \{proxy address\}\}\} \{\true\} \{\false\}\}\}\} \{\false\} \{\true\} \{\false\}\*

Creates an object by using the proxy at \textit{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:nnn \{\object_create:nnn \{\object_create:nnn \{\object_create:nnn \{proxy address\}\}\} \{\object_create:nnn \{\object_create:nnn \{\object_create:nnn \{proxy address\}\}\} \{\true\} \{\false\}\}\}\} \{\false\} \{\true\} \{\false\}\*

Same as \texttt{\object_create:nnnNN} but both create only public objects, and the :nn version only global ones. Always use these two function instead of \texttt{\object_create:nnnNN} unless you strictly need private objects.

From: 2.3

\embedded_create:nnn \{\embedded_create:nnn \{\embedded_create:nnn \{\embedded_create:nnn \{parent object\}\} \{\embedded_create:nnn \{\embedded_create:nnn \{\embedded_create:nnn \{parent object\}\}\} \{\true\}\}\}\}\} \{\false\} \{\true\} \{\false\}\*

Creates an embedded object with name \textit{id} inside \textit{parent object}.

From: 2.2
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>From</th>
</tr>
</thead>
<tbody>
<tr>
<td>\c_object_local_str</td>
<td>Possible values for \texttt{{scope}} parameter.</td>
<td>1.0</td>
</tr>
<tr>
<td>\c_object_global_str</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\c_object_public_str</td>
<td>Possible values for \texttt{{visibility}} parameter.</td>
<td>1.0</td>
</tr>
<tr>
<td>\c_object_private_str</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\begin{itemize}
\item \texttt{\object_create_set:NnnNN} \texttt{\{str var\}} \texttt{\{module\}} \texttt{\{id\}} \texttt{\{scope\}} \texttt{\{visibility\}}
\item \texttt{\object_gallocate_set:NnnNN}
\item \texttt{\object_create_gset:NnnNN}
\item \texttt{\object_gallocate_gset:NnnNN}
\item \texttt{\object_allocate_set:NNnnNN} \texttt{\{str var\}} \texttt{\{int var\}} \texttt{\{module\}} \texttt{\{id\}} \texttt{\{scope\}} \texttt{\{visibility\}}
\item \texttt{\object_gallocate_set:NNnnNN}
\item \texttt{\object_gallocate_set:NNVnNN}
\item \texttt{\object_gallocate_gset:NNnnNN}
\item \texttt{\object_gallocate_gset:NNVnNN}
\end{itemize}

See the table for the parameter meanings. This command creates an object and sets its fully expanded address inside \texttt{\{str var\}}.

\begin{itemize}
\item \texttt{\proxy_create:nnN} \texttt{\{module\}} \texttt{\{id\}} \texttt{\{visibility\}}
\item \texttt{\proxy_create_set:NnnN} \texttt{\{str var\}} \texttt{\{module\}} \texttt{\{id\}} \texttt{\{visibility\}}
\item \texttt{\proxy_create_gset:NnnN}
\end{itemize}

Deprecated in: 2.3

\begin{itemize}
\item \texttt{\proxy_push_member:nnn} \texttt{\{proxy address\}} \texttt{\{member name\}} \texttt{\{member type\}}
\item \texttt{\proxy_push_member:Vnn}
\end{itemize}

From: 1.0

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

\begin{itemize}
\item \texttt{\proxy_push_member:nn}
\item \texttt{\proxy_push_member:nnn}
\item \texttt{\proxy_push_member:Vnn}
\end{itemize}

From: 2.3

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

\begin{itemize}
\item \texttt{\proxy_push_member:nn}
\item \texttt{\proxy_push_member:nnn}
\item \texttt{\proxy_push_member:Vnn}
\end{itemize}

From: 1.0

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

\begin{itemize}
\item \texttt{\object_allocate_set:NNnnNN} \texttt{\{str var\}} \texttt{\{int var\}} \texttt{\{module\}} \texttt{\{id\}} \texttt{\{scope\}} \texttt{\{visibility\}}
\item \texttt{\object_gallocate_set:NNnnNN}
\item \texttt{\object_gallocate_set:NNVnNN}
\item \texttt{\object_gallocate_gset:NNnnNN}
\item \texttt{\object_gallocate_gset:NNVnNN}
\end{itemize}

Creates an object and sets its fully expanded address inside \texttt{\{str var\}}.

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

From: 1.1

Creates a global public proxy object.

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 \texttt{\object_member_type} functions.
\proxy_push_embedded:nnn
\proxy_push_embedded:Vnn
\proxy_add_initializer:nN
\proxy_add_initializer:VN
\object_assign:nn
\object_assign:(Vn|nV|VV)

7 Examples

Example 1

Create a public proxy with id \textit{myproxy} with the specification of a single member variable with name \textit{myvar} and type \textit{tl}, then set its address inside \texttt{\_myproxy\_str}.

Then create a new object with name \textit{myobj} with that proxy, assign then token list \texttt{\_c\_dollar\_str\{} - \texttt{\_dollar\} - \texttt{\_c\_dollar\_str\{}} to \textit{myvar} and then print it.
You can also avoid to specify an object identify and use \object_gallocate_gincr instead:

```latex
\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 }
```

Example 2

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

Internal proxy

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

Container proxy

```latex
\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:

```latex
\str_new:N \g_EXTobj_str
\int_new:N \g_intcount_int
\object_gallocate_gincr:NNnNN \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

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:

```
\tl_use:c 
{ 
  \object_member_adr:nnn 
  { \object_address:nn { MD3 }{ U } } 
  { MEM }{ tl }
}
```

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:

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

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

```
\myaux_print_tl:nnn
{ \object_address:nn { MD3 }{ U } } 
{ 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

```
\myaux_print_tl:nnn
{ \object_address:nn { MD3 }{ U } } 
{ 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
This function does much more: in the second argument you can put also the parameters \#1 and \#2 that will expand respectively to the type of specified member and its scope. Let \myaux_print:nnn be our version of \object_member_use:nnn that retrieves the valued of the specified member, we are now able to define it in this way:

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

8 Implementation

\msg_new:nnn \rawobjects \{ \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.

(End definition for \rwobj_address_f:n. This function is documented on page 7.)
\cs_new:Nn \_rawobjects_scope_pfx:N
\begin{verbatim}
\str_if_eq:NNF #1 \c_object_local_str
\{ g \}
\end{verbatim}
\cs_generate_variant:Nn \_rawobjects_scope_pfx:N { c }
\cs_new:Nn \_rawobjects_scope_pfx_cl:n
\begin{verbatim}
\_rawobjects_scope_pfx:c{
\object_ncmember_adr:nnn
\{ \object_embedded_adr:nn { #1 }{ /_I_/ } \}
\{ S \}{ str }
\}
\}
\end{verbatim}
\cs_new:Nn \_rawobjects_vis_var:N
\begin{verbatim}
\str_use:N #1
\end{verbatim}
\cs_new:Nn \_rawobjects_vis_fun:N
\begin{verbatim}
\str_if_eq:NNT #1 \c_object_private_str
\{ --\}
\}
\end{verbatim}

(End definition for \c_object_local_str and others. These variables are documented on page 14.)

\textbf{object_address:nn} Get address of an object
\begin{verbatim}
\cs_new:Nn \object_address:nn { 
\tl_to_str:n { #1 _ #2 }
\end{verbatim}

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

\textbf{object_embedded_adr:nn} Address of embedded object
\begin{verbatim}
\cs_new:Nn \object_embedded_adr:nn
\begin{verbatim}
#1 \tl_to_str:n{ _SUB_ #2 }
\}
\end{verbatim}
\cs_generate_variant:Nn \object_embedded_adr:nn{ Vn }

(End definition for \textbf{object_embedded_adr:nn}. This function is documented on page 7.)
\texttt{\object_address_set:Nnn} Saves the address of an object into a string variable

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

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

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

\texttt{\object_if_exist_p:n} \texttt{\object_if_exist:nTF} Tests if object exists.

\begin{verbatim}
\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{verbatim}

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

\texttt{\object_get_module:n} \texttt{\object_get_proxy_adr:n} Retrieve the name, module and generating proxy of an object

\begin{verbatim}
\cs_new:Nn \object_get_module:n { 
\object_ncmember_use:n \{ \object_embedded_adr:nn\{ \#1 \}\{ /_I_/ \} \} { M }{ str }
\}
\end{verbatim}

\begin{verbatim}
\cs_new:Nn \object_get_proxy_adr:n { 
\object_ncmember_use:n \{ \object_embedded_adr:nn\{ \#1 \}\{ /_I_/ \} \} { P }{ str }
\end{verbatim}
Test the specified parameters.

\begin{verbatim}
\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_/ } 
\}
\end{verbatim}
\prg_new_conditional:Nnn \object_if_private:n { p, T, F, TF }
{ str_if_eq:cNTF
 \object_ncmember_adr:nn
 { \object_embedded_adr:nn{ #1 }{ _I_/ } }
{ str }
\object_private_str
{ prg_return_true: }
{ prg_return_false: }
}
\prg_generate_conditional_variant:Nnn \object_if_local:n { V }
\object_if_local:nTF
\prg_generate_conditional_variant:Nnn \object_if_global:n { V }
\prg_generate_conditional_variant:Nnn \object_if_public:n { V }
\prg_generate_conditional_variant:Nnn \object_if_private:n { V }
(End definition for \object_if_local:nTF and others. These functions are documented on page 7.)

Generic macro address
\object_macro_adr:nn
\object_macro_use:nn
\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

22
Macro address without object inference

\cs_new:Nn \__rawobjects_member_adr:nnnNN
\__rawobjects_member_adr:nnn

Get the address of a member variable

\cs_new:Nn \object_member_adr:nnn
\__rawobjects_member_adr:nnn

Tests if the specified member exists

\prg_new_conditional:Nnn \object_member_if_exist:nnn
\object_member_if_exist:nnn

\object_member_if_tracked:nn
Tests if the member is tracked.

\object_member_if_tracked:nnTF
(End definition for \object_member_if_exist:nnTF. This function is documented on page 8.)
\object_member_if_tracked:Vn \{ p, T, F, TF \}

(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.

\cs_new:Nn \object_member_type:nn
\{ \cs_if_exist:cTF
\{ \object_rcmember_adr:nnn { #1 }{ #2 _ type }{ str }
\}
\{ \object_rcmember_use:nnn { #1 }{ #2 _ type }{ str }
\}
\{ \cs_if_exist:cT
\{ \object_ncmember_adr:nnn
\{ \object_embedded_adr:nn { #1 }{ /_T_/ } \}
\{ #2 _ type }{ str }
\}
\{ \object_ncmember_use:nnn
\{ \object_embedded_adr:nn { #1 }{ /_T_/ } \}
\{ #2 _ type }{ str }
\}
\}
\}

(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

\cs_new:Nn \object_member_adr:nn
\{ \object_member_adr:nnf { #1 }{ #2 }
\{ \object_member_type:nn { #1 }{ #2 }
\}
\}
\cs_generate_variant:Nn \object_member_adr:nn \{ Vn \}

25
Helper functions for \texttt{\_\_rawobjects\_\_generate} functions.

\begin{verbatim}
\cs_new:Nn \__rawobjects_par_trans:N
\{\str_case:nnF { #1 }
\{ { N }\{ N 
\{ V }\{ N 
\{ n }\{ n 
\{ v }\{ n 
\{ f }\{ n 
\{ x }\{ n 
\{ e }\{ n 
\{ o }\{ n 
\{ - }\{}
\}
\{ #1 
\} \}
\}
\cs_new:Nn \__rawobjects_par_trans:n
\{ \str_map_function:nN { #1 } \__rawobjects_par_trans:N \}
\str_new:N \l__rawobjects_tmp_fa_str
\cs_new_protected:Nn \__rawobjects_save_dat:n
\{ \str_set:Nx \l__rawobjects_tmp_fa_str { \str_tail:n{ #1 } } \}
\cs_new_protected:Nn \__rawobjects_save_dat:nnN
\{ \str_set:Nx \l__rawobjects_tmp_fa_str { \str_tail:n{ #2 } } \}
\cs_new_protected:Nn \__rawobjects_save_dat_aux:n
\{ \__rawobjects_save_dat:nnN #1 \}
\cs_generate_variant:Nn \__rawobjects_save_dat_aux:n { f }
\cs_new_protected:Nn \__rawobjects_save_fun:N
\{ \__rawobjects_save_dat:nnf { \cs_split_function:N #1 } \}
\cs_new_protected:Nn \__rawobjects_save_dat:nn
\{ #1 : #2 \str_use:N \l__rawobjects_tmp_fa_str \}
\end{verbatim}
Generate member versions of specified functions.

\cs_new_protected:Nn \__rawobjects_mgen:nN
\__rawobjects_save_fun:N #2
\cs_new:cpn { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
\object_member_adr:nnn{ ##1 }{ ##2 }{ ##3 }
\cs_new:cpn { #1 : nn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2
\object_member_adr:nn{ ##1 }{ ##2 }
\cs_new_protected:Nn \__rawobjects_mgen_pr:nN
\__rawobjects_save_fun:N #2
\cs_new_protected:cpn
\__rawobjects_auxfun_#1 :nn
\use:c { #2 : #3 }
\cs_generate_variant:cn { __rawobjects_auxfun_#1 :nn }{ nf, ff }
\cs_new:cpn { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
\use:c { __rawobjects_auxfun_#1 :nf }
\cs_new:cpn { #1 : nn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2
{ \use:c { __rawobjects_auxfun_#1 :ff } 
  \object_member_type:nn { ##1 }{ ##2 } 
  \__rawobjects_scope_pfx_cl:n{ ##1 } 
  \object_member_adr:nnn{ ##1 }{ ##2 }{ ##3 } 
}

\cs_new_protected:Nn \__rawobjects_mgen_pr:nnn
{ \__rawobjects_save_dat:n { #3 } 
  \cs_new:cpn { __rawobjects_auxfun_#1 :nn } ##1##2
  \use:c { #2 : #3 } 
  \cs_generate_variant:cn { __rawobjects_auxfun_#1 :nn }{ nf, ff } 

  \cs_new_protected:cpn
  { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
  \use:c { __rawobjects_auxfun_#1 :nf } 
  \object_member_type:nn { ##1 }{ ##2 } 
  \__rawobjects_scope_pfx_cl:n{ ##1 } 
  \object_member_adr:nnn{ ##1 }{ ##2 }{ ##3 } 
}

\cs_new_protected:cpn
{ #1 : nn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
{ \use:c { __rawobjects_auxfun_#1 :ff } 
  \object_member_type:nn { ##1 }{ ##2 } 
  \__rawobjects_scope_pfx_cl:n{ ##1 } 
}

28
\begin{verbatim}
{ \object_member_adr:nn{ #1 }{ #2 }
 }
\end{verbatim}

\begin{verbatim}
\cs_generate_variant:Nn \__rawobjects_mgen:nN { fN }
\cs_generate_variant:Nn \__rawobjects_mgen:nnn { fnn }
\cs_generate_variant:Nn \__rawobjects_mgen_pr:nN { fN }
\cs_generate_variant:Nn \__rawobjects_mgen_pr:nnn { fnn }
\cs_new_protected:Nn \object_member_generate:NN
{ \__rawobjects_mgen:fN { \cs_to_str:N #1 } #2 }
\cs_new_protected:Nn \object_member_generate_inline:Nnn
{ \__rawobjects_mgen:fnn { \cs_to_str:N #1 }{ #2 }{ #3 }
}
\cs_new_protected:Nn \object_member_generate_protected:NN
{ \__rawobjects_mgen_pr:fN { \cs_to_str:N #1 } #2 }
\cs_new_protected:Nn \object_member_generate_protected_inline:Nnn
{ \__rawobjects_mgen_pr:fnn { \cs_to_str:N #1 }{ #2 }{ #3 }
}
\end{verbatim}

\begin{verbatim}
\cs_new_protected:Nn \__rawobjects_ncgen:nN
{ \__rawobjects_save_fun:N #2
 \cs_new:cpn { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
 { \object_ncmember_adr:nnn{ ##1 }{ ##2 }{ ##3 }
 }

\cs_new_protected:Nn \__rawobjects_ncgen_pr:nN
{ \__rawobjects_save_fun:N #2
 \cs_new_protected:cpn
 { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
 { \object_ncmember_adr:nnn{ #1 }{ #2 }{ #3 }
 }

\end{verbatim}

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

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

Generate ncmember versions of specified functions.

\begin{verbatim}
\cs_new_protected:Nn \__rawobjects_ncgen:nN
{ \__rawobjects_save_fun:N #2
 \cs_new:cpn { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
 { #2
 { \object_ncmember_adr:nnn{ #1 }{ #2 }{ #3 }
 } }
\end{verbatim}

\begin{verbatim}
\cs_new_protected:Nn \__rawobjects_ncgen_pr:nN
{ \__rawobjects_save_fun:N #2
 \cs_new_protected:cpn
 { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
 { #2

\end{verbatim}

29
\cs_new_protected:Nn \__rawobjects_ncgen:nnn
{\__rawobjects_save_dat:n { #3 }\cs_new:cpn { __rawobjects_auxfun_#1 :nn } ##1##2
  {\use:c{ #2 : #3 }\cs_generate_variant:cn { __rawobjects_auxfun_#1 :nn }{ nf }\cs_new_protected:cpn
   { #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str } ##1##2##3
     {\use:c { __rawobjects_auxfun_#1 :nf }
      { #3 }\__rawobjects_scope_pfx_cl:n{ #1 }
     }\object_ncmember_adr:nnn{ #1 }{ #2 }{ #3 }\cs_generate_variant:Nn \__rawobjects_ncgen:nN { fN }\cs_generate_variant:Nn \__rawobjects_ncgen:nnn { fnn }\cs_generate_variant:Nn \__rawobjects_ncgen:nnn { fnn }
30
Generate ncmember versions of specified functions.

(End definition for \object_ncmember_generate:NN and others. These functions are documented on page 16.)
\_rawobjects\_save\_dat:n \{ #3 \}
\cs\new\:cpn \{ \_rawobjects\_auxfun\_#1 :nn \} ##1##2
{ \use:c \{ #2 : #3 \} }
\cs\generate\_variant:cn \{ \_rawobjects\_auxfun\_#1 :nn \} \{ nf \}
\cs\new\:cpn \{ #1 : nnn \str\_use:N \l\_rawobjects\_tmp\_fa\_str \} ##1##2##3
{ \use:c \{ \_rawobjects\_auxfun\_#1 :nf \} ##3 }
{ \__rawobjects\_scope\_pfx\_cl:n\{ ##1 \} }
{ \object\_rcmember\_adr:nnn\{ ##1 \}\{ ##2 \}\{ ##3 \} }
\cs\generate\_variant:Nn \__rawobjects\_rcgen\_pr:nnn
{ \__rawobjects\_save\_dat:n \{ #3 \} }
\cs\new\:cpn \{ \_rawobjects\_auxfun\_#1 :nn \} ##1##2
{ \use:c \{ #2 : #3 \} }
\cs\generate\_variant:cn \{ \_rawobjects\_auxfun\_#1 :nn \} \{ nf \}
\cs\new\_protected:cpn
\{ #1 : nnn \str\_use:N \l\_rawobjects\_tmp\_fa\_str \} ##1##2##3
{ \use:c \{ \_rawobjects\_auxfun\_#1 :nf \} ##3 }
{ \__rawobjects\_scope\_pfx\_cl:n\{ ##1 \} }
{ \object\_rcmember\_adr:nnn\{ ##1 \}\{ ##2 \}\{ ##3 \} }
\cs\generate\_variant:Nn \__rawobjects\_rcgen\_pr:nnn
{ \__rawobjects\_save\_dat:n \{ #3 \} }
\cs\new\:cpn \{ \_rawobjects\_auxfun\_#1 :nn \} ##1##2
{ \use:c \{ #2 : #3 \} }
\cs\generate\_variant:cn \{ \_rawobjects\_auxfun\_#1 :nn \} \{ nf \}
\cs\new\_protected:cpn
\{ #1 : nnn \str\_use:N \l\_rawobjects\_tmp\_fa\_str \} ##1##2##3
{ \use:c \{ \_rawobjects\_auxfun\_#1 :nf \} ##3 }
{ \__rawobjects\_scope\_pfx\_cl:n\{ ##1 \} }
{ \object\_rcmember\_adr:nnn\{ ##1 \}\{ ##2 \}\{ ##3 \} }
\cs\generate\_variant:Nn \__rawobjects\_rcgen\:nnn
{ fN }
\cs\generate\_variant:Nn \__rawobjects\_rcgen\:nn \{ fnn \}
\cs\generate\_variant:Nn \__rawobjects\_rcgen\_pr:N \{ fn \}
\cs\generate\_variant:Nn \__rawobjects\_rcgen\_pr:nnn \{ fnn \}
\cs\new\_protected:Nn \object\_rcmember\_generate\:NN
\{ \__rawobjects\_rcgen:fN \{ \cs\to\_str:N \#1 \} \#2 \}

32
\cs_new_protected:Nn \object_rcmember_generate_inline:Nnn
{ \__rawobjects_rcgen:fnn \cs_to_str:N #1 \#2 \#3 }
\cs_new_protected:Nn \object_rcmember_generate_protected:NN
{ \__rawobjects_rcgen_pr:fn \cs_to_str:N #1 \#2 }
\cs_new_protected:Nn \object_rcmember_generate_protected_inline:Nnn
{ \__rawobjects_rcgen_pr:fnn \cs_to_str:N #1 \#2 \#3 }

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

Auxiliary functions

\cs_generate_variant:Nn \cs_generate_variant:Nn { cx }
\cs_new_protected:Nn \__rawobjects_genmem_int:nnn
{ \__rawobjects_mgen:nnn \cs_generate_variant:cx
   #1 \#2 \#3 \cs_generate_variant:cx
   \str_use:N \l__rawobjects_tmp_fa_str
   { Vnn \str_use:N \l__rawobjects_tmp_fa_str, nnv \str_use:N \l__rawobjects_tmp_fa_str }
   { Vn \str_use:N \l__rawobjects_tmp_fa_str }
}
\cs_new_protected:Nn \__rawobjects_genmem_pr_int:nnn
{ \__rawobjects_mgen_pr:nnn \cs_generate_variant:cx
   #1 \#2 \#3 \cs_generate_variant:cx
   \str_use:N \l__rawobjects_tmp_fa_str
   { Vnn \str_use:N \l__rawobjects_tmp_fa_str, nnv \str_use:N \l__rawobjects_tmp_fa_str }
   { Vn \str_use:N \l__rawobjects_tmp_fa_str }
}
\cs_new_protected:Nn \__rawobjects_genncm_int:nnn
{ \__rawobjects_ncgen:nnn \cs_generate_variant:cx
   #1 \#2 \#3 \cs_generate_variant:cx
   \str_use:N \l__rawobjects_tmp_fa_str
   { Vnn \str_use:N \l__rawobjects_tmp_fa_str, nnv \str_use:N \l__rawobjects_tmp_fa_str }
}
\cs_new_protected:Nn \__rawobjects_genncm_pr_int:nnn
{ \__rawobjects_ncgen_pr:nnn \cs_generate_variant:cx
   #1 \#2 \#3 \cs_generate_variant:cx
   \str_use:N \l__rawobjects_tmp_fa_str
   { Vnn \str_use:N \l__rawobjects_tmp_fa_str, nnv \str_use:N \l__rawobjects_tmp_fa_str }
}
\begin{verbatim}
\cs_new_protected:Nn \__rawobjects_genrcm_int:nnn { \__rawobjects_rgennen:n { #1 }{ #2 }{ #3 } \cs_generate_variant:cx
\{ #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str \}
\{ Vnn \str_use:N \l__rawobjects_tmp_fa_str \}
}\cs_new_protected:Nn \__rawobjects_genrcm_pr_int:nnn { \__rawobjects_rgennen_pr:n { #1 }{ #2 }{ #3 } \cs_generate_variant:cx
\{ #1 : nnn \str_use:N \l__rawobjects_tmp_fa_str \}
\{ Vnn \str_use:N \l__rawobjects_tmp_fa_str \}

\msg_new:nnnn { rawobjects }{ noerr }{ Unspecified \ - \ scope }{ Object \ - \ #1 \ - \ hasn’t \ - \ a \ - \ scope \ - \ variable }

\object_new_member:nnn \object_new_member_tracked:nnn
Creates a new member variable

\__rawobjects_genmem_pr_int:nnn { object_new_member }{ #1 _ new }{ c }
\cs_new_protected:Nn \object_new_member_tracked:nnn
\{ \object_new_member:nnn { #1 }{ #2 }{ #3 } \str_const:cn
\{ \object_ncmember_adr:nnn
\{ \object_embedded_adr:n { #1 }{ /_T_/ }
\{ #2 _ type }{ str }
\{ #3 }
\}
\cs_generate_variant:Nn \object_new_member_tracked:nnn { Vnn, nnv }

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

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

\__rawobjects_genmem_int:nnn {object_member_use}{ #1_use }{c}
\end{verbatim}

34
\texttt{%generate_variant:Nn \object_member_use:nnn \{vnn\}}

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

\texttt{\object_member_set:nnnn \object_member_set:nnn}
Set the value a member.

\texttt{%rawobjects_genmem_pr_int:nnn \{object_member_set\}\{ \#1_\#2 set \}\{ cn \}}

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

\texttt{\object_member_set_eq:nnnN \object_member_set_eq:nnN}
Make a member equal to another variable.

\texttt{%rawobjects_genmem_pr_int:nnn \{object_member_set_eq\}\{ \#1_\#2 set_eq \}\{ cn \}}

\texttt{%generate_variant:Nn \object_member_set_eq:nnnN \{ nnnn, Vnnn \}}
\texttt{%generate_variant:Nn \object_member_set_eq:nnN \{ nnn, Vnn \}}

(End definition for \texttt{\object_member_set_eq:nnnN} and \texttt{\object_member_set_eq:nnN}. These functions are documented on page 9.)

\texttt{\object_ncmember_adr:nnn}
Get address of near constant

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

\texttt{%generate_variant:Nn \object_ncmember_adr:nnn \{ Vnn, vnn \}}

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

\texttt{\object_rcmember_adr:nnn}
Get the address of a remote constant.

\texttt{%cs_new:Nn \object_rcmember_adr:nnn}
\texttt{\object_ncmember_adr:vnn}
\texttt{\object_embedded_adr:nn\{ \#1 \}/_/}
\texttt{\{ P \}\{ str \}}
\texttt{\{ \#2 \}\{ \#3 \}}

\texttt{%generate_variant:Nn \object_rcmember_adr:nnn \{ Vnn \}}

(End definition for \texttt{\object_rcmember_adr:nnn}. This function is documented on page 10.)
Tests if the specified member constant exists.

\begin{verbatim}
\newconditional: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:
}
\end{verbatim}

\begin{verbatim}
\newconditional: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:
}
\end{verbatim}

\begin{verbatim}
\generateconditionalvariant:Nnn \object_ncmember_if_exist:nnn
  \{ Vnn \} { p, T, F, TF }
\end{verbatim}

(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
\object_rcmember_use:nnn
Uses a near/remote constant.

\begin{verbatim}
\__rawobjects_genncm_int:nnn { object_ncmember_use }{ #1_use}{ c }
\__rawobjects_genrcm_int:nnn { object_rcmember_use }{ #1_use}{ c }
\end{verbatim}

(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

\begin{verbatim}
\__rawobjects_genncm_pr_int:nnn { object_newconst }{ #1 _ const }{ cn }
\end{verbatim}

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

Create 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 }
\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 12.)

\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 12.)

\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 }
\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 11.)
Tests if the specified member constant exists.

\begin{verbatim}
\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 }
\prg_generate_conditional_variant:Nnn \object_rcmethod_if_exist:nnn { Vnn }
\end{verbatim}

(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

\begin{verbatim}
\cs_new_protected:Nn \object_new_cmethod:nnnn
  { \cs_new:cn
    \object_ncmethod_adr:nnn { #1 }{ #2 }{ #3 }
    \object_ncmethod_adr:nnn { #1 }{ #2 }{ #3 }
    \object_ncmethod_adr:nnn { #1 }{ #2 }{ #3 }
    \cs_generate_variant:Nn \object_new_cmethod:nnnn { Vnnn }
\end{verbatim}

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

\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 \object_if_proxy:n T F
Test if an object is a proxy.

\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 }
1075 } \bool_if:cTF
1076 { \object_ncmember_adr:nnn
1077 { \object_embedded_adr:n{ #1 }{ /I_/ } }
1078 } { \ifprox }{ bool }
1079 }
1080 { \prg_return_true:
1081 } { \prg_return_false:
1082 }
1083 { \prg_return_false:
1084 }
1085 { \prg_return_true:
1086 } { \prg_return_false:
1087 }
1088 { \prg_return_false:
1089 }
1090 }
1091 { \prg_return_false:
1092 }
1093 }
1094
1095 (End definition for \object_if_proxy:nTF. This function is documented on page 13.)

\object_test_proxy_p:nn \object_test_proxy:nn TF \object_test_proxy_p:nN \object_test_proxy:nN
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:n{ #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
  }

41
\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 13.)

\object_create:nnnNN
\object_create_set:NnnnNN
\object_create_gset:NnnnNN
\object_create:nnnN
\object_create_set:NnnnN
\object_create_gset:NnnnN
\embedded_create:nnn

Creates an object from a proxy.
\object_newconst_str:nnV
{ \object_embedded_adr:nn{ #3 }{ /_I_/ } }
\object_newconst_str:nnV
{ \object_embedded_adr:nn{ #3 }{ /_I_/ } }
\seq_map_inline:cn
{ \object_member_adr:nnn { #1 }{ varlist }{ seq } }
\seq_map_inline:cn
{ \object_member_adr:nnn { #1 }{ objlist }{ seq } }
\seq_map_inline:cn
{ \object_member_adr:nnn { #1 }{ init }{ tl } }
\cs_generate_variant:Nn \__rawobjects_create_anon:nnnNN { xnxNN, xvxcc }
\cs_new_protected:Nn \object_create:nnnNN
{ \__rawobjects_create_anon:xnxNN { #1 }{ #2 }{ #3 } }
\object_address:nn { #2 }{ #3 }
\cs_generate_variant:Nn \object_create:nnnNN { VnnNN }
\cs_new_protected:Nn \object_create_set:NnnnnNN
{\object_create:nnnNN { #2 }{ #3 }{ #4 } #5 #6
 \str_set:Nx #1 { \object_address:nn { #3 }{ #4 } }
}
\cs_new_protected:Nn \object_create_gset:NnnnnNN
{\object_create:nnnNN { #2 }{ #3 }{ #4 } #5 #6
 \str_gset:Nx #1 { \object_address:nn { #3 }{ #4 } }
}
\cs_generate_variant:Nn \object_create_set:NnnnnNN { NVnnNN, NnnfNN }
\cs_generate_variant:Nn \object_create_gset:NnnnnNN { NVnnNN, NnnfNN }
\cs_new_protected:Nn \object_create:nnnN
{\object_create:nnnNN { #1 }{ #2 }{ #3 }
 \c_object_public_str
}
\cs_generate_variant:Nn \object_create:nnnN { VnnN }
\cs_new_protected:Nn \object_create_set:NnnnnN
{\object_create_set:NnnnnNN #1 { #2 }{ #3 }{ #4 }
 \c_object_global_str \c_object_public_str
}
\cs_new_protected:Nn \object_create_gset:NnnnnN
{\object_create_gset:NnnnnNN #1 { #2 }{ #3 }{ #4 }
 \c_object_global_str \c_object_public_str
}
\cs_generate_variant:Nn \object_create_set:NnnnnN { NVnnN }
\cs_generate_variant:Nn \object_create_gset:NnnnnN { NVnnN }
\cs_new_protected:Nn \object_create:nnn
{\object_create:nnnNN { #1 }{ #2 }{ #3 }
 \c_object_global_str \c_object_public_str
}
\cs_generate_variant:Nn \object_create:nnn { Vnn }
\cs_new_protected:Nn \object_create_set:Nnnn
{\object_create_set:NnnnnNN #1 { #2 }{ #3 }{ #4 }
 \c_object_global_str \c_object_public_str
}
\cs_new_protected:Nn \object_create_gset:Nnnn
  { \object_create_gset:Nnnn #1 { #2 }{ #3 }{ #4 }
   \c_object_global_str \c_object_public_str }
\cs_new_protected:Nn \embedded_create:nnn
  { \_rawobjects_create_anon:xvxcc { #2 }
    { \object_ncmember_adr:nnn
      { \object_embedded_adr:nn { #1 }{ /_I_/ } }{ M }{ str }
    }{ M }{ str }
    { \object_ncmember_adr:nnn
      { \object_embedded_adr:nn { #1 }{ /_I_/ } }{ S }{ str }
    }{ S }{ str }
    { \object_ncmember_adr:nnn
      { \object_embedded_adr:nn { #1 }{ /_I_/ } }{ V }{ str }
    }{ V }{ str }
  }{ nvn, Vnn }
\cs_generate_variant:Nn \embedded_create:nnn { nvn, Vnn }

(End definition for \object_create:nnnNnN 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

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

45
Push a new member inside a proxy.

\proxy_push_member:nnn
\begin{verbatim}
c_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 }
}
c_generate_variant:Nn \proxy_push_member:nnn { Vnn }
\end{verbatim}

(End definition for \proxy_create:nn, \proxy_create_set:Nnn, and \proxy_create_gset:Nnn. These functions are documented on page 14.)
\proxy_push_embedded:nnn  Push a new embedded object inside a proxy.
\begin{verbatim}
\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{verbatim}
\end{definition}

\proxy_add_initializer:nN  Push a new embedded object inside a proxy.
\begin{verbatim}
\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{verbatim}
\end{definition}

\c_proxy_address_str  Variable containing the address of the proxy object.
\begin{verbatim}
\str_const:Nx \c_proxy_address_str
    { \object_address:nn { rawobjects }{ proxy } }
\object_newconst_str:nnn
    { \object_member_adr:Vn \c_proxy_address_str { /_I_/ } \tl }
\object_newconst_str:nnV
    { \object_member_adr:Vn \c_proxy_address_str { /_I_/ } }
\object_newconst_str:nnV
\end{verbatim}

47
\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 19.)

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 } 
{ \cs_to_str:N #1 } 
}
\cs_new:Nn \__rawobjects_combine:Nn
{ 
\__rawobjects_combine:Nn #1 { #2 } 
}
\cs_new_protected:Nn \object_gallocate_incr:NNnnNN 
{ \object_create_gset:NnnfNN #1 { #3 }{ #4 }
  \__rawobjects_combine:Nn #2 { #3 }
  \int_incr:N #2
}

\cs_new_protected:Nn \object_gallocate_gincr:NNnnNN 
{ \object_create_set:NnnfNN #1 { #3 }{ #4 }
  \__rawobjects_combine:Nn #2 { #3 }
  \int_gincr:N #2
}

\cs_new_protected:Nn \object_allocate_incr:NNnnNN 
{ \object_create_gset:NnnfNN #1 { #3 }{ #4 }
  \__rawobjects_combine:Nn #2 { #3 }
  \int_incr:N #2
}

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

\cs_generate_variant:Nn \object_allocate_incr:NNnnNN { NNVnNN }
\cs_generate_variant:Nn \object_gallocate_incr:NNnnNN { NNVnNN }
\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 14.)

\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_nmember_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 15.)