1 Introduction

First to all notice that lt3rawobjects means “raw object(s)”, indeed lt3rawobjects introduces a new mechanism to create 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.

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 Instead changes introduced in minor and patch version updates are always backward compatible, and any withdrawn function is declared deprecated instead of being removed.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 To do

- Uniform declarations for templated proxies;
- Constant objects.

3 Objects and proxies

Usually an object in programming languages can be seen as a collection of variables (organized in different ways depending on the chosen language) treated as part of a single entity. In \texttt{lt3rawobjects} objects are collections of

- \LaTeX variables, called \textit{members};
- \LaTeX functions, called \textit{methods}.

Both members and methods can be retrieved from a string representing the container object, that is the \textit{address} of the object and act like the address of a structure in C.

An address is composed of two parts: the \textit{module} in which variables are created and an \textit{identifier} that identify uniquely the object inside its module. It’s up to the caller that two different objects have different identifiers. The address of an object can be obtained with the \texttt{\textbackslash object\_address} function. Identifiers and module names should not contain numbers, \_, \# characters in order to avoid conflicts with hidden auxiliary commands. However you can use non letter characters like - in order to organize your members and methods.

Moreover normal control sequences have an address too, but it’s simply any token list for which a \texttt{\textbackslash c} expansion retrieves the original control sequence. We impose also that any \texttt{x} or \texttt{e} fully expansion will be a string representing the control sequence’s name, for this reason inside an address \# characters and \texttt{\textbackslash exp\_not} functions aren’t allowed.

In \texttt{lt3rawobjects} objects are created from an existing object that have a suitable inner structure. These objects that can be used to create other objects are called \textit{proxy}. Every object is generated from a particular proxy object, called \textit{generator}, and new objects can be created from a specified proxy with the \texttt{\textbackslash object\_create} functions.

Since proxies are themself objects we need a proxy to instantiate user defined proxies, you can use the \texttt{\textbackslash proxy} object in the \texttt{rawobjects} module to create you own proxy, which address is held by the \texttt{\textbackslash c\_proxy\_address\_str} variable. Proxies must be created from the \texttt{\textbackslash proxy} object otherwise they won’t be recognized as proxies. Instead of using \texttt{\textbackslash object\_create} to create proxies you can directly use the function \texttt{\textbackslash proxy\_create}.

Each member or method inside an object belongs to one of these categories:

1. \textit{mutables};
2. \textit{near constants};
3. \textit{remote constants}.

\textbf{Warning}: Currently only members (variables) can be mutables, not methods. Mutable members can be added in future releases if they’ll be needed.

Members declared as mutables works as normal variables: you can modify their value and retrieve it at any time. Instead members and methods declared as near constant works as constants: when you create them you must specify their initial value (or function body for methods) and you won’t be allowed to modify it later. Remote constants for
an object are simply near constants defined in its generator: all near constants defined inside a proxy are automatically visible as remote constants to every object generated from that proxy. Usually functions involving near constants have \texttt{nc} inside their name, and \texttt{rc} if instead they use remote constants.

Instead of creating mutable members in each of your objects you can push their specifications inside the generating proxy via \texttt{proxy\_push\_member}. In this way either object created from such proxy will have the specified members. Specify mutable members in this way allows you to omit that member type in some functions as \texttt{object\_member\_adr} for example, their member type will be deduced automatically from its specification inside generating proxy.

Objects can be declared public, private and local, global. In a public/private object every nonconstant member and method is declared public/private, but inside local/global object only assignation to mutable members is performed locally/globally since allocation is always performed globally via \texttt{(type)\_new:NN} functions (nevertheless members will be accordingly declared \texttt{g\_} or \texttt{l\_}). This is intentional in order to follow the \LaTeX{}3 guidelines about variables management, for additional motivations you can see this thread in the \LaTeX{}3 repository.

Address of members/methods can be obtained with functions in the form \texttt{object\_\langle\texttt{category}\rangle\langle\texttt{item}\rangle\_adr} where \texttt{\langle\texttt{item}\rangle} is \texttt{member} or \texttt{method} and \texttt{\langle\texttt{category}\rangle} is empty for mutable members, \texttt{nc} for near constants and \texttt{rc} for remote constants. For example \texttt{object\_rcmethod\_adr} retrieves the address of specified remote constant method.

\section{Library functions}

### 4.1 Base object functions

\begin{align*}
\texttt{object\_address:nn} & \quad \texttt{\langle\texttt{module}\rangle\{\langle\texttt{id}\rangle\}} \\
\texttt{object\_address_set:nn} & \quad \texttt{\langle\texttt{str\ var}\rangle\{\langle\texttt{module}\rangle\{\langle\texttt{id}\rangle\}} \\
\texttt{object\_get_module:n} & \quad \texttt{\langle\texttt{address}\rangle} \\
\texttt{object\_get_proxy_adr:n} & \quad \texttt{\langle\texttt{address}\rangle}
\end{align*}

Composes the address of object in module \texttt{\langle\texttt{module}\rangle} with identifier \texttt{\langle\texttt{id}\rangle} and places it in the input stream. Notice that \texttt{\langle\texttt{module}\rangle} and \texttt{\langle\texttt{id}\rangle} are converted to strings before composing them in the address, so they shouldn’t contain any command inside. If you want to execute its content you should use a new variant, for example \texttt{V}, \texttt{f} or \texttt{e} variants.

From: \texttt{1.0}

\begin{align*}
\texttt{object\_address_set:Vn} & \quad \texttt{\langle\texttt{str\ var}\rangle\{\langle\texttt{module}\rangle\{\langle\texttt{id}\rangle\}} \\
\texttt{object\_address_gset:Vn} & \quad \texttt{\langle\texttt{address}\rangle} \\
\texttt{object\_get_module:V} & \quad \texttt{\langle\texttt{address}\rangle} \\
\texttt{object\_get_proxy_adr:V} & \quad \texttt{\langle\texttt{address}\rangle}
\end{align*}

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

From: \texttt{1.1}

\begin{align*}
\texttt{object\_if\_exist:p:n} & \quad \texttt{\langle\texttt{address}\rangle} \\
\texttt{object\_if\_exist:p:V} & \quad \texttt{\langle\texttt{address}\rangle} \\
\texttt{object\_if\_exist:nTF} & \quad \texttt{\langle\texttt{address}\rangle} \{\langle\texttt{true\ code}\rangle\} \{\langle\texttt{false\ code}\rangle\}
\end{align*}

Tests if an object was instantiated at the specified address.

From: \texttt{1.0}

\begin{align*}
\texttt{object\_get_module:n} & \quad \texttt{\langle\texttt{address}\rangle} \\
\texttt{object\_get_module:V} & \quad \texttt{\langle\texttt{address}\rangle} \\
\texttt{object\_get_proxy_adr:n} & \quad \texttt{\langle\texttt{address}\rangle} \\
\texttt{object\_get_proxy_adr:V} & \quad \texttt{\langle\texttt{address}\rangle}
\end{align*}

Get the object module and its generator.

From: \texttt{1.0}
Tests if the object is local or global.

From: 1.0

4.2 Members

Fully expands to the address of specified member variable. If type is not specified it'll be retrieved from the generator proxy, but only if member is specified in the generator.

From: 1.0

Tests if the specified member exist.

From: 2.0

Fully expands to the type of member \textit{(member name)}. Use this function only with member variables specified in the generator proxy, not with other member variables.

From: 1.0

Creates a new member variable with specified name and type. You can't retrieve the type of these variables with \texttt{object\_member\_type} functions.

From: 1.0
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\object_member_use:nnn</code></td>
<td>Uses the specified member variable.</td>
</tr>
<tr>
<td>`\object_member_use:(Vnn</td>
<td>nnv)`</td>
</tr>
<tr>
<td><code>\object_member_use:nn</code></td>
<td>*</td>
</tr>
<tr>
<td><code>\object_member_use:Vn</code></td>
<td>*</td>
</tr>
<tr>
<td><code>\object_member_set_eq:nnnN</code></td>
<td>Sets the value of specified member equal to the value of (variable).</td>
</tr>
<tr>
<td>`\object_member_set_eq:(nnvN</td>
<td>VnnN</td>
</tr>
<tr>
<td>`\object_member_set_eq:(VnN</td>
<td>nncc</td>
</tr>
<tr>
<td><code>\object_ncmember_adr:nnn</code></td>
<td>Fully expands to the address of specified near/remote constant member.</td>
</tr>
<tr>
<td>`\object_ncmember_adr:(Vnn</td>
<td>vnn)`</td>
</tr>
<tr>
<td><code>\object_rcmember_adr:nnn</code></td>
<td>*</td>
</tr>
<tr>
<td><code>\object_rcmember_adr:Vnn</code></td>
<td>*</td>
</tr>
<tr>
<td><code>\object_ncmember_if_exist_p:nnn</code></td>
<td>Tests if the specified member constant exist.</td>
</tr>
<tr>
<td>`\object_ncmember_if_exist_p:(Vnn</td>
<td>vnn)`</td>
</tr>
<tr>
<td><code>\object_rcmember_if_exist_p:nnn</code></td>
<td>*</td>
</tr>
<tr>
<td><code>\object_rcmember_if_exist_p:Vnn</code></td>
<td>*</td>
</tr>
<tr>
<td><code>\object_ncmember_use:nnn</code></td>
<td>Uses the specified near/remote constant member.</td>
</tr>
<tr>
<td><code>\object_ncmember_use:Vn</code></td>
<td>*</td>
</tr>
<tr>
<td><code>\object_rcmember_use:nnn</code></td>
<td>*</td>
</tr>
<tr>
<td><code>\object_rcmember_use:Vn</code></td>
<td>*</td>
</tr>
</tbody>
</table>

### 4.3 Methods

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

Tests if the specified method constant exist.
From: 2.0

Creates a new method with specified name and argument types. The `{method arguments}` should be a string composed only by n and N characters that are passed to `\cs_new:Nn`.
From: 2.0

Calls the specified method. This function is expandable if and only if the specified method was not declared protected.
From: 2.0

### 4.4 Constant member creation

Unlike normal variables, constant variables in \LaTeX are created in different ways depending on the specified type. So we dedicate a new section only to collect some of these functions readapted for near constants (remote constants are simply near constants created on the generator proxy).
\object_newconst_{type}:nnn \{\langle address\rangle\} \{\langle constant name\rangle\} \{\langle value\rangle\}

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

From: 1.1

\object_newconst_seq_from_clist:nnn \{\langle address\rangle\} \{\langle constant name\rangle\} \{\langle comma-list\rangle\}

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

From: 1.1

\object_newconst_prop_from_keyval:nnn \{\langle address\rangle\} \{\langle constant name\rangle\} \{\langle key\rangle = \langle value\rangle, \ldots\}

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

From: 1.1

4.5 Proxy utilities and object creation

\object_if_proxy_p:n \{\langle address\rangle\}
\object_if_proxy:nTF \{\langle address\rangle\} \{\langle true code\rangle\} \{\langle false code\rangle\}

Test if the specified object is a proxy object.

From: 1.0

\object_test_proxy_p:nn \{\langle object address\rangle\} \{\langle proxy address\rangle\}
\object_test_proxy:nTF \{\langle object address\rangle\} \{\langle proxy address\rangle\} \{\langle true code\rangle\} \{\langle false code\rangle\}

Test if the specified object is generated by the selected proxy, where \langle proxy variable\rangle is a string variable holding the proxy address.

\textbf{\LaTeX\hackers note:} Remember that this command uses internally an \texttt{e} expansion so in older engines (any different from \LaTeX 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
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.

\textbf{From: 2.0}

The address of the proxy object in the rawobjects module.

\textbf{From: 1.0}

Creates an object by using the proxy at ⟨proxy address⟩ and the specified parameters.

\textbf{From: 1.0}

Possible values for ⟨scope⟩ parameter.

\textbf{From: 1.0}

Possible values for ⟨visibility⟩ parameter.

\textbf{From: 1.0}

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.

\textbf{From: 1.1}

Creates a global proxy object.

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

From: 1.0

object_assign:nn {to address} {from address}
Assigns the content of each variable of object at {from address} to each corresponsive variable in {to address}. Both the objects should be created with the same proxy object and only variables listed in the proxy are assigned.

From: 1.0

5 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 \l_myproxy_str.

\str_new:N \l_myproxy_str
\proxy_create_set:NnnN \l_myproxy_str { example }{ myproxy } \c_object_public_str
\proxy_push_member:Vnn \l_myproxy_str { myvar }{ tl }

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

\str_new:N \l_myobj_str
\object_create_set:NVnnNN \l_myobj_str \l_myproxy_str
{ example }{ myobj } \c_object_local_str \c_object_public_str
\tl_set:cn
{ \object_member_adr:Vn \l_myobj_str { myvar } }
{ \c_dollar_str{} - dollar - \c_dollar_str{} } \object_member_use:Vn \l_myobj_str { myvar }

Output: $ dollar $
If you don’t want to specify an object identifier you can also do

\int_new:N \l_intc_int
\object_allocate_incr:NVnnNN \l_myobj_str \l_intc_int \l_myproxy_str
{ example } \c_object_local_str \c_object_public_str
\tl_set:cn
{ \object_member_adr:Vn \l_myobj_str { myvar } }
{ \c_dollar_str{} - dollar - \c_dollar_str{} } \object_member_use:Vn \l_myobj_str { myvar }

Output: $ dollar $
6 Templated proxies

At the current time there isn’t a standardized approach to templated proxies. One problem of standardized templated proxies is how to define struct addresses for every kind of argument (token lists, strings, integer expressions, non expandable arguments, ...).

Even if there isn’t currently a function to define every kind of templated proxy you can anyway define your templated proxy with your custom parameters. You simply need to define at least two functions:

- an expandable macro that, given all the needed arguments, fully expands to the address of your templated proxy. This address can be obtained by calling \object_address \{ \langle module \rangle \} \{ \langle id \rangle \} where \langle id \rangle starts with the name of your templated proxy and is followed by a composition of specified arguments;

- a not expandable macro that tests if the templated proxy with specified arguments is instantiated and, if not, instantiate it with different calls to \proxy_create and \proxy_push_member.

In order to apply these concepts we’ll provide a simple implementation of a linked list with a template parameter representing the type of variable that holds our data. A linked list is simply a sequence of nodes where each node contains your data and a pointer to the next node. For the moment we’ll show a possible implementation of a template proxy class for such node objects.

First to all we define an expandable macro that fully expands to our node name:

```
\cs_new:Nn \node_address:n
{ \object_address:nn \{ \langle module \rangle \} \{ \langle id \rangle \} }
```

where the \#1 argument is simply a string representing the type of data held by our linked list (for example \texttt{tl}, \texttt{str}, \texttt{int}, ...). Next we need a functions that instantiate our proxy address if it doesn’t exist:

```
\cs_new_protected:Nn \node_instantiate:n
{ \object_if_exist:nF { \node_address:n \{ \#1 \} }
  { \proxy_create:nnN \{ \langle module \rangle \} \{ \langle id \rangle \} \c_object_public_str \proxy_push_member:nnn \{ \node_address:n \{ \#1 \} \} \{ \langle next \rangle \} \{ \texttt{str} \} \proxy_push_member:nnn \{ \node_address:n \{ \#1 \} \} \{ \langle data \rangle \} \{ \#1 \} }
}
```

As you can see when \node_instantiate is called it first test if the proxy object exists. If not then it creates a new proxy with that name and populates it with the specifications of two members: a \texttt{next} member variable of type \texttt{str} that points to the next node, and a \texttt{data} member of the specified type that holds your data.

Clearly you can define new functions to work with such nodes, for example to test if the next node exists or not, to add and remove a node, search inside a linked list, ...
7 Implementation

\(\texttt{\langle \texttt{\#rawobjects} \rangle}\)

\c_object_local_str
\c_object_global_str
\c_object_public_str
\c_object_private_str

\str_const:Nn \c_object_local_str \{loc\}
\str_const:Nn \c_object_global_str \{glo\}
\str_const:Nn \c_object_public_str \{pub\}
\str_const:Nn \c_object_private_str \{pri\}

\str_const:Nn \c__rawobjects_const_str \{con\}

(End definition for \c_object_local_str and others. These variables are documented on page 8.)

\object_address:nn
Get address of an object

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

(End definition for \object_address:nn. This function is documented on page 3.)

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

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

(End definition for \object_address_set:Nnn and \object_address_gset:Nnn. These functions are documented on page 3.)

\cs_new:Nn \__rawobjects_object_modvar:n{
  c __ #1 _ MODULE _ str
}
\cs_new:Nn \__rawobjects_object_pxyvar:n{
  c __ #1 _ PROXY _ str
}
\cs_new:Nn \__rawobjects_object_scovar:n{
  c __ #1 _ SCOPE _ str
}
\cs_new:Nn \__rawobjects_object_visvar:n{
  c __ #1 _ VISIB _ str
}
\cs_generate_variant:Nn \__rawobjects_object_modvar:n { V }
\cs_generate_variant:Nn \__rawobjects_object_pxyvar:n { V }
\cs_generate_variant:Nn \__rawobjects_object_scovar:n { V }
\cs_generate_variant:Nn \__rawobjects_object_visvar:n { V }


\textbf{\object_if_exist_p:n} Tests if object exists.

\begin{verbatim}
\prg_new_conditional:Nnn \object_if_exist:n { p, T, F, TF }
  { \cs_if_exist:cTF
    { \_rawobjects_object_modvar:n { #1 }
      { \prg_return_true: }
      { \prg_return_false: }
    }
  }
\prg_generate_conditional_variant:Nnn \object_if_exist:n { V }
  { p, T, F, TF }
\end{verbatim}

(End definition for \object_if_exist:nTF. This function is documented on page 3.)

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

\begin{verbatim}
\cs_new:Nn \object_get_module:n { \str_use:c { \_rawobjects_object_modvar:n { #1 } } }
\cs_new:Nn \object_get_proxy_adr:n { \str_use:c { \_rawobjects_object_pxyvar:n { #1 } } }
\cs_generate_variant:Nn \object_get_module:n { V }
\cs_generate_variant:Nn \object_get_proxy_adr:n { V }
\end{verbatim}

(End definition for \object_get_module:n and \object_get_proxy_adr:n. These functions are documented on page 3.)

\textbf{\object_if_local_p:n} Test the specified parameters.

\begin{verbatim}
\prg_new_conditional:Nnn \object_if_local:n { p, T, F, TF }
  { \str_if_eq:cNTF { \_rawobjects_object_scovar:n {#1} } { 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 { \_rawobjects_object_scovar:n {#1} } { 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 { \_rawobjects_object_scovar:n {#1} } { c_object_public_str
      { \prg_return_true: }
      { \prg_return_false: }
    }
  }
\prg_new_conditional:Nnn \object_if_private:n { p, T, F, TF }
  { \str_if_eq:cNTF { \_rawobjects_object_scovar:n {#1} } { c_object_private_str
      { \prg_return_true: }
      { \prg_return_false: }
    }
  }
\end{verbatim}

12
\prg_return_true:
} {
\prg_return_false:
}

\prg_new_conditional:Nnn \object_if_public:n { p, T, F, TF }
{
\str_if_eq:cNTF { \__rawobjects_object_visvar:n { #1 } }
\c_object_public_str
{
\prg_return_true:
}
\prg_return_false:
}

\prg_new_conditional:Nnn \object_if_private:n { p, T, F, TF }
{
\str_if_eq:cNTF { \__rawobjects_object_visvar:n { #1 } }
\c_object_private_str
{
\prg_return_true:
}
\prg_return_false:
}

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

(End definition for \object_if_local:nTF and others. These functions are documented on page 4.)

\object_member_adr:nnn
\object_member_adr:nnn
Get the address of a member variable

\cs_new:Nn \_rawobjects_scope:n
{
\object_if_local:nTF { #1 }
{
1
}
{\str_if_eq:cNTF { \__rawobjects_object_scovar:n { #1 } }
\c__rawobjects_const_str
}
\cs_new:Nn \__rawobjects_scope_pfx:n
{
\object_if_local:nF { #1 }
{ g }
}
\cs_new:Nn \__rawobjects_vis_var:n
{\object_if_private:nTF { #1 }
{ --
}
{ --
}
}
\cs_new:Nn \__rawobjects_vis_fun:n
{\object_if_private:nT { #1 }
{ --
}
}
\cs_new:Nn \object_member_adr:nnn
{\__rawobjects_scope:n { #1 }
\__rawobjects_vis_var:n { #1 }
#1 \tl_to_str:n { _ MEMBER _ #2 _ #3 }
}
\cs_generate_variant:Nn \object_member_adr:nnn { Vnn, vnn, nnv }
\cs_new:Nn \object_member_adr:nn
{\object_member_adr:nnv { #1 }{ #2 }
{ object_rcmember_adr:nnn { #1 }
  { #2 _ type }{ str }
}
}
\cs_generate_variant:Nn \object_member_adr:nn { Vn }

(End definition for \object_member_adr:nnn and \object_member_adr:nn. These functions are documented on page 4.)
\object_member_type:nn

Deduce the member type from the generating proxy.

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

(End definition for \object_member_type:nn. This function is documented on page 4.)

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

\cs_new_protected:Nn \__rawobjects_force_scope:n
{\bool_if:nF
{\object_if_local_p:n { #1 } || \object_if_global_p:n { #1 }
} \msg_error:nnx { rawobjects }{ scoperr }{ #1 }
}

\object_member_if_exist:nnn \object_member_if_exist:nn
\object_member_if_exist:nn
\object_member_if_exist:nnn TF

Tests if the specified member exists

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

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

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

(End definition for \object_member_if_exist:nnnTF and \object_member_if_exist:nnTF. These functions are documented on page 4.)

\object_new_member:nnn Creates a new member variable

\cs_new_protected:Nn \object_new_member:nnn
{
  \__rawobjects_force_scope:n { #1 }
  \cs_if_exist_use:cT { #3 _ new:c }
  {
    \object_member_adr:nnn { #1 }{ #2 }{ #3 }
  }
}
\cs_generate_variant:Nn \object_new_member:nnn { Vnn, nnv }

(End definition for \object_new_member:nnn. This function is documented on page 4.)

\object_member_use:nnn \object_member_use:nn

\cs_new:Nn \object_member_use:nnn
{
  \cs_if_exist_use:cT { #3 _ use:c }
  {
    \object_member_adr:nnn { #1 }{ #2 }{ #3 }
  }
}
\cs_generate_variant:Nn \object_new_member:nnn { Vnn, vnn, nnv }
\cs_generate_variant:Nn \object_member_use:nn { Vn }

(End definition for \object_member_use:nnn and \object_member_use:nn. These functions are documented on page 5.)
\end{verbatim}
Set the value of a variable to a member.
\begin{verbatim}
\cs_new_protected:Nn \object_member_set_eq:nnnN { #1 }
  \cs_if_exist_use:cT
    { \_rawobjects_scope_pfx:n { #1 } set _ eq:cN }
  { \object_member_adr:nnn { #1 }{ #2 }{ #3 } } #4
\end{verbatim}
\begin{verbatim}
\cs_generate_variant:Nn \object_member_set_eq:nnnN { VnnN, nnnc, Vnnc, nnvN }
\end{verbatim}
\begin{verbatim}
\cs_new_protected:Nn \object_member_set_eq:nnN { #1 }{ #2 }
  \object_rcmember_adr:nnn { #1 }{ #2 _ type }{ str }
\end{verbatim}
\begin{verbatim}
\cs_generate_variant:Nn \object_member_set_eq:nnnN { VnN, nnc, Vnc }
\end{verbatim}

Get the address of a near/remote constant.
\begin{verbatim}
\cs_new:Nn \object_ncmember_adr:nnn { c _ #1 \tl_to_str:n { _ CONST _ #2 _ #3 } }
\end{verbatim}
\begin{verbatim}
\cs_generate_variant:Nn \object_ncmember_adr:nnn { Vnn, vnn }
\end{verbatim}
\begin{verbatim}
\cs_new:Nn \object_rcmember_adr:nnn { \object_ncmember_adr:vnn { \_rawobjects_object_pxyvar:n { #1 } }{ #2 }{ #3 } }
\end{verbatim}
\begin{verbatim}
\cs_generate_variant:Nn \object_rcmember_adr:nnn { Vnn }
\end{verbatim}

Tests if the specified member constant exists.
\begin{verbatim}
\prg_new_conditional:Nnn \object_ncmember_if_exist:nnn {p, T, F, TF }
\end{verbatim}
\begin{verbatim}
\object_ncmember_if_exist_p:nnn \object_ncmember_if_exist:nnn {p, T, F, TF }
\end{verbatim}
\begin{verbatim}
\object_rcmember_if_exist:nnn \object_rcmember_if_exist_p:nnn \object_rcmember_if_exist:nnn {p, T, F, TF }
\end{verbatim}

(End definition for \object_member_set_eq:nnnN and \object_member_set_eq:nnnN. These functions are documented on page 5.)

(End definition for \object_ncmember_adr:nnn and \object_rcmember_adr:nnn. These functions are documented on page 5.)

(End definition for \object_ncmember_if_exist:nnn and \object_rcmember_if_exist:nnn. These functions are documented on page 5.)
\cs_if_exist:cTF
{
  \object_ncmember_adr:nnn { #1 }{ #2 }{ #3 }
}
{ \prg_return_true:
  { \prg_return_false:
    
  }
}
\prg_new_conditional:Nnn \object_rcmember_if_exist:nnn { p, T, F, TF }
{
  \cs_if_exist:cTF
    { \object_rcmember_adr:nnn { #1 }{ #2 }{ #3 }
      }
    { \prg_return_true:
      { \prg_return_false:
        
      }
    }

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

(End definition for \object_ncmember_if_exist:nnnTF and \object_rcmember_if_exist:nnnTF. These functions are documented on page 5.)

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

\cs_new:Nn \object_ncmember_use:nnn
  \cs_if_exist_use:cT { #3 _ use:c }
  { \object_ncmember_adr:nnn { #1 }{ #2 }{ #3 }
  }
\cs_new:Nn \object_rcmember_use:nnn
{ \cs_if_exist_use:cT { #3 _ use:c }
  { \object_rcmember_adr:nnn { #1 }{ #2 }{ #3 }
  }
\cs_generate_variant:Nnn \object_ncmember_use:nnn { Vnn }

18
\cs_generate_variant:Nn \object_rcremember_use:nnn { Vnn }

(End definition for \object_ncmember_use:nnn and \object_rcremember_use:nnn. These functions are documented on page 5.)

Create constants

\cs_new_protected:Nn \__rawobjects_const_create:nnnn
{ \use:c { \_const:cn } }
{ \object_ncmember_adr:nnn { #2 }{ #3 }{ #1 } }
{ #4 }

\cs_new_protected:Nn \object_newconst_tl:nnn
{ \__rawobjects_const_create:nnnn { tl }{ #1 }{ #2 }{ #3 } }

\cs_new_protected:Nn \object_newconst_str:nnn
{ \__rawobjects_const_create:nnnn { str }{ #1 }{ #2 }{ #3 } }

\cs_new_protected:Nn \object_newconst_int:nnn
{ \__rawobjects_const_create:nnnn { int }{ #1 }{ #2 }{ #3 } }

\cs_new_protected:Nn \object_newconst_clist:nnn
{ \__rawobjects_const_create:nnnn { clist }{ #1 }{ #2 }{ #3 } }

\cs_new_protected:Nn \object_newconst_dim:nnn
{ \__rawobjects_const_create:nnnn { dim }{ #1 }{ #2 }{ #3 } }

\cs_new_protected:Nn \object_newconst_skip:nnn
{ \__rawobjects_const_create:nnnn { skip }{ #1 }{ #2 }{ #3 } }

\cs_new_protected:Nn \object_newconst_fp:nnn
{ \__rawobjects_const_create:nnnn { fp }{ #1 }{ #2 }{ #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 }
\object_newconst_seq_from_clist:nnn

Creates a seq constant.

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

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

\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 7.)

\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
\__rawobjects_object_pxyvar:n { #1 }
\#2 \#3
\cs_generate_variant:Nn \object_rcmethod_adr:nnn { Vnn }

\cs_generate_variant:Nn \object_rcmethod_adr:nnn { Vnn }

20
Tests if the specified member constant exists.

\begin{verbatim}
\prg_new_conditional:Nnn \object_ncmethod_if_exist:n {p, T, F, TF }
{ \cs_if_exist:cTF
  { \object_ncmethod_adr:n { #1 }{ #2 }{ #3 }
    \prg_return_true:
  }
  { \prg_return_false:
  }
}
\prg_new_conditional:Nnn \object_rcmethod_if_exist:n {p, T, F, TF }
{ \cs_if_exist:cTF
  { \object_rcmethod_adr:n { #1 }{ #2 }{ #3 }
    \prg_return_true:
  }
  { \prg_return_false:
  }
}
\prg_generate_conditional_variant:Nnn \object_ncmethod_if_exist:n { Vnn }
\prg_generate_conditional_variant:Nnn \object_rcmethod_if_exist:n { Vnn }
\end{verbatim}

Creates a new method

\begin{verbatim}
\cs_new_protected:Nn \object_new_cmethod:n { \cs_new:cn
  { \object_ncmethod_adr:n { #1 }{ #2 }{ #3 }
    { #4 }
    \cs_generate_variant:Nn \object_new_cmethod:n { Vnn }
}
\end{verbatim}
\texttt{\object_ncmethod_call:nnn}\quad \text{Calls the specified method.}

\begin{verbatim}
\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{verbatim}

\texttt{\c_proxy_address_str}\quad \text{The address of the proxy object.}

\begin{verbatim}
\str_const:Nx \c_proxy_address_str
{ \object_address:nn { rawobjects }{ proxy } }
\end{verbatim}

\texttt{\object_if_proxy_p:n}\quad \text{Test if an object is a proxy.}

\begin{verbatim}
\prg_new_conditional:Nnn \object_if_proxy:n {p, T, F, TF}
\end{verbatim}
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 { \_rawobjects_object_pxyvar:n { #1 } }
\prg_return_true:
\prg_return_false:

\prg_generate_conditional_variant:Nnn \object_test_proxy:nN {p, T, F, TF}
\str_if_eq:cNTF { \_rawobjects_object_pxyvar:n { #1 } }
\prg_return_true:
\prg_return_false:

\prg_generate_conditional_variant:Nnn \object_test_proxy:nn { Ve }{ p, T, F, TF }
\msg_new:nnn { aa }{ mess }{ #1 }

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

(End definition for \object_test_proxy:nNTF and \object_test_proxy:nTF. These functions are documented on page 7.)

Creates an object from a proxy

\object_create:nnnNN
\object_create_set:NnnNN
\object_create_gset:NnnNNN
\msg_new:nnn { aa }{ mess }{ #1 }
\msg_new:n { rawobjects }{ notproxy }{ Fake ~ proxy }
{
   Object ~ #1 ~ is ~ not ~ a ~ proxy.
}

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

\cs_new_protected:Nn \_rawobjects_create_anon:nnnNN
{
\__rawobjects_force_proxy:n { #1 }
\str_const:cn { \__rawobjects_object_modvar:n { #2 } }{ #3 }
\str_const:cx { \__rawobjects_object_pxyvar:n { #2 } }{ #1 }
\str_const:cV { \__rawobjects_object_scovar:n { #2 } }#4
\str_const:cV { \__rawobjects_object_visvar:n { #2 } }#5
\seq_map_inline:cn
{ \object_member_adr:nnn { #1 }{ varlist }{ seq } }
{
   \object_new_member:nnv { #2 }{ ##1 }
   { \object_ncmember_adr:nnn { #1 }{ ##1 _ type }{ str } }
}

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

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

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

\cs_generate_variant:Nn \object_create:nnnNN { VnnNN }
\message{rawobjects (notproxy) \\
   Object ~ #1 ~ is ~ not ~ a ~ proxy.
}
Create an address and use it to instantiate an object

\begin{verbatim}
\cs_new:Nn \object_allocate_incr:NNnnNN { \object_create_set:NnnfNN #1 { #3 }{ #4 }
\__rawobjects_combine_aux:Vnf #2 { #3 }
\cs_to_str:N #1 }
\int_incr:N #2
}
\cs_new_protected:Nn \object_gallocate_incr:NNnnNN { \object_create_gset:NnnfNN #1 { #3 }{ #4 }
\__rawobjects_combine_aux:Vnf #2 { #3 }
\int_incr:N #2
}
\cs_generate_variant:Nn \object_allocate_incr:NNnnNN { NNVnNN }
\cs_generate_variant:Nn \object_gallocate_incr:NNnnNN { NNVnNN }
\cs_new_protected:Nn \object_allocate_gincr:NNnnNN { \object_create_set:NnnfNN #1 { #3 }{ #4 }
\__rawobjects_combine_aux:Vnf #2 { #3 }
\int_incr:N #2
}
\cs_new_protected:Nn \object_gallocate_gincr:NNnnNN { \object_create_gset:NnnfNN #1 { #3 }{ #4 }
\__rawobjects_combine_aux:Vnf #2 { #3 }
\int_incr:N #2
}
\end{verbatim}
__rawobjects_combine:Nn #2 { #3 }  
  
  \int_gincr:N #2

\cs_new_protected:Nn \object_gallocate_gincr:NNnnNN
{
  \object_create_gset:NNNN { #3 }{ #4 }
  \__rawobjects_combine:Nn #2 { #3 }
  \int_gincr:N #2
}

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

\proxy_create:nnN (End definition for \object_allocate_incr:NNnnNN and others. These functions are documented on page 8.)

\proxy_create_set:NnnN
\proxy_create_gset:NnnN

\cs_new_protected:Nn \proxy_create:nnN
{
  \object_create:VnnNN \c_proxy_address_str { #1 }{ #2 }
  \c_object_global_str #3
}

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

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

(End definition for \proxy_create:nnN, \proxy_create_set:NnnN, and \proxy_create_gset:NnnN. These functions are documented on page 8.)

\proxy_push_member:nnn

\cs_new_protected:Nn \proxy_push_member:nnn
{
  \__rawobjects_force_scope:n { #1 }
  \object_newconst_str:nnn { #1 }{ #2 _ type }{ #3 }

(26)
Copy an object to another one.

A simple forward list proxy
\rawobjects_fwl_inst:n { #1 }
\object_create:nnnNN
{
  \object_address:nn { rawobjects }{ fwl ! #1 }
}
{ #2 }{ #3 } #4 #5
}

(/package)