The \textit{lt3rawobjects} package

Paolo De Donato

Released 2022/07/11 Version 1.1.1

Contents

1 Introduction 1
2 To do 1
3 Objects and proxies 2
4 Constants 3
5 Library functions 3
  5.1 Base object functions 3
  5.2 Operating with member variables and constants 4
  5.3 Constant creation 5
  5.4 Proxy utilities and object creation 6
6 Examples 7
7 Templated proxies 8
8 Implementation 9

1 Introduction

First to all notice that \textit{lt3rawobjects} means “raw object(s)”, indeed \textit{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.

2 To do

- Introduce member functions in objects and member function specifications in proxies;
- Uniform declarations for templated proxies;
- Introduce 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. Also in lt3rawobjects objects are collections of variables, called member variables, which can be retrieved from a string representing that object. Such string is the address of the object and act like the address of a structure in C.

An address is composed of two parts, the module in which variables are created and an 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 \object_address function. Identifiers and module names should not contain numbers, # and _ characters in order to avoid conflicts with automatically generated addresses.

In C each object/structure has a type that tells the compiler how each object should be organized and instantiated in the memory. So if you need to create objects with the same structure you should first create a new struct entity and then create object with such type.

In lt3rawobjects objects are created from an existing object with a particular structure that holds all the needed informations to organize their variables. Such objects that can be used to instantiate new objects are called proxies and the proxy object used to instantiate an object is its generator. In order to create new objects with a specified proxy you can use the \object_create functions.

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

Once you’ve created you proxy object you should specify its member variables that will be created in each object initialized with such proxy. You can add a variable specification with the \proxy_push_member function. Once you’ve added all your variables specifications you can use your proxy to create objects. You should never modify a proxy once you’ve used it to create at least one object, since these modifications won’t be updated on already created objects, leading to hidden errors in subsequential code.

When you create a new variable specification with the \proxy_push_member you can notice the presence of (type) parameter. It represents the type of such variable and can be a standard type (like tl, str, int, seq, ...) or user defined types if the following functions are defined:

\(\langle\text{type}\rangle\text{\_new}:N\) and \c\ variant;
\(\langle\text{type}\rangle\text{\_set\_eq}:NN\) and \cN, \nc, \cc variants.

Every object, and so proxies too, is characterized by the following parameters:

- the module in which it has been created;
- the address of the proxy generator;
- a parameter saying if the object is local or global;
- a parameter saying if the object is public or private;
- zero or more member variables.
In a local/global/public/private object every member variable is declared local/global/pub-
lic/private. Address of a member variable can be obtained with the \texttt{object\_member\_adr} function, and you can instantiate new members that haven’t been specified in its
generator with the function \texttt{object\_new\_member}. members created in this way aren’t
described by generator proxy, so its type can’t be deduced and should be always specified
in functions like \texttt{object\_member\_adr} or \texttt{object\_member\_use}.

4 Constants

This feature is available only from version 1.1 of \texttt{lt3rawobjects}. There’re two different
kinds of constants you can define on a object:

1. near constants are constants defined directly inside the associated object;
2. remote constants are constants that are defined instead on the generator proxy and
so every object generated with that proxy can access the constant.

Currently it’s possible to define only public constants, if you need private constants use
member variables instead.

Notice that all near constants declared on a proxy are automatically remote constants
for every generated object, but remote constants for a proxy aren’t directly accessible by
generated objects.

You can retrieve the address of a near constant with the \texttt{object\_nconst\_adr} func-
tion and of a remote constant with \texttt{object\_rconst\_adr}.

5 Library functions

5.1 Base object functions

\texttt{\textbar{\textbackslash object\_address:nn}} \star \texttt{\textbar{\textbackslash object\_address:nn \{\textbar{\textbackslash module}\}\{\textbar{\textbackslash 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 \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. If you want to
execute its content you should use a new variant, for example \texttt{\langle\langle V, f or e\rangle\rangle} variants.

\texttt{\textbar{\textbackslash object\_address_set:NNn}} \star \texttt{\textbar{\textbackslash object\_address_set:NNn \{\textbar{\textbackslash str\ var}\}\{\textbar{\textbackslash module}\}\{\textbar{\textbackslash id}\}}

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

\texttt{\textbar{\textbackslash object\_get\_module:n}} \star \texttt{\textbar{\textbackslash object\_get\_module:n \{\textbar{\textbackslash address}\}}
\texttt{\textbar{\textbackslash object\_get\_proxy\_adr:n}} \star \texttt{\textbar{\textbackslash object\_get\_proxy\_adr:n \{\textbar{\textbackslash address}\}}

Get the object module and its generator.
5.2 Operating with member variables and constants

\texttt{\object_member_adr:nnn} \star \texttt{\object_member_adr:nn} \{(member name)\} \{(member type)\}
\texttt{\object_member_adr:nn} \star \texttt{\object_member_adr:Vn} \star
\texttt{\object_member_type:nn} \star \texttt{\object_member_type:Vn} \star
\texttt{\object_new_member:nnn} \star \texttt{\object_new_member:nn} \{(member name)\} \{(member type)\}
\texttt{\object_new_member:(Vnn|nnv)} \star
\texttt{\object_member_use:nnn} \star \texttt{\object_member_use:nn} \{(member name)\} \{(member type)\}
\texttt{\object_member_use:Vn} \star

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.

\texttt{\object_member_type:nn} \star \texttt{\object_member_type:nn} \{(member name)\}
\texttt{\object_member_type:Vn} \star

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

\texttt{\object_new_member:nnn} \star \texttt{\object_new_member:nn} \{(member name)\} \{(member type)\}
\texttt{\object_new_member:(Vnn|nnv)} \star

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.

\texttt{\object_member_use:nnn} \star \texttt{\object_member_use:nn} \{(member name)\} \{(member type)\}
\texttt{\object_member_use:Vn} \star

Uses the specified member variable.
Sets the value of specified member equal to the value of \langle variable \rangle.

\textit{From: 1.0}

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

\textit{From: 1.1}

\section*{5.3 Constant creation}

Unlike normal variables, constants in \LaTeX{}3 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).

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

\textit{From: 1.1}

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

\textit{From: 1.1}
\object_newconst_prop_from_keyval:nnn  \object_newconst_prop_from_keyval:nnn \{(address)\} \{(constant name)\}
{
  \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

5.4 Proxy utilities and object creation

\object_if_proxy_p:n \object_if_proxy_p:V
\object_if_proxy:nTF \object_if_proxy:VTF
Test if the specified object is a proxy object.
From: 1.0

\c_proxy_address_str
The address of the proxy object in the rawobjects module.
From: 1.0

\object_create:nnnNN \object_create:VnnNN
\object_create:nnnNN \langle proxy address \rangle \langle module \rangle \langle id \rangle \langle scope \rangle \langle visibility \rangle
Creates an object by using the proxy at \langle proxy address \rangle and the specified parameters.
From: 1.0

\c_object_local_str \c_object_global_str
Possible values for \langle scope \rangle parameter.
From: 1.0

\c_object_public_str \c_object_private_str
Possible values for \langle visibility \rangle parameter.
From: 1.0

\object_create_set:NnnnNN \object_create_set:NVnnNN
\object_create_set:NnnnNN \langle str var \rangle \langle proxy address \rangle \langle module \rangle \langle id \rangle \langle scope \rangle \langle visibility \rangle
Creates an object and sets its fully expanded address inside \langle str var \rangle.
From: 1.0

\object_allocate_incr:NNnnNN \object_allocate_incr:NNVnNN
\object_gallocate_incr:NNnnNN \object_gallocate_incr:NNVnNN
\object_allocate_gincr:NNnnNN \object_allocate_gincr:NNVnNN
\object_gallocate_gincr:NNnnNN \object_gallocate_gincr:NNVnNN
Build a new object address with module \langle module \rangle and an identifier generated from \langle proxy address \rangle and the integer contained inside \langle int var \rangle, then increments \langle int var \rangle. This is very useful when you need to create a lot of objects, each of them on a different address. the _incr version increases \langle int var \rangle locally whereas _gincr does it globally.
From: 1.1
\proxy_create:nnN \proxy_create_set:NNnN \proxy_create_gset:NNnN
\proxy_push_member:nnn \proxy_push_member:Vnn
\object_assign:nn \object_assign:(Vn|nV|VV)

Creates a global proxy object.

From: 1.0

Creates a global proxy object.

From: 1.0

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

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

From: 1.0

6 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:NNnVVNN \l_myobj_str \l_intc_int \l_myproxy_str
{ example } \c_object_local_str \c_object_public_str
\tl_set:cn
7 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 { linklist } { node - #1 } \}

where the #1 argument is simply a string representing the type of data held by our linked list (for example tl, str, 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 { linklist } { node - #1 }
\c_object_public_str
\proxy_push_member:nnn {\node_address:n { #1 } } { next } { str }
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 next member variable of type \texttt{str} that points to the next node, and a 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, ...

8 Implementation

\begin{verbatim}
\proxy_push_member:nnn {\node_address:n { #1 } }
  { data }{ #1 }
}\node_address:nn
\end{verbatim}

Get address of an object

\begin{verbatim}
\object_address:nn
\end{verbatim}

Saves the address of an object into a string variable
Tests if object exists.

\begin{Verbatim}
\texttt{\object_if_exist\_p:n}
\texttt{\object_if_exist\_nTF}
\end{Verbatim}

\begin{Verbatim}
\texttt{\object_if_exist:n\{ p, T, F, TF \}}
\end{Verbatim}

Retrieve the name, module and generating proxy of an object.

\begin{Verbatim}
\texttt{\object_get\_module:n}
\texttt{\object_get\_proxy\_adr:n}
\end{Verbatim}

Test the specified parameters.

\begin{Verbatim}
\texttt{\object_if_local\_p:n}
\texttt{\object_if_local\_nTF}
\texttt{\object_if_global\_p:n}
\texttt{\object_if_global\_nTF}
\texttt{\object_if_public\_p:n}
\texttt{\object_if_public\_nTF}
\texttt{\object_if_private\_p:n}
\texttt{\object_if_private\_nTF}
\end{Verbatim}

(End definition for \texttt{\object_if_exist\_nTF}. This function is documented on page 3.)

(End definition for \texttt{\object_get\_module:n} and \texttt{\object_get\_proxy\_adr:n}. These functions are documented on page 3.)
\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_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 { V }
{ p, T, F, TF }
\prg_generate_conditional_variant:Nnn \object_if_global:n { V }
{ p, T, F, TF }
\prg_generate_conditional_variant:Nnn \object_if_public:n { V }
{ p, T, F, TF }
\prg_generate_conditional_variant:Nnn \object_if_private:n { V }
{ p, T, F, TF }

(End definition for \object_if_local:nTF and others. These functions are documented on page 4.)
\object_member_adr:nnn
Get the address of a member variable

\object_member_adr:nn
\cs_new:Nn \__rawobjects_scope:n
{ \object_if_global:nTF { #1 }
{ g
}
{ \str_if_eq:cNTF { \__rawobjects_object_scovar:n { #1 } } \c__rawobjects_const_str
{ l
}
}
}
\cs_new:Nn \object_member_adr:nnn
{ \__rawobjects_scope:n { #1 }
\object_if_private:nTF { #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 { \__rawobjects_object_pxyvar:n { #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_member_use:vnn { \__rawobjects_object_pxyvar:n { #1 } }
\msg_new:nnnn \referencemodule\{ rawobjects \}{ scoperr \}{ Nonstandard ~ scope }
\begin{verbatim}
\{ Operation ~ not ~ permitted ~ on ~ object ~ #1 ~ 
~ since ~ it ~ wasn’t ~ declared ~ local ~ or ~ global \}
\end{verbatim}
\cs_new_protected:Nn \__rawobjects_force_scope:n
\begin{verbatim}
\bool_if:nF
\begin{verbatim}
\{ \object_if_local_p:n { #1 } || \object_if_global_p:n { #1 }
\}
\end{verbatim}
\begin{verbatim}
\msg_error:nnx \referencemodule\{ rawobjects \}{ scoperr \}{ #1 }
\end{verbatim}
\end{verbatim}
\cs_generate_variant:Nn \object_new_member:nnn { Vnn, nnv }
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ nn \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ nnv \}
\end{verbatim}
\object_new_member:nnn \referencemodule\{ nnn \}
creates a new member variable
\cs_new_protected:Nn \object_new_member:nnn
\begin{verbatim}
\{ \__rawobjects_force_scope:n \{ #1 \}
\cs_if_exist_use:cT \{ #3 _ new:c \}
\begin{verbatim}
\{ \object_member_adr:nnn \{ #1 \}\{ #2 \}\{ #3 \} \}
\end{verbatim}
\end{verbatim}
\cs_generate_variant:Nn \object_new_member:nnn { Vnn, nnv }
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ nn \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ nnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ nnn \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ nnnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ Vnn, nnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ nnnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ Vnn, nnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ nnnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ Vnn, nnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ nnnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ Vnn, nnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ nnnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ Vnn, nnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ nnnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ Vnn, nnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ nnnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ Vnn, nnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ nnnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ Vnn, nnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ nnnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ Vnn, nnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ nnnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ Vnn, nnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ nnnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ Vnn, nnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ nnnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ Vnn, nnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ nnnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ Vnn, nnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ nnnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ Vnn, nnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ nnnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ Vnn, nnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ nnnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ Vnn, nnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ nnnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ Vnn, nnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referencemodule\{ nnnv \}
\end{verbatim}
\begin{verbatim}
\object_new_member:nnn \referen...
\object_member_adr:vnn \{ \_rawobjects_object_pxyvar:n \{ #1 \} \}
{ #2 _ type ){ str }
}
\cs_generate_variant:Nn \object_member_use: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 4.)

\object_member_set_eq:nnnN \object_member_set_eq:nnN
Set the value of a variable to a member.
\cs_new_protected:Nn \object_member_set_eq:nnnN
{ \_rawobjects_force_scope:n \{ #1 \}
  \cs_if_exist_use:cT
  { \object_if_global:nT \{ #1 \}{ g } set _ eq:cN
    }
  { \object_member_adr:nnn \{ #1 \}{ #2 }{ #3 } } #4
}
\cs_generate_variant:Nn \object_member_set_eq:nnnN \{ VnnN, nnnc, Vnnc, nnvN \}
\cs_new_protected:Nn \object_member_set_eq:nnN
{ \object_member_set_eq:nnnN \{ #1 \}{ #2 }\{ #3 \} #4
}
\cs_generate_variant:Nn \object_member_set_eq:nnN \{ VnN, nnc, Vnc \}

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

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

\object_nconst_use:nnn \object_rconst_use:nnn

Uses a near/remote constant.

\cs_new:Nn \object_nconst_use:nnn
\cs_if_exist_use:cT { #3 _ use:c }
\{ \object_nconst_adr:nnn { #1 }{ #2 }{ #3 } \}
\}
\cs_new:Nn \object_rconst_use:nnn
\cs_if_exist_use:cT { #3 _ use:c }
\{ \object_rconst_adr:nnn { #1 }{ #2 }{ #3 } \}
\}
\cs_generate_variant:Nn \object_nconst_use:nnn { Vnn }
\cs_generate_variant:Nn \object_rconst_use:nnn { Vnn }

\end{verbatim}

Create constants

\cs_new_protected:Nn \__rawobjects_const_create:nnnn
\use:c { #1 _ const:cn }
\{ \object_nconst_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_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_int:nnn
\{ \__rawobjects_const_create:nnnn { int }{ #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 } \}
\end{verbatim}

(End definition for \object_nconst_use:nnn and \object_rconst_use:nnn. These functions are documented on page 5.)
\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 }

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

\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_nconst 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 5.)

\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_nconst 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 6.)

\c_proxy_address_str The address of the proxy object.
\str_const:Nx \c_proxy_address_str { \object_address:nn { rawobjects }{ proxy } }

(End definition for \c_proxy_address_str. This variable is documented on page 6.)

Source of proxy object
\str_const:cn { \__rawobjects_object_modvar:V \c_proxy_address_str }
\str_const:cV { \__rawobjects_object_pxyvar:V \c_proxy_address_str }
\str_const:cV { \__rawobjects_object_scovar:V \c_proxy_address_str }
\str_const:cV { \__rawobjects_object_visvar:V \c_proxy_address_str }
\c__rawobjects_const_str
\str_const:cV { \__rawobjects_object_visvar:V \c_proxy_address_str }
\c_object_public_str
\cs_generate_variant:Nn \seq_const_from_clist:Nn { cx }
\str_const:cn { \object_member_adr:Vnn \c_proxy_address_str { varlist }{ seq } }
\seq_const_from_clist:cn
\str_const:cn { \object_member_adr:Vnn \c_proxy_address_str { varlist_type }{ str } }
\seq_const:cn
\str_if_eq:cNTF { \__rawobjects_object_pxyvar:n { #1 } } \c_proxy_address_str
\prg_new_conditional:Nnn \object_if_proxy:n {p, T, F, TF}
\object_if_proxy:p:n
\object_if_proxy:nTF

Test if an object is a proxy.
\prg_new_conditional:Nnn \object_if_proxy:n {p, T, F, TF}
\object_if_proxy:nTF
\prg_return_true:
\prg_return_false:

(End definition for \object_if_proxy:nTF. This function is documented on page 6.)
\object_create:nnnNN
\object_create_set:NnnnNN
\object_create_gset:NnnnNN

Creates an object from a proxy

\msg_new:nnn { aa }{ mess }{ #1 }

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

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

\cs_new_protected:Nn \__rawobjects_create_anon:nnnNN
{ \__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_member_adr:nnn { #1 }{ ##1 _ type }{ str } }
  }
}

\cs_new_protected:Nn \object_create:nnnNN
{ \__rawobjects_create_anon:nnnNN { #1 }{ \object_address:nn { #2 }{ #3 } }{ #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 }
}
Create an address and use it to instantiate an object

\cs_new:Nn \__rawobjects_combine:nn
{ anon . #2 . #1 }
\cs_generate_variant:Nn \__rawobjects_combine:nn { Vn }
\cs_new_protected:Nn \object_allocate_incr:NNnnNN
{ \object_create_set:NNnnNN #1 { #3 }{ #4 }

\__rawobjects_combine:Vn #2 { #3 }
\int_incr:N #2 }
\cs_generate_variant:Nn \object_allocate_incr:NNnnNN { NNVnNN }
\cs_new_protected:Nn \object_gallocate_incr:NNnnNN
{ \object_create_gset:NNnnNN #1 { #3 }{ #4 }

\__rawobjects_combine:Vn #2 { #3 }
\int_incr:N #2 }
\cs_generate_variant:Nn \object_gallocate_incr:NNnnNN { NNVnNN }
\cs_new_protected:Nn \object_allocate_gincr:NNnnNN
{ \object_create_set:NNnnNN #1 { #3 }{ #4 }

\__rawobjects_combine:Vn #2 { #3 }
\int_incr:N #2 }
\cs_generate_variant:Nn \object_allocate_gincr:NNnnNN { NNVnNN }
\cs_new_protected:Nn \object_gallocate_gincr:NNnnNN
{ \object_create_gset:NNnnNN #1 { #3 }{ #4 }

\__rawobjects_combine:Vn #2 { #3 }
\int_incr:N #2 }
\cs_generate_variant:Nn \object_gallocate_gincr:NNnnNN { NNVnNN }

(End definition for \object_create:NNnnNN, \object_create_set:NNnnNN, and \object_create_gset:NNnnNN. These functions are documented on page 6.)
\int_gincr:N #2
\cs_new_protected:Nn \object_gallocate_gincr:NNnnNN
\{ \object_create_gset:NnnnNN #1 { #3 }{ #4 }
\{ \__rawobjects_combine:Vn #2 { #3 }
\} #5 #6
\int_gincr:N #2 \}
\cs_generate_variant:Nn \object_allocate_gincr:NNnnNN { NNVnNN }
\cs_generate_variant:Nn \object_gallocate_gincr:NNnnNN { NNVnNN }
\proxy_create:nnN \proxy_create_set:NnnN \proxy_create_gset:NnnN
\proxy_push_member:nnn

Creates a new proxy object.

\proxy_create:nnN
\{ \object_create:VnnNN \c_proxy_address_str { #1 }{ #2 }
\c_object_global_str #3 \}
\proxy_create_set:NnnN
\{ \object_create_set:NVnnNN #1 \c_proxy_address_str { #2 }{ #3 }
\c_object_global_str #4 \}
\proxy_create_gset:NnnN
\{ \object_create_gset:NVnnNN #1 \c_proxy_address_str { #2 }{ #3 }
\c_object_global_str #4 \}

\proxy_push_member:nnn
\{ \__rawobjects_force_scope:n { #1 }
\object_new_member:nnn { #1 }{ #2 _ type }{ str }
\str_set:cn
\{ \object_member_adr:nnn { #1 }{ #2 _ type }{ str } \}

(End definition for \object_allocate_incr:NNnnNN and others. These functions are documented on page 6.)

(End definition for \proxy_create:nnN, \proxy_create_set:NnnN, and \proxy_create_gset:NnnN. These functions are documented on page 7.)
\object_assign

Copy an object to another one.

\cs_new_protected:Nn \object_assign:nn
{ \seq_map_inline:cn
  \object_member_adr:vnn
  \__rawobjects_object_pxyvar:n { #1 }
  \object_if_exist:nF
  \object_address:nn { rawobjects }{ fwl ! #1 }
  \proxy_create:nnN { rawobjects }{ fwl ! #1 } \c_object_private_str
  \proxy_push_member
  \object_address:nn { rawobjects }{ fwl ! #1 }
  \object_member_adr:nn { #2 }
  \object_member_set_eq:nnc { #1 }{ ##1 }
}{ varlist ){ seq }
}{ #2 }
}
\cs_generate_variant:Nn \object_assign:nn { nV, Vn, VV }

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

A simple forward list proxy

\cs_new_protected:Nn \rawobjects_fwl_inst:n
{ \object_if_exist:nF
  \object_address:nn { rawobjects }{ fwl ! #1 }
  \proxy_create:nnN { rawobjects }{ fwl ! #1 } \c_object_private_str
  \proxy_push_member
  \object_address:nn { rawobjects }{ fwl ! #1 }
  \object_member_adr:nn { #2 }
  \object_member_set_eq:nnc { #1 }{ ##1 }
}{ varlist ){ seq }
}{ #2 }
}
\cs_generate_variant:Nn \rawobjects_fwl_newnode:nnnNN { Vnn }

(End definition for \rawobjects_fwl_inst:n. This function is documented on page 7.)
\{ \rawobjects_fwl_inst:n \{ \#1 \} \}
\object_create:nnnnN
\{
\object_address:nn \{ \rawobjects \}{ \fwl ! \#1 }\}
\{ \#2 \}{ \#3 } \#4 \#5
\}

\{/package\}