# Debug objects life time

**Thomas Gleixner**

**tglx@linutronix.de**

# Debug objects life time

by Thomas Gleixner

Copyright © 2008 Thomas Gleixner

This documentation is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.
This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
For more details see the file COPYING in the source distribution of Linux.

# Table of Contents

**1. Introduction** ..... **1**
**2. Howto use debugobjects** ..... **3**
**3. Debug functions** ..... **5**
3.1. Debug object function reference ..... 5
debug_object_init ..... 5
debug_object_init_on_stack ..... 5
debug_object_activate ..... 6
debug_object_deactivate ..... 7
debug_object_destroy ..... 8
debug_object_free ..... 8
debug_object_active_state ..... 9
3.2. debug_object_init ..... 10
3.3. debug_object_init_on_stack ..... 11
3.4. debug_object_activate ..... 11
3.5. debug_object_deactivate ..... 11
3.6. debug_object_destroy ..... 12
3.7. debug_object_free ..... 12
**4. Fixup functions** ..... **13**
4.1. Debug object type description structure ..... 13
struct debug_obj ..... 13
struct debug_obj_descr ..... 14
4.2. fixup_init ..... 15
4.3. fixup_activate ..... 15
4.4. fixup_destroy ..... 16
4.5. fixup_free ..... 16
**5. Known Bugs And Assumptions** ..... **19**

# Chapter 1. Introduction

debugobjects is a generic infrastructure to track the life time of kernel objects and validate the operations on those.

debugobjects is useful to check for the following error patterns:

- Activation of uninitialized objects
- Initialization of active objects
- Usage of freed/destroyed objects

debugobjects is not changing the data structure of the real object so it can be compiled in with a minimal runtime impact and enabled on demand with a kernel command line option.

# Chapter 2. Howto use debugobjects

A kernel subsystem needs to provide a data structure which describes the object type and add calls into the debug code at appropriate places. The data structure to describe the object type needs at minimum the name of the object type. Optional functions can and should be provided to fixup detected problems so the kernel can continue to work and the debug information can be retrieved from a live system instead of hard core debugging with serial consoles and stack trace transcripts from the monitor.

The debug calls provided by debugobjects are:

- debug_object_init
- debug_object_init_on_stack
- debug_object_activate
- debug_object_deactivate
- debug_object_destroy
- debug_object_free

Each of these functions takes the address of the real object and a pointer to the object type specific debug description structure.

Each detected error is reported in the statistics and a limited number of errors are printk'ed including a full stack trace.

The statistics are available via /sys/kernel/debug/debug_objects/stats. They provide information about the number of warnings and the number of successful fixups along with information about the usage of the internal tracking objects and the state of the internal tracking objects pool.

# Chapter 3. Debug functions

## 3.1. Debug object function reference

## debug_object_init

LINUX

Kernel Hackers ManualSeptember 2026

### Name

`debug_object_init` — debug checks when an object is initialized

### Synopsis

```
void debug_object_init (void * addr, struct debug_obj_descr *
descr);
```

### Arguments

*addr*

address of the object

*descr*

pointer to an object specific debug description structure

# debug_object_init_on_stack

**LINUX**

Kernel Hackers ManualSeptember 2026

## Name

`debug_object_init_on_stack` — debug checks when an object on stack is initialized

## Synopsis

```
void debug_object_init_on_stack (void * addr, struct
debug_obj_descr * descr);
```

## Arguments

*addr*

address of the object

*descr*

pointer to an object specific debug description structure

# debug_object_activate

**LINUX**

Kernel Hackers ManualSeptember 2026

## Name

`debug_object_activate` — debug checks when an object is activated

## Synopsis

```
void debug_object_activate (void * addr, struct
debug_obj_descr * descr);
```

## Arguments

*addr*

address of the object

*descr*

pointer to an object specific debug description structure

# debug_object_deactivate

## LINUX

Kernel Hackers ManualSeptember 2026

## Name

`debug_object_deactivate` — debug checks when an object is deactivated

## Synopsis

```
void debug_object_deactivate (void * addr, struct
debug_obj_descr * descr);
```

## Arguments

*addr*

address of the object

*descr*

pointer to an object specific debug description structure

# debug_object_destroy

## LINUX

Kernel Hackers ManualSeptember 2026

## Name

`debug_object_destroy` — debug checks when an object is destroyed

## Synopsis

```
void debug_object_destroy (void * addr, struct debug_obj_descr
* descr);
```

## Arguments

*addr*

address of the object

*descr*

pointer to an object specific debug description structure

# debug_object_free

LINUX

Kernel Hackers ManualSeptember 2026

## Name

`debug_object_free` — debug checks when an object is freed

## Synopsis

```
void debug_object_free (void * addr, struct debug_obj_descr *
descr);
```

## Arguments

*addr*

address of the object

*descr*

pointer to an object specific debug description structure

# debug_object_active_state

LINUX

Kernel Hackers ManualSeptember 2026

## Name

`debug_object_active_state` — debug checks object usage state machine

## Synopsis

```
void debug_object_active_state (void * addr, struct
debug_obj_descr * descr, unsigned int expect, unsigned int
next);
```

## Arguments

*addr*

address of the object

*descr*

pointer to an object specific debug description structure

*expect*

expected state

*next*

state to move to if expected state is found

# 3.2. debug_object_init

This function is called whenever the initialization function of a real object is called.

When the real object is already tracked by debugobjects it is checked, whether the object can be initialized. Initializing is not allowed for active and destroyed objects. When debugobjects detects an error, then it calls the fixup_init function of the object type description structure if provided by the caller. The fixup function can correct the problem before the real initialization of the object happens. E.g. it can deactivate an active object in order to prevent damage to the subsystem.

When the real object is not yet tracked by debugobjects, debugobjects allocates a tracker object for the real object and sets the tracker object state to ODEBUG_STATE_INIT. It verifies that the object is not on the callers stack. If it is on the callers stack then a limited number of warnings including a full stack trace is

printk'ed. The calling code must use debug_object_init_on_stack() and remove the object before leaving the function which allocated it. See next section.

## 3.3. debug_object_init_on_stack

This function is called whenever the initialization function of a real object which resides on the stack is called.

When the real object is already tracked by debugobjects it is checked, whether the object can be initialized. Initializing is not allowed for active and destroyed objects. When debugobjects detects an error, then it calls the fixup_init function of the object type description structure if provided by the caller. The fixup function can correct the problem before the real initialization of the object happens. E.g. it can deactivate an active object in order to prevent damage to the subsystem.

When the real object is not yet tracked by debugobjects debugobjects allocates a tracker object for the real object and sets the tracker object state to ODEBUG_STATE_INIT. It verifies that the object is on the callers stack.

An object which is on the stack must be removed from the tracker by calling debug_object_free() before the function which allocates the object returns. Otherwise we keep track of stale objects.

## 3.4. debug_object_activate

This function is called whenever the activation function of a real object is called.

When the real object is already tracked by debugobjects it is checked, whether the object can be activated. Activating is not allowed for active and destroyed objects. When debugobjects detects an error, then it calls the fixup_activate function of the object type description structure if provided by the caller. The fixup function can correct the problem before the real activation of the object happens. E.g. it can deactivate an active object in order to prevent damage to the subsystem.

When the real object is not yet tracked by debugobjects then the fixup_activate function is called if available. This is necessary to allow the legitimate activation of statically allocated and initialized objects. The fixup function checks whether the object is valid and calls the debug_objects_init() function to initialize the tracking of this object.

When the activation is legitimate, then the state of the associated tracker object is set to ODEBUG_STATE_ACTIVE.

## 3.5. debug_object_deactivate

This function is called whenever the deactivation function of a real object is called.

When the real object is tracked by debugobjects it is checked, whether the object can be deactivated. Deactivating is not allowed for untracked or destroyed objects.

When the deactivation is legitimate, then the state of the associated tracker object is set to ODEBUG_STATE_INACTIVE.

## 3.6. debug_object_destroy

This function is called to mark an object destroyed. This is useful to prevent the usage of invalid objects, which are still available in memory: either statically allocated objects or objects which are freed later.

When the real object is tracked by debugobjects it is checked, whether the object can be destroyed. Destruction is not allowed for active and destroyed objects. When debugobjects detects an error, then it calls the fixup_destroy function of the object type description structure if provided by the caller. The fixup function can correct the problem before the real destruction of the object happens. E.g. it can deactivate an active object in order to prevent damage to the subsystem.

When the destruction is legitimate, then the state of the associated tracker object is set to ODEBUG_STATE_DESTROYED.

## 3.7. debug_object_free

This function is called before an object is freed.

When the real object is tracked by debugobjects it is checked, whether the object can be freed. Free is not allowed for active objects. When debugobjects detects an error, then it calls the fixup_free function of the object type description structure if provided by the caller. The fixup function can correct the problem before the real free of the object happens. E.g. it can deactivate an active object in order to prevent damage to the subsystem.

Note that debug_object_free removes the object from the tracker. Later usage of the object is detected by the other debug checks.

# Chapter 4. Fixup functions

## 4.1. Debug object type description structure

## struct debug_obj

LINUX

Kernel Hackers ManualSeptember 2026

### Name

`struct debug_obj` — representaion of an tracked object

### Synopsis

```
struct debug_obj {
  struct hlist_node node;
  enum debug_obj_state state;
  unsigned int astate;
  void * object;
  struct debug_obj_descr * descr;
};
```

### Members

node

hlist node to link the object into the tracker list

state

tracked object state

astate

current active state

object

pointer to the real object

descr

pointer to an object type specific debug description structure

# struct debug_obj_descr

LINUX

Kernel Hackers ManualSeptember 2026

## Name

`struct debug_obj_descr` — object type specific debug description structure

## Synopsis

```
struct debug_obj_descr {
  const char * name;
  void *(* debug_hint) (void *addr);
  int (* fixup_init) (void *addr, enum debug_obj_state state);
  int (* fixup_activate) (void *addr, enum debug_obj_state state);
  int (* fixup_destroy) (void *addr, enum debug_obj_state state);
  int (* fixup_free) (void *addr, enum debug_obj_state state);
};
```

## Members

name

name of the object typee

debug_hint

function returning address, which have associated kernel symbol, to allow identify the object

fixup_init

fixup function, which is called when the init check fails

fixup_activate

fixup function, which is called when the activate check fails

fixup_destroy

fixup function, which is called when the destroy check fails

fixup_free

fixup function, which is called when the free check fails

## 4.2. fixup_init

This function is called from the debug code whenever a problem in debug_object_init is detected. The function takes the address of the object and the state which is currently recorded in the tracker.

Called from debug_object_init when the object state is:

- ODEBUG_STATE_ACTIVE

The function returns 1 when the fixup was successful, otherwise 0. The return value is used to update the statistics.

Note, that the function needs to call the debug_object_init() function again, after the damage has been repaired in order to keep the state consistent.

## 4.3. fixup_activate

This function is called from the debug code whenever a problem in debug_object_activate is detected.

Called from debug_object_activate when the object state is:

- ODEBUG_STATE_NOTAVAILABLE
- ODEBUG_STATE_ACTIVE

The function returns 1 when the fixup was successful, otherwise 0. The return value is used to update the statistics.

Note that the function needs to call the debug_object_activate() function again after the damage has been repaired in order to keep the state consistent.

The activation of statically initialized objects is a special case. When debug_object_activate() has no tracked object for this object address then fixup_activate() is called with object state ODEBUG_STATE_NOTAVAILABLE. The fixup function needs to check whether this is a legitimate case of a statically initialized object or not. In case it is it calls debug_object_init() and debug_object_activate() to make the object known to the tracker and marked active. In this case the function should return 0 because this is not a real fixup.

## 4.4. fixup_destroy

This function is called from the debug code whenever a problem in debug_object_destroy is detected.

Called from debug_object_destroy when the object state is:

- ODEBUG_STATE_ACTIVE

The function returns 1 when the fixup was successful, otherwise 0. The return value is used to update the statistics.

## 4.5. fixup_free

This function is called from the debug code whenever a problem in debug_object_free is detected. Further it can be called from the debug checks in kfree/vfree, when an active object is detected from the debug_check_no_obj_freed() sanity checks.

Called from debug_object_free() or debug_check_no_obj_freed() when the object state is:

- ODEBUG_STATE_ACTIVE

The function returns 1 when the fixup was successful, otherwise 0. The return value is used to update the statistics.

# Chapter 5. Known Bugs And Assumptions

None (knock on wood).