|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [Interface Overview](http://docs.google.com/gcinterface.html) | [Tutorial Slides](http://www.hboehm.info/gc/04tutorial.pdf) | [FAQ](http://www.hboehm.info/gc/faq.html) | [Example](http://docs.google.com/simple_example.html) | [Download](http://www.hboehm.info/gc/gc_source/) | [License](http://www.hboehm.info/gc/license.txt) |

# A garbage collector for C and C++

* [Platforms](#_gjdgxs)
* [Scalable multiprocessor versions](#_30j0zll)
* [Some collector details](#_1fob9te)
* [Further reading](#_3znysh7)
* [Current users](#_2et92p0)
* [Local Links for this collector](#_tyjcwt)
* [Local Background Links](#_3dy6vkm)
* [Contacts and Mailing List](#_1t3h5sf)

[ This is an updated version of the page formerly at www.hpl.hp.com/personal/Hans\_Boehm/gc/, before that at http://reality.sgi.com/boehm/gc.html and before that at <ftp://ftp.parc.xerox.com/pub/gc/gc.html>. ]

The [Boehm](http://www.hboehm.info)-[Demers](http://www.cs.cornell.edu/annual_report/00-01/bios.htm#demers)-[Weiser](http://www-sul.stanford.edu/weiser/) conservative garbage collector can be used as a garbage collecting replacement for C malloc or C++ new. It allows you to allocate memory basically as you normally would, without explicitly deallocating memory that is no longer useful. The collector automatically recycles memory when it determines that it can no longer be otherwise accessed. A simple example of such a use is given [here](http://docs.google.com/simple_example.html).

The collector is also used by a number of programming language implementations that either use C as intermediate code, want to facilitate easier interoperation with C libraries, or just prefer the simple collector interface. For a more detailed description of the interface, see [here](http://docs.google.com/gcinterface.html).

Alternatively, the garbage collector may be used as a [leak detector](http://docs.google.com/leak.html) for C or C++ programs, though that is not its primary goal.

Typically several versions will be available. Usually you should first try to use [gc\_source/gc.tar.gz](http://www.hboehm.info/gc/gc_source/gc.tar.gz), which is normally an older, more stable version.

If that fails, try the latest explicitly numbered version in  [gc\_source/](http://www.hboehm.info/gc/gc_source/). Later versions may contain additional features, platform support, or bug fixes, but are likely to be less well tested.

A slightly older version of the garbage collector is now also included as part of the [GNU compiler](http://gcc.gnu.org/) distribution. The source code for that version is available for browsing [here](http://gcc.gnu.org/cgi-bin/cvsweb.cgi/gcc/boehm-gc/).

The arguments for and against conservative garbage collection in C and C++ are briefly discussed in [issues.html](http://www.hboehm.info/gc/issues.html). The beginnings of a frequently-asked-questions list are [here](http://www.hboehm.info/gc/faq.html).

The garbage collector code is copyrighted by [Hans-J. Boehm](http://www.hboehm.info), Alan J. Demers, [Xerox Corporation](http://www.xerox.com/), [Silicon Graphics](http://www.sgi.com/), and [Hewlett-Packard Company](http://www.hp.com/). It may be used and copied without payment of a fee under minimal restrictions. See the README file in the distribution or the [license](http://www.hboehm.info/gc/license.txt) for more details. **IT IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY EXPRESSED OR IMPLIED. ANY USE IS AT YOUR OWN RISK**.

Empirically, this collector works with most unmodified C programs, simply by replacing malloc with GC\_malloc calls, replacing realloc with GC\_realloc calls, and removing free calls. Exceptions are discussed in [issues.html](http://www.hboehm.info/gc/issues.html).

## Platforms

The collector is not completely portable, but the distribution includes ports to most standard PC and UNIX/Linux platforms. The collector should work on Linux, \*BSD, recent Windows versions, MacOS X, HP/UX, Solaris, Tru64, Irix and a few other operating systems. Some ports are more polished than others.

Irix pthreads, Linux threads, Win32 threads, Solaris threads (pthreads only), HP/UX 11 pthreads, Tru64 pthreads, and MacOS X threads are supported in recent versions.

### Separately distributed ports

For MacOS 9/Classic use, Patrick Beard's latest port is available from <http://homepage.mac.com/pcbeard/gc/>. (Unfortunately, that's now quite dated. I'm not in a position to test under MacOS. Although I try to incorporate changes, it is impossible for me to update the project file.)

Precompiled versions of the collector for NetBSD are available [here](ftp://ftp.netbsd.org/pub/pkgsrc/current/pkgsrc/devel/boehm-gc/README.html).

[Debian Linux](http://www.debian.org/) includes prepackaged versions of the collector.

## Scalable multiprocessor versions

Kenjiro Taura, Toshio Endo, and Akinori Yonezawa have made available a [parallel collector](http://web.yl.is.s.u-tokyo.ac.jp/gc/) based on this one. Their collector takes advantage of multiple processors during a collection. Starting with collector version 6.0alpha1 we also do this, though with more modest processor scalability goals. Our approach is discussed briefly in [scale.html](http://docs.google.com/scale.html).

## Some Collector Details

The collector uses a [mark-sweep](http://www.hboehm.info/gc/complexity.html) algorithm. It provides incremental and generational collection under operating systems which provide the right kind of virtual memory support. (Currently this includes SunOS[45], IRIX, OSF/1, Linux, and Windows, with varying restrictions.) It allows [*finalization*](http://docs.google.com/finalization.html) code to be invoked when an object is collected. It can take advantage of type information to locate pointers if such information is provided, but it is usually used without such information. See the README and gc.h files in the distribution for more details.

For an overview of the implementation, see [here](http://docs.google.com/gcdescr.html).

The garbage collector distribution includes a C string ([*cord*](http://docs.google.com/include/cord.h)) package that provides for fast concatenation and substring operations on long strings. A simple curses- and win32-based editor that represents the entire file as a cord is included as a sample application.

Performance of the nonincremental collector is typically competitive with malloc/free implementations. Both space and time overhead are likely to be only slightly higher for programs written for malloc/free (see Detlefs, Dosser and Zorn's [Memory Allocation Costs in Large C and C++ Programs](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.30.3073&rep=rep1&type=ps).) For programs allocating primarily very small objects, the collector may be faster; for programs allocating primarily large objects it will be slower. If the collector is used in a multi-threaded environment and configured for thread-local allocation, it may in some cases significantly outperform malloc/free allocation in time.

We also expect that in many cases any additional overhead will be more than compensated for by decreased copying etc. if programs are written and tuned for garbage collection.

# Further Reading:

**The beginnings of a frequently asked questions list for this collector are** [**here**](http://www.hboehm.info/gc/faq.html).

**The following provide information on garbage collection in general**:

Paul Wilson's [garbage collection ftp archive](ftp://ftp.cs.utexas.edu/pub/garbage) and [GC survey](ftp://ftp.cs.utexas.edu/pub/garbage/gcsurvey.ps).

The Ravenbrook  [Memory Management Reference](http://www.memorymanagement.org/).

David Chase's [GC FAQ](http://www.iecc.com/gclist/GC-faq.html).

Richard Jones'  [Garbage Collection Page](https://www.cs.kent.ac.uk/people/staff/rej/gc.html) and  [his book](http://www.cs.kent.ac.uk/people/staff/rej/gcbook/gcbook.html).

**The following papers describe the collector algorithms we use and the underlying design decisions at a higher level.**

(Some of the lower level details can be found [here](http://docs.google.com/gcdescr.html).)

The first one is not available electronically due to copyright considerations. Most of the others are subject to ACM copyright.

Boehm, H., "Dynamic Memory Allocation and Garbage Collection", *Computers in Physics 9*, 3, May/June 1995, pp. 297-303. This is directed at an otherwise sophisticated audience unfamiliar with memory allocation issues. The algorithmic details differ from those in the implementation. There is a related letter to the editor and a minor correction in the next issue.

Boehm, H., and [M. Weiser](http://www.ubiq.com/hypertext/weiser/weiser.html), ["Garbage Collection in an Uncooperative Environment"](http://www.hboehm.info/spe_gc_paper/), *Software Practice & Experience*, September 1988, pp. 807-820.

Boehm, H., A. Demers, and S. Shenker, ["Mostly Parallel Garbage Collection"](http://www.hboehm.info/gc/papers/pldi91.ps.Z), Proceedings of the ACM SIGPLAN '91 Conference on Programming Language Design and Implementation, *SIGPLAN Notices 26*, 6 (June 1991), pp. 157-164.

Boehm, H., ["Space Efficient Conservative Garbage Collection"](http://www.hboehm.info/gc/papers/pldi93.ps.Z), Proceedings of the ACM SIGPLAN '93 Conference on Programming Language Design and Implementation, *SIGPLAN Notices 28*, 6 (June 1993), pp. 197-206.

Boehm, H., "Reducing Garbage Collector Cache Misses",  *Proceedings of the 2000 International Symposium on Memory Management* .  [Official version.](http://portal.acm.org/citation.cfm?doid=362422.362438)  [Technical report version.](http://www.hpl.hp.com/techreports/2000/HPL-2000-99.html) Describes the prefetch strategy incorporated into the collector for some platforms. Explains why the sweep phase of a "mark-sweep" collector should not really be a distinct phase.

M. Serrano, H. Boehm, "Understanding Memory Allocation of Scheme Programs", *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming*, 2000, Montreal, Canada, pp. 245-256.  [Official version.](http://dl.acm.org/citation.cfm?id=351264)  [Earlier Technical Report version.](http://www.hpl.hp.com/techreports/2000/HPL-2000-62.html) Includes some discussion of the collector debugging facilities for identifying causes of memory retention.

Boehm, H., "Fast Multiprocessor Memory Allocation and Garbage Collection",  [HP Labs Technical Report HPL 2000-165](http://www.hpl.hp.com/techreports/2000/HPL-2000-165.html). Discusses the parallel collection algorithms, and presents some performance results.

Boehm, H., "Bounding Space Usage of Conservative Garbage Collectors", *Proceedings of the 2002 ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Jan. 2002, pp. 93-100.  [Official version.](http://portal.acm.org/citation.cfm?doid=503272.503282)  [Technical report version.](http://www.hpl.hp.com/techreports/2001/HPL-2001-251.html) Includes a discussion of a collector facility to much more reliably test for the potential of unbounded heap growth.

**The following papers discuss language and compiler restrictions necessary to guaranteed safety of conservative garbage collection.**

We thank John Levine and JCLT for allowing us to make the second paper available electronically, and providing PostScript for the final version.

Boehm, H., ["Simple Garbage-Collector-Safety"](http://www.hboehm.info/gc/papers/pldi96.ps.gz), Proceedings of the ACM SIGPLAN '96 Conference on Programming Language Design and Implementation.

Boehm, H., and D. Chase, ["A Proposal for Garbage-Collector-Safe C Compilation"](http://www.hboehm.info/gc/papers/boecha.ps.gz), *Journal of C Language Translation 4*, 2 (Decemeber 1992), pp. 126-141.

**Other related information:**

The Detlefs, Dosser and Zorn's [Memory Allocation Costs in Large C and C++ Programs](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.30.3073&rep=rep1&type=ps). This is a performance comparison of the Boehm-Demers-Weiser collector to malloc/free, using programs written for malloc/free.

Joel Bartlett's [mostly copying conservative garbage collector for C++](ftp://gatekeeper.dec.com/pub/compaq/WRL/research-reports/WRL-TN-12.ps).

John Ellis and David Detlef's [Safe Efficient Garbage Collection for C++](ftp://ftp.parc.xerox.com/pub/ellis/gc/gc.ps) proposal.

Henry Baker's [paper collection](http://home.pipeline.com/%7Ehbaker1/).

Slides for Hans Boehm's [Allocation and GC Myths](http://www.hboehm.info/gc/myths.ps) talk.

# Current users:

Known current users of some variant of this collector include:

The runtime system for [GCJ](http://gcc.gnu.org/java), the static GNU java compiler.

[W3m](http://w3m.sourceforge.net/), a text-based web browser.

Some versions of the Xerox DocuPrint printer software.

The [Mozilla](http://www.mozilla.org/) project, as leak detector.

The [Mono](http://www.mono-project.com) project, an open source implementation of the .NET development framework.

The [DotGNU Portable.NET project](http://www.gnu.org/projects/dotgnu/), another open source .NET implementation.

The [Irssi IRC client](http://irssi.org/).

[The Berkeley Titanium project](http://titanium.cs.berkeley.edu/).

[The NAGWare f90 Fortran 90 compiler](http://www.nag.co.uk/nagware/NP/NP_desc.asp).

Elwood Corporation's Eclipse Common Lisp system, C library, and translator.

The [Bigloo Scheme](http://www-sop.inria.fr/mimosa/fp/Bigloo/) and [Camloo ML compilers](https://github.com/samoht/camloo) written by Manuel Serrano and others.

Brent Benson's [libscheme](http://www.cs.indiana.edu/scheme-repository/libscheme-vhll/final.html).

The MzScheme scheme implementation.

The [University of Washington Cecil Implementation](http://www.cs.washington.edu/research/projects/cecil/www/cecil-home.html).

[The Berkeley Sather implementation](http://www1.icsi.berkeley.edu/~sather/).

[The Berkeley Harmonia Project](http://www.cs.berkeley.edu/~harmonia/harmonia/index.html).

The [Toba](http://www.cs.arizona.edu/projects/sumatra/toba/) Java Virtual Machine to C translator.

The [Gwydion Dylan compiler](http://www.gwydiondylan.org/).

The  [GNU Objective C runtime](http://gcc.gnu.org/onlinedocs/gcc/Objective-C.html).

[Macaulay 2](http://www.math.uiuc.edu/Macaulay2), a system to support research in algebraic geometry and commutative algebra.

The [Vesta](http://www.vestasys.org/) configuration management system.

[Visual Prolog 6](http://www.visual-prolog.com/vip6).

[Asymptote LaTeX-compatible vector graphics language.](http://asymptote.sf.net/)

# More collector information at this site

[A simple illustration of how to build and use the collector](http://docs.google.com/simple_example.html).

[Description of alternate interfaces to the garbage collector.](http://docs.google.com/gcinterface.html)

[Slides from an ISMM 2004 tutorial about the GC](http://www.hboehm.info/gc/04tutorial.pdf).

[A FAQ (frequently asked questions) list](http://www.hboehm.info/gc/faq.html).

[How to use the garbage collector as a leak detector.](http://docs.google.com/leak.html)

[Some hints on debugging garbage collected applications.](http://docs.google.com/debugging.html)

[An overview of the implementation of the garbage collector.](http://docs.google.com/gcdescr.html)

[The data structure used for fast pointer lookups.](http://docs.google.com/tree.html)

[Scalability of the collector to multiprocessors.](http://docs.google.com/scale.html)

[Directory containing garbage collector source](http://www.hboehm.info/gc/gc_source/).

# More background information at this site

[An attempt to establish a bound on space usage of conservative garbage collectors](http://www.hboehm.info/gc/bounds.html).

[Mark-sweep versus copying garbage collectors and their complexity](http://www.hboehm.info/gc/complexity.html).

[Pros and cons of conservative garbage collectors, in comparison to other collectors](http://www.hboehm.info/gc/conservative.html).

[Issues related to garbage collection vs. manual memory management in C/C++](http://www.hboehm.info/gc/issues.html).

[An example of a case in which garbage collection results in a much faster implementation as a result of reduced synchronization](http://www.hboehm.info/gc/example.html).

[Slide set discussing performance of nonmoving garbage collectors](http://www.hboehm.info/gc/nonmoving/).

[Slide set discussing *Destructors, Finalizers, and Synchronization* (POPL 2003)](http://www.hboehm.info/popl03/web/).

[Paper corresponding to above slide set](http://portal.acm.org/citation.cfm?doid=604131.604153) ( [Technical Report version](http://www.hpl.hp.com/techreports/2002/HPL-2002-335.html)).

[A Java/Scheme/C/C++ garbage collection benchmark](http://www.hboehm.info/gc/gc_bench/).

[Slides for talk on memory allocation myths](http://www.hboehm.info/gc/myths.ps).

[Slides for OOPSLA 98 garbage collection talk](http://www.hboehm.info/gc/gctalk.ps).

[Related papers](http://www.hboehm.info/gc/papers/).

# Contacts and Mailing List

We have recently set up two mailing list for collector announcements and discussions:

* [bdwgc-announce@lists.opendylan.org](mailto:bdwgc-announce@lists.opendylan.org) is used for announcements of new versions. Postings are restricted. We expect this to always remain a very low volume list.
* [bdwgc@lists.opendylan.org](mailto:bdwgc@lists.opendylan.org) is used for discussions, bug reports, and the like. Subscribers may post. On-topic posts by nonsubscribers will usually also be accepted, but it may take some time to review them.

To subscribe to these lists, please visit [lists.opendylan.org/mailman/listinfo/bdwgc-announce](https://lists.opendylan.org/mailman/listinfo/bdwgc-announce) and [lists.opendylan.org/mailman/listinfo/bdwgc](https://lists.opendylan.org/mailman/listinfo/bdwgc), respectively.

The archives for these lists appear [here](https://lists.opendylan.org/pipermail/bdwgc-announce/) and [here](https://lists.opendylan.org/pipermail/bdwgc/), respectively. The gc list archive may also be read at [gmane.org](http://dir.gmane.org/gmane.comp.programming.garbage-collection.boehmgc).

Some prior discussion of the collector has taken place on the gcc java mailing list, whose archives appear [here](http://gcc.gnu.org/ml/java/), and also on [gclist@iecc.com](http://lists.tunes.org/mailman/listinfo/gclist).

Comments and bug reports may also be sent to ([boehm@acm.org](mailto:boehm@acm.org)), but the gc mailing list is usually preferred.