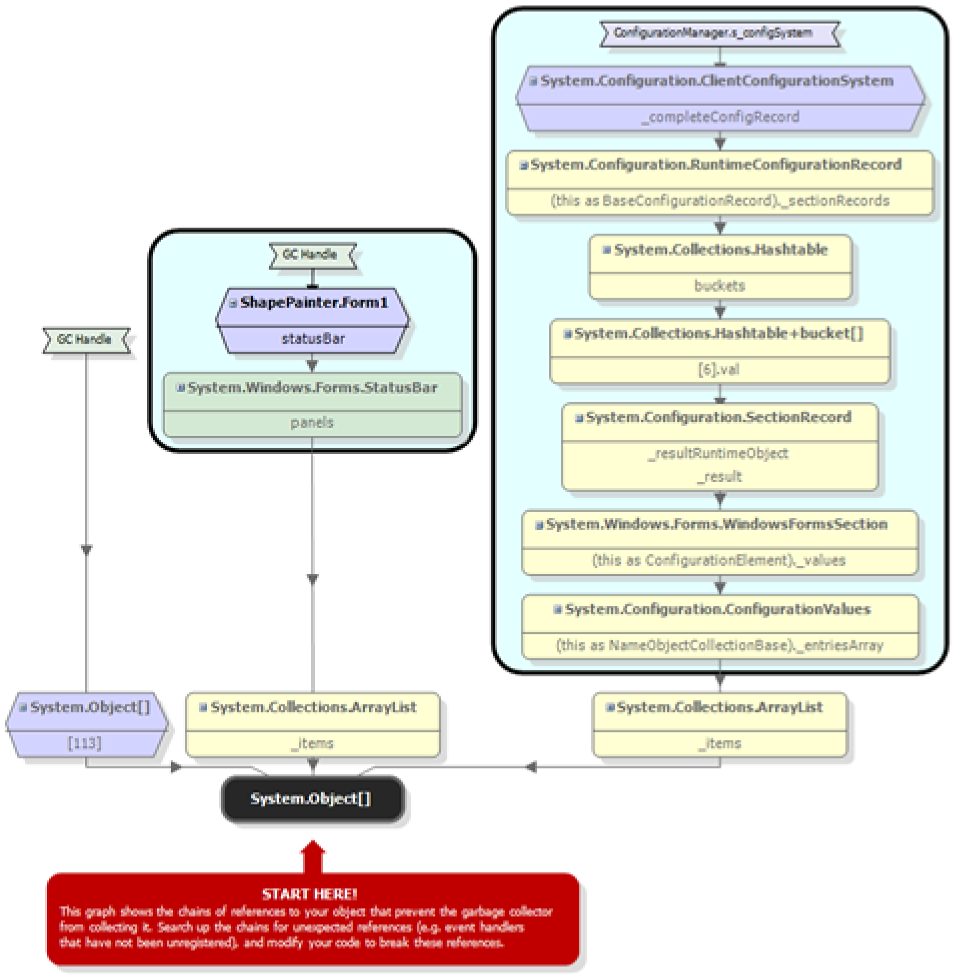
The instance retention graph

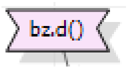
The **instance retention graph** shows chains of references between GC roots and your selected object.

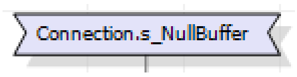
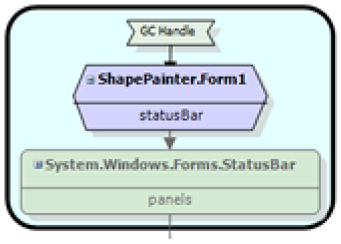
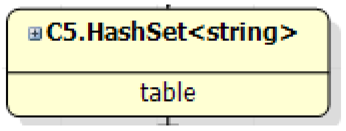
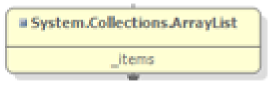
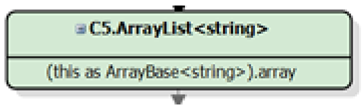
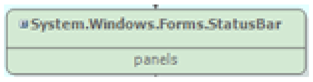
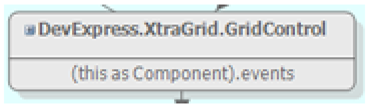
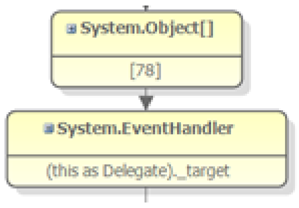
Start at your selected object and follow the chains of references up towards the GC roots, to identify references that are preventing the garbage collector from collecting your object.

When you find a reference that shouldn't be there, modify your code to break the reference, and then profile the application again to check the problem is fixed.


ConfigurationManager.s_configSystem
System.Configuration.ClientConfigurationSystem
_completeConfigRecord
System.Configuration.RuntimeConfigurationRecord
(this as BaseConfigurationRecord)._sectionRecords
System.Collections.Hashtable
buckets
System.Collections.Hashtable+bucket[]
[6].val
System.Configuration.SectionRecord
_resultRuntimeObject
_result
System.Windows.Forms.WindowsFormsSection
(this as ConfigurationElement)._values
System.Configuration.ConfigurationValues
(this as NameObjectCollectionBase)._entriesArray
System.Collections.ArrayList
_items
GC Handle
ShapePainter.Form1
statusBar
System.Windows.Forms.StatusBar
panels
System.Collections.ArrayList
_items
GC Handle
System.Object[]
[113]
System.Object[]
START HERE!
This graph shows the chains of references to your object that prevent the garbage collector from collecting it. Search up the chains for unexpected references (e.g. event handlers that have not been unregistered), and modify your code to break these references.

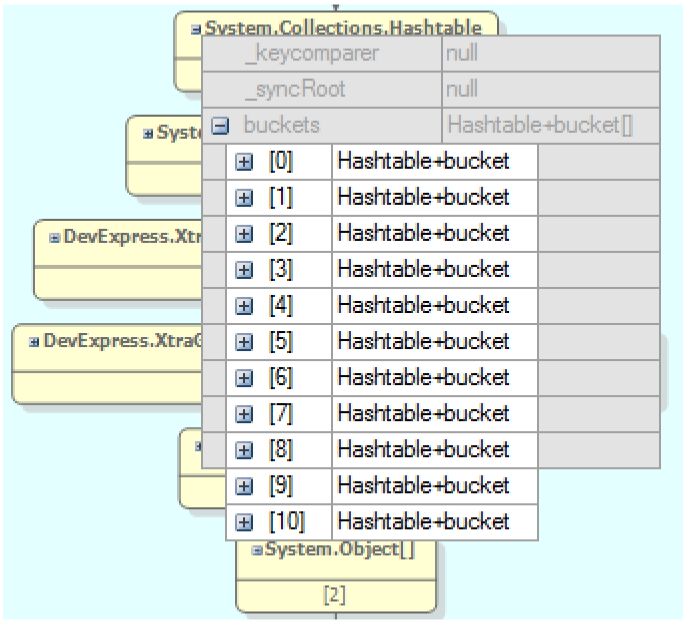
Key

	The instance you selected.
	GC root object.
	The object under this one in the graph is a GC root because it is a GC Handle.
	The object under this one in the graph is a GC root because it is on the stack, having been put on the stack by `bz.d()`.

	The object under this one in the graph is a GC root because it is a member of the static variable `Connection.s_NullBuffer`.
 GC Handle ShapePainter.Form1 statusBar System.Windows.Forms.StatusBar panels	Group of strongly-connected objects (see tips below).
	Non-disposable instance for which you have source code.
	Non-disposable instance for which you do not have source code.
	Disposable instance for which you have source code.
	Disposable instance for which you do not have source code.
 DevExpress.XtraGrid.GridControl (this as Component).events	Disposed instance.
 System.Object[] [78] System.EventHandler (this as Delegate)._target	The simplest path between these two instances. (Note that if you break this link, the objects may still be connected by a more complex link.)

Move your mouse pointer over an object to view more information in a tooltip.

Click ⊞ on a node in the graph to see the properties of a specific instance.



_keycomparer	null
_syncRoot	null
buckets	Hashtable+bucket[]
[0]	Hashtable+bucket
[1]	Hashtable+bucket
[2]	Hashtable+bucket
[3]	Hashtable+bucket
[4]	Hashtable+bucket
[5]	Hashtable+bucket
[6]	Hashtable+bucket
[7]	Hashtable+bucket
[8]	Hashtable+bucket
[9]	Hashtable+bucket
[10]	Hashtable+bucket

You can't see the properties of an instance if you selected **Make snapshots faster by leaving out object values** when setting up the profiling session, or if the snapshot was bigger than 4GB.

Tips

- The instance retention graph only shows the shortest chain of references from each GC root to your selected object. When you break this chain of references, the object may still be kept in memory by a longer chain of references.
  After you have modified your code to break the first chain of references, profile your application again; the instance retention graph updates to show the chain of references which is now the shortest chain. You will need to modify your code again, to break this chain of references, and repeat until all the chains of references are broken and the object is no longer in memory.
- Objects grouped in a box are strongly connected; every object references every other object in the group (the reference may be direct or indirect). To remove an object from memory, you do not normally need to break all the references between a GC root and your object: only *one*of the references needs to be removed to prevent your object from being kept in memory.

  However, the relationship between strongly connected objects is complex, so in this case you may need to break more than one reference to prevent your object from being held in memory. Break the references one at a time, and take new snapshots each time to check whether your object is still in memory.
- If there is an event handler in the chain of references from a GC root to your object, look at the objects that directly reference the event handler; these references are often a good point to break the chain of references to your object.
- If your graph shows an object which does not appear to be referenced by anything, it may be because this object is on the finalizer queue. The graph hides finalizer queue GC roots by default, because they do not normally indicate a memory problem. To show these objects on the graph, clear the **Hide finalizer queue GC roots** option (in the bar above the graph).
- If the graph shows your object is being kept in memory by an object on the finalizer queue, take another memory snapshot. Taking a snapshot forces the garbage collector to run, so your object should now be removed from memory.