




Working with the Instance List



The **instance list** shows information about all instances of a class. This can help to identify instances that may be the cause of a memory problem. When you apply filters, the instance list only includes the filtered objects.

| New Object | Value | Size (bytes) | Size with Children (bytes) | GC Root Object | Distance from GC Root |
|------------|-----------|--------------|----------------------------|----------------|-----------------------|
| No | Signature | 48 | 60 | No | 7 |
| Yes | Signature | 48 | 60 | No | 7 |
| No | Signature | 48 | 68 | No | 3 |
| Yes | Signature | 48 | 72 | No | 10 |
| Yes | Signature | 48 | 68 | No | 10 |
| Yes | Signature | 48 | 72 | No | 10 |
| Yes | Signature | 48 | 68 | No | 10 |
| Yes | Signature | 48 | 72 | No | 10 |
| Yes | Signature | 48 | 68 | No | 10 |
| Yes | Signature | 48 | 72 | No | 10 |

The instance list is useful for understanding what instances of a class are in memory, and for identifying objects which are likely to be involved in a memory leak.

Select an instance and then click  in the **Value** column to find out more about the properties of the specific instance.

 **Instance List for SystemEvents**  Showing 1 of 1 objects (0 fi

| New Object | Value | Size (t |
|------------|--|---------|
| | <div><div></div>consoleHandlerNativeMethods+ConHndlr</div> <div>windowHandle+0x000811e6</div> <div><div></div>windowProcNativeMethods+WndProc</div> | |

New Object

The **New Object** column indicates whether the object was created between two snapshots you are comparing, or whether it existed already in the earlier snapshot. (This column is only populated when you are comparing two snapshots.)

When you are comparing two snapshots, either **Yes** or **No** in the **New Object** column may be an indication of leaked objects, depending on when you took your snapshots.

- **Yes** may indicate a memory leak when you are comparing snapshots and you expect instances of the selected class to be cleaned up by the garbage collector between snapshots. For example, you take a snapshot before and after opening and closing a dialog box. Objects created by the action should be cleaned up before you take the second snapshot, so new objects in the second snapshot are likely to indicate a memory leak.

When you investigate these new objects further, apply the **New objects** filter; this ensures you are only looking at objects which are new in the second snapshot.


- **No** may indicate a memory leak when you are comparing snapshots and you expect instances that exist before the first snapshot to be cleaned up by an action you take between snapshots. For example, you populate a list with data, and then take a snapshot before and after clearing the list. Objects created before the first snapshot should be cleaned up before you take the second snapshot, so old objects in the second snapshot are likely to indicate a memory leak.

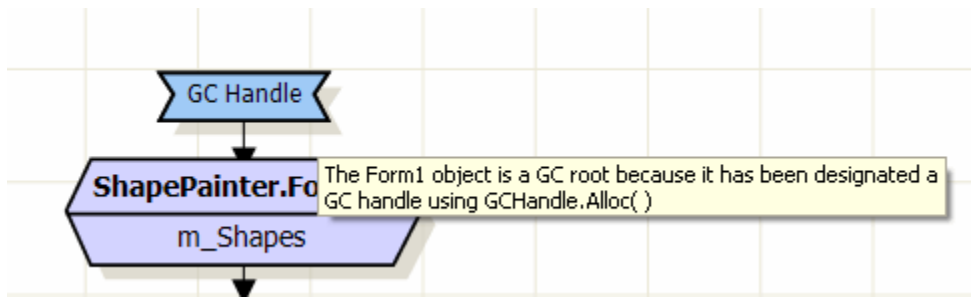
When you investigate these objects further, apply the **Surviving objects** filter; this ensures you are only looking at objects which exist in both snapshots.

GC Root Object

The **GC Root Object** column indicates whether the object is a GC root object. A GC root can be any storage slot to which the running program has access, such as a local variable, static variables, or a CPU register. (Note that the object itself is not the GC root; the storage slot that holds the reference to the object is the GC root.)

When the garbage collector runs, it determines which objects are not garbage by 'walking the heap', starting at the GC roots. Objects which can be reached by following a chain of references from a GC root are designated as not garbage, and are not collected.

To find out why an object in the instance list is a GC root, select it and then click . Information on the graph shows why the object is a GC root.



GC root objects are not usually the source of memory leaks. However, they can be useful in *finding* memory leaks because there is always a chain of references between the leaked object and one or more GC roots. To enable the garbage collector to clean up the object, you need to break this chain of references by changing your code to remove one of the "links" in the chain.

The GC Root Object column shows 'Yes - Weakly Referenced' if the object is weakly referenced. Objects with weak references are often used for caching because these objects can be destroyed by the garbage collector if memory becomes low. For this reason, weakly referenced objects are not usually the source of memory leaks.

Distance from GC Root

The **Distance from GC Root** column shows the number of references in the chain between the object and its nearest GC root.

It is likely that shorter, more obvious chains of references between objects and their GC roots have been broken already. Often objects which are at a greater distance from a GC root may be involved in a memory leak, because the chain of references from the GC root to the object is more complex.

Size with children

The **Size with children** column shows the size of the object and any object that it references that is further away from a GC root. This means that the value is a realistic estimate of the amount of memory that would be saved by removing a particular object from memory.