
Subject: Re: memory leaks

Posted by [Elwin Dijck](#) on Mon, 14 Sep 2009 12:07:22 GMT

[View Forum Message](#) <> [Reply to Message](#)

Whether using `Clear()` instead of `Delete()` is a memory leak indeed depends on the type of class in the `TClonesArray`. I had some problems with `TClonesArrays` earlier (see this thread); this is what I think that the different calls do, please correct me if I'm wrong:

1. `Clear()`

Marks the memory occupied by the objects in the `TClonesArray` as available, without running destructors and without actually deallocating anything. The memory will be reused for new objects put into the array, which will overwrite the remains of the old objects.

2. `Clear("C")`

Same as previous, but will call the function `Clear()` on each of the objects in the array before marking the memory as available (note that `TObject` has an implementation of `Clear()` that does nothing, which should then be overloaded).

3. `Delete()`

Will run the destructor for each of the objects in the array, still without actually freeing the memory of the objects themselves and then does the same as 1. This happens in a kind of weird way and is supposed to be slow (probably only matters for large arrays though, not sure).

However, calling `Delete()` is needed for classes containing for instance `TStrings` as data members, to make sure the internal (dynamically allocated) storage of the strings is deallocated.

I think this would mean the following:

Because `FairHit` doesn't dynamically allocate anything, indeed just using `Clear()` will be ok.

Classes that do dynamically allocate objects should have a proper `Clear()` function to be able to use `Clear("C")` on the `TClonesArray`, otherwise `Delete()` is needed to prevent memory leaks.

For classes that contain (not dynamically allocated) strings or other objects like containers, that themselves dynamically allocate memory, `Delete()` is needed. I don't think it would be possible to prevent memory leaks by implementing some `Clear()` function in these cases (unless all these objects have a function that makes them deallocate their internal storage, but for `TString` and STL classes, only destructors do that I think, since it would typically bring the objects into an invalid state).

Using `Delete()` is the safest thing to do but also the slowest, so it might indeed be useful to check when it actually matters.

When just using `TClonesArrays` while reading from a `TTree` using `GetEntry()`, there is normally no need to do any `Clear()` or `Delete()` as `ROOT` will free memory automatically, though it might still be useful when there are transient data members involved.

Best regards,
Elwin Dijck
