TClonesArray Class Reference

An array of clone (identical) objects.

Memory for the objects stored in the array is allocated only once in the lifetime of the clones array. All objects must be of the same class. For the rest this class has the same properties as TObjArray.

To reduce the very large number of new and delete calls in large loops like this (O(100000) x O(10000) times new/delete):

```
TObjArray a(10000);
while (TEvent *ev = (TEvent *)next()) {      // O(100000) events
   for (int i = 0; i < ev->Ntracks; i++) {   // O(10000) tracks
      a[i] = new TTrack(x,y,z,...);
      ...
      ...
   }
   ...
   a.Delete();
}
```

One better uses a TClonesArray which reduces the number of new/delete calls to only O(10000):

```
TClonesArray a("TTrack", 10000);
while (TEvent *ev = (TEvent *)next()) {      // O(100000) events
   for (int i = 0; i < ev->Ntracks; i++) {   // O(10000) tracks
      new(a[i]) TTrack(x,y,z,...);
      ...
      ...
   }
   ...
   a.Delete(); // or a.Clear() or a.Clear("C")
}
```

To reduce the number of call to the constructor (especially useful if the user class requires memory allocation), the object can be added (and constructed when needed) using ConstructedAt which only calls the constructor once per slot.

```
TClonesArray a("TTrack", 10000);
while (TEvent *ev = (TEvent *)next()) {       // O(100000) events
   for (int i = 0; i < ev->Ntracks; i++) {    // O(10000) tracks
      TTrack *track = (TTrack*)a.ConstructedAt(i);
      track->Set(x,y,z,....);
      ...
      ...
   }
   ...
   a.Clear(); // or a.Clear("C");
}
```

Note: the only supported way to add objects to a TClonesArray is via the new with placement method or the ConstructedAt method. The other Add() methods ofTObjArray and its base classes are not allowed.

Considering that a new/delete costs about 70 mus on a 300 MHz HP, O(10^9) new/deletes will save about 19 hours.

NOTE 1

C/C++ offers the possibility of allocating and deleting memory. Forgetting to delete allocated memory is a programming error called a "memory leak", i.e. the memory of your process grows and eventually your program crashes. Even if you *always* delete the allocated memory, the recovered space may not be efficiently reused. The process knows that there are portions of free memory, but when you allocate it again, a fresh piece of memory is grabbed. Your program is free from semantic errors, but the total memory of your process still grows, because your program's memory is full of "holes" which reduce the efficiency of memory access; this is called "memory fragmentation". Moreover new / delete are expensive operations in terms of CPU time.

Without entering into technical details,TClonesArray allows you to "reuse" the same portion of memory for new/delete avoiding memory fragmentation and memory growth and improving the performance by orders of magnitude. Every time the memory of the TClonesArray has to be reused, the Clear() method is used. To provide its benefits, eachTClonesArray must be allocated*once* per process and disposed of (deleted)*only when not needed any more*.

So a job should see *only one* deletion for each TClonesArray, which should be Clear()ed during the job several times. Deleting aTClonesArray is a double waste. Not only you do not avoid memory fragmentation, but you worsen it because the TClonesArray itself is a rather heavy structure, and there is quite some code in the destructor, so you have more memory fragmentation and slower code.

NOTE 2

When investigating misuse of TClonesArray, please make sure of the following:

- Use Clear() or Clear("C") instead of Delete(). This will improve program execution time.
- TClonesArray object classes containing pointers allocate memory. To avoid causing memory leaks, special Clear("C") must be used for clearing TClonesArray. When option "C" is specified, ROOT automatically executes the Clear() method (by default it is empty contained in TObject). This method must be overridden in the relevant TClonesArray object class, implementing the reset procedure for pointer objects.
- If the objects are added using the placement new then the Clear must deallocate the memory.
- If the objects are added using TClonesArray::ConstructedAt then the heap-based memory can stay allocated and reused as the constructor is not called for already constructed/added object.
- To reduce memory fragmentation, please make sure that the TClonesArrays are not destroyed and created on every event. They must only be constructed/destructed at the beginning/end of the run.

Definition at line 32 of file TClonesArray.h.

Public Types

enum EStatusBits { kBypassStreamer = BIT(12), kForgetBits = BIT(15) }
Saved copies of pointers to objects. More...

▸ Public Types inherited from TObjArray

▸ Public Types inherited from TCollection

▸ Public Types inherited from TObject

Public Member Functions

TClonesArray ()
Default Constructor. More...

TClonesArray (const char *classname, Int_t size=1000, Bool_t call_dtor=kFALSE)
Create an array of clone objects of classname. More...

TClonesArray (const TClass *cl, Int_t size=1000, Bool_t call_dtor=kFALSE)
Create an array of clone objects of class cl. More...

TClonesArray (const TClonesArray &tc)
Copy ctor. More...

virtual	~TClonesArray () Delete a clones array. More...
void	AbsorbObjects (TClonesArray *tc) Directly move the object pointers from tc without cloning (copying). More...
void	AbsorbObjects (TClonesArray *tc, Int_t idx1, Int_t idx2) Directly move the range of object pointers from tc without cloning (copying). More...
void	AddAfter (const TObject *, TObject *) Add object in the slot after object after. More...
void	AddAt (TObject *, Int_t) Add object at position ids. More...
void	AddAtAndExpand (TObject *, Int_t) Add object at position idx. More...
Int_t	AddAtFree (TObject *) Return the position of the new object. More...
void	AddBefore (const TObject *, TObject *) Add object in the slot before object before. More...
void	AddFirst (TObject *) Add object in the first slot of the array. More...
void	AddLast (TObject *) Add object in the next empty slot in the array. More...
TObject *	AddrAt (Int_t idx)
void	BypassStreamer (Bool_t bypass=kTRUE) When the kBypassStreamer bit is set, the automatically generated Streamer can call directlyTClass::WriteBuffer. More...
Bool_t	CanBypassStreamer () const
virtual void	Clear (Option_t *option="") Clear the clones array. More...
virtual void	Compress () Remove empty slots from array. More...

TObject *	ConstructedAt (Int_t idx) Get an object at index 'idx' that is guaranteed to have been constructed. More...
TObject *	ConstructedAt (Int_t idx, Option_t *clear_options) Get an object at index 'idx' that is guaranteed to have been constructed. More...
virtual void	Delete (Option_t *option="") Clear the clones array. More...
virtual void	Expand (Int_t newSize) Expand or shrink the array to newSize elements. More...
virtual void	ExpandCreate (Int_t n) Expand or shrink the array to n elements and create the clone objects by calling their default ctor. More...
virtual void	ExpandCreateFast (Int_t n) Expand or shrink the array to n elements and create the clone objects by calling their default ctor. More...
TClass *	GetClass () const
void	MultiSort (Int_t nTCs, TClonesArray **tcs, Int_t upto=kMaxInt) Sort multiple TClonesArrays simultaneously with this array. More...
TObject *	New (Int_t idx) Create an object of type fClass with the default ctor at the specified index. More...
TClonesArray &	operator= (const TClonesArray &tc) Assignment operator. More...
TObject *&	operator[] (Int_t idx) Return pointer to reserved area in which a new object of clones class can be constructed. More...
TObject *	operator[] (Int_t idx) const Return the object at position idx. Returns 0 if idx is out of bounds. More...
virtual TObject *	Remove (TObject *obj) Remove object from array. More...
virtual TObject *	RemoveAt (Int_t idx) Remove object at index idx. More...
virtual void	RemoveRange (Int_t idx1, Int_t idx2)

	Remove objects from index idx1 to idx2 included. More...
void	SetClass (const char *classname, Int_t size=1000) see TClonesArray::SetClass(const TClass*) More...
void	SetClass (const TClass *cl, Int_t size=1000) Create an array of clone objects of class cl. More...
virtual void	SetOwner (Bool_t enable=kTRUE) A TClonesArray is always the owner of the object it contains. More...
virtual void	Sort (Int_t upto=kMaxInt) If objects in array are sortable (i.e. More...

▸ Public Member Functions inherited from TObjArray

▸ Public Member Functions inherited from TSeqCollection

▸ Public Member Functions inherited from TCollection

▸ Public Member Functions inherited from TObject

Protected Attributes

TClass *	fClass
TObjArray *	fKeep Pointer to the class of the elements. More...

▸ Protected Attributes inherited from TObjArray

▸ Protected Attributes inherited from TSeqCollection

▸ Protected Attributes inherited from TCollection

Additional Inherited Members

▸ Static Public Member Functions inherited from TSeqCollection

▸ Static Public Member Functions inherited from TCollection

▸ Static Public Member Functions inherited from TObject

▸ Protected Types inherited from **TCollection**

▸ Protected Member Functions inherited from **TObjArray**

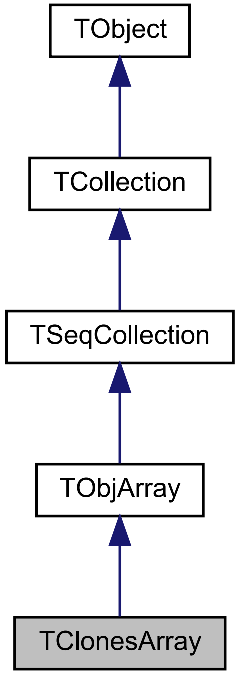
▸ Protected Member Functions inherited from **TSeqCollection**

▸ Protected Member Functions inherited from **TCollection**

▸ Protected Member Functions inherited from **TObject**

```
#include <TClonesArray.h>
```

Inheritance diagram for TClonesArray:



[legend]

Member Enumeration Documentation

◆ EStatusBits

enum TClonesArray::EStatusBits

Saved copies of pointers to objects.

Enumerator	
kBypassStreamer	
kForgetBits	

Definition at line 39 of file TClonesArray.h.

Constructor & Destructor Documentation

◆ TClonesArray() [1/4]

TClonesArray::TClonesArray ( )

Default Constructor.

Definition at line 159 of file TClonesArray.cxx.

◆ TClonesArray() [2/4]

```
TClonesArray::TClonesArray ( const char *  classname,
                             Int_t         s = 1000,
                             Bool_t        call_dtor = kFALSE
                           )
```

Create an array of clone objects of classname.

The class must inherit fromTObject. The second argument s indicates an approximate number of objects that will be entered in the array. If more than s objects are entered, the array will be automatically expanded.

The third argument is not used anymore and only there for backward compatibility reasons.

Definition at line 175 of file TClonesArray.cxx.

◆ TClonesArray() [3/4]

```
TClonesArray::TClonesArray ( const TClass *  cl,
                             Int_t           s = 1000,
                             Bool_t          call_dtor = kFALSE
                           )
```

Create an array of clone objects of class cl.

The class must inherit fromTObject. The second argument, s, indicates an approximate number of objects that will be entered in the array. If more than s objects are entered, the array will be automatically expanded.

The third argument is not used anymore and only there for backward compatibility reasons.

Definition at line 191 of file TClonesArray.cxx.

◆ TClonesArray() [4/4]

TClonesArray::TClonesArray (const TClonesArray & tc)

Copy ctor.

Definition at line 200 of file TClonesArray.cxx.

◆ ~TClonesArray()

TClonesArray::~TClonesArray ( ) virtual

Delete a clones array.

Definition at line 252 of file TClonesArray.cxx.

Member Function Documentation

◆ AbsorbObjects() [1/2]

void TClonesArray::AbsorbObjects (TClonesArray * tc)

Directly move the object pointers from tc without cloning (copying).

This TClonesArray takes over ownership of all of tc's object pointers. The tc array is left empty upon return.

Definition at line 931 of file TClonesArray.cxx.

◆ AbsorbObjects() [2/2]

```
void TClonesArray::AbsorbObjects ( TClonesArray *  tc,
                                   Int_t           idx1,
                                   Int_t           idx2
                                 )
```

Directly move the range of object pointers from tc without cloning (copying).

This TClonesArray takes over ownership of all of tc's object pointers from idx1 to idx2. The tc array is re-arranged by return.

Definition at line 944 of file TClonesArray.cxx.

◆ AddAfter()

```
void TClonesArray::AddAfter ( const TObject *  after,
                              TObject *        obj
                            )
```

inline virtual

Add object in the slot after object after.

If after=0 add object in the last empty slot. Note that this will overwrite any object that might have already been in this slot. For insertion semantics use either a TList or a TOrdCollection.

Reimplemented from TObjArray.

Definition at line 64 of file TClonesArray.h.

◆ AddAt()

```
void TClonesArray::AddAt ( TObject * obj,
                           Int_t     idx
                         )
```
inline virtual

Add object at position ids.

Give an error when idx is out of bounds (i.e. the array is not expanded).

Reimplemented from TObjArray.

Definition at line 61 of file TClonesArray.h.

◆ AddAtAndExpand()

```
void
TClonesArray::AddAtAndExpand   ( TObject * obj,
                                 Int_t     idx
                               )
```
inline virtual

Add object at position idx.

If idx is larger than the current size of the array, expand the array (double its size).

Reimplemented from TObjArray.

Definition at line 62 of file TClonesArray.h.

◆ AddAtFree()

Int_t TClonesArray::AddAtFree (TObject * obj) inline virtual

Return the position of the new object.

Find the first empty cell or AddLast if there is no empty cell

Reimplemented from TObjArray.

Definition at line 63 of file TClonesArray.h.

◆ AddBefore()

```
void
TClonesArray::AddBefore    ( const TObject *  before,
                             TObject *        obj
                           )
```
inline virtual

Add object in the slot before object before.

If before=0 add object in the first slot. Note that this will overwrite any object that might have already been in this slot. For insertion semantics use either a TList or a TOrdCollection.

Reimplemented from TObjArray.

Definition at line 65 of file TClonesArray.h.

◆ AddFirst()

void TClonesArray::AddFirst (TObject * obj) inline virtual

Add object in the first slot of the array.

This will overwrite the first element that might have been there. To have insertion semantics use either a TList or a TOrdCollection.

Reimplemented from TObjArray.

Definition at line 59 of file TClonesArray.h.

◆ AddLast()

void
TClonesArray::AddLast (TObject * obj) inline virtual

Add object in the next empty slot in the array.

Expand the array if necessary.

Reimplemented from TObjArray.

Definition at line 60 of file TClonesArray.h.

◆ AddrAt()

TObject *
TClonesArray::AddrAt (Int_t idx) inline

Definition at line 89 of file TClonesArray.h.

◆ BypassStreamer()

```
void
TClonesArray::BypassStreamer        ( Bool_t  bypass = kTRUE )
```

When the kBypassStreamer bit is set, the automatically generated Streamer can call directly TClass::WriteBuffer.

Bypassing the Streamer improves the performance when writing/reading the objects in the TClonesArray. However there is a drawback: When a TClonesArray is written with split=0 bypassing the Streamer, the StreamerInfo of the class in the array being optimized, one cannot use later the TClonesArray with split>0. For example, there is a problem with the following scenario:

1. A class Foo has a TClonesArray of Bar objects
2. The Foo object is written with split=0 to Tree T1. In this case the StreamerInfo for the class Bar is created in optimized mode in such a way that data members of the same type are written as an array improving the I/O performance.
3. In a new program, T1 is read and a new Tree T2 is created with the object Foo in split>1
4. When the T2 branch is created, the StreamerInfo for the class Bar is created with no optimization (mandatory for the split mode). The optimized Bar StreamerInfo is going to be used to read the TClonesArray in T1. The result will be Bar objects with data member values not in the right sequence. The solution to this problem is to call BypassStreamer(kFALSE) for the TClonesArray. In this case, the normal Bar::Streamer function will be called. The Bar::Streamer function works OK independently if the Bar StreamerInfo had been generated in optimized mode or not.

Definition at line 292 of file TClonesArray.cxx.

◆ CanBypassStreamer()

```
Bool_t TClonesArray::CanBypassStreamer ( ) const                                    inline
```

Definition at line 67 of file TClonesArray.h.

◆ Clear()

void
TClonesArray::Clear (Option_t * option = "") virtual

Clear the clones array.

Only use this routine when your objects don't allocate memory since it will not call the object dtors. However, if the class in the TClonesArray implements the function Clear(Option_t *option) and if option = "C" the function Clear() is called for all objects in the array. In the function Clear(), one can delete objects or dynamic arrays allocated in the class. This procedure is much faster than calling TClonesArray::Delete(). When the option starts with "C+", eg "C+xyz" the objects in the array are in turn cleared with the option "xyz"

Reimplemented from TObjArray.

Definition at line 391 of file TClonesArray.cxx.

◆ Compress()

void TClonesArray::Compress ( ) virtual

Remove empty slots from array.

Reimplemented from TObjArray.

Definition at line 303 of file TClonesArray.cxx.

◆ ConstructedAt() [1/2]

TObject *
TClonesArray::ConstructedAt (Int_t idx)

Get an object at index 'idx' that is guaranteed to have been constructed.

It might be either a freshly allocated object or one that had already been allocated (and assumingly used). In the later case, it is the callers responsibility to insure that the object is returned to a known state, usually by calling the Clear method on the TClonesArray.

Tests to see if the destructor has been called on the object. If so, or if the object has never been constructed the class constructor is called using New(). If not, return a pointer to the correct memory location. This explicitly to deal withTObject classes that allocate memory which will be reset (but not deallocated) in their Clear() functions.

Definition at line 348 of file TClonesArray.cxx.

◆ ConstructedAt() [2/2]

```
TObject *
TClonesArray::ConstructedAt ( Int_t      idx,
                              Option_t * clear_options
                            )
```

Get an object at index 'idx' that is guaranteed to have been constructed.

It might be either a freshly allocated object or one that had already been allocated (and assumingly used). In the later case, the function Clear will be called and passed the value of 'clear_options'

Tests to see if the destructor has been called on the object. If so, or if the object has never been constructed the class constructor is called using New(). If not, return a pointer to the correct memory location. This explicitly to deal withTObject classes that allocate memory which will be reset (but not deallocated) in their Clear() functions.

Definition at line 370 of file TClonesArray.cxx.

◆ Delete()

void TClonesArray::Delete (Option_t * option = "") virtual

Clear the clones array.

Use this routine when your objects allocate memory (e.g. objects inheriting fromTNamed or containing TStrings allocate memory). If not you better use Clear() since if is faster.

Reimplemented from TObjArray.

Definition at line 423 of file TClonesArray.cxx.

◆ Expand()

void TClonesArray::Expand (Int_t newSize) virtual

Expand or shrink the array to newSize elements.

Reimplemented from TObjArray.

Definition at line 450 of file TClonesArray.cxx.

◆ ExpandCreate()

void TClonesArray::ExpandCreate (Int_t n) virtual

Expand or shrink the array to n elements and create the clone objects by calling their default ctor.

If n is less than the current size the array is shrunk and the allocated space is freed. This routine is typically used to create a clonesarray into which one can directly copy object data without going via the "new (arr[i]) MyObj()" (i.e. the vtbl is already set correctly).

Definition at line 480 of file TClonesArray.cxx.

◆ ExpandCreateFast()

void
TClonesArray::ExpandCreateFast (Int_t n) virtual

Expand or shrink the array to n elements and create the clone objects by calling their default ctor.

If n is less than the current size the array is shrunk but the allocated space is *not* freed. This routine is typically used to create a clonesarray into which one can directly copy object data without going via the "new (arr[i]) MyObj()" (i.e. the vtbl is already set correctly). This is a simplified version of ExpandCreate used in the TTree mechanism.

Definition at line 520 of file TClonesArray.cxx.

◆ GetClass()

TClass*
TClonesArray::GetClass ( ) const inline

Definition at line 56 of file TClonesArray.h.

◆ MultiSort()

```
void TClonesArray::MultiSort ( Int_t            nTCs,
                               TClonesArray **  tcs,
                               Int_t            upto = kMaxInt
                             )
```

Sort multiple TClonesArrays simultaneously with this array.

If objects in array are sortable (i.e.IsSortable() returns true for all objects) then sort array.

Definition at line 1000 of file TClonesArray.cxx.

◆ New()

```
TObject *
TClonesArray::New    ( Int_t  idx )
```

Create an object of type fClass with the default ctor at the specified index.

Returns 0 in case of error.

Definition at line 907 of file TClonesArray.cxx.

◆ operator=()

TClonesArray &
TClonesArray::operator= (const TClonesArray & tc)

Assignment operator.

Definition at line 216 of file TClonesArray.cxx.

◆ operator[]() [1/2]

TObject *&
TClonesArray::operator[] (Int_t idx) virtual

Return pointer to reserved area in which a new object of clones class can be constructed.

This operator should not be used for lefthand side assignments, like a[2] = xxx. Only like, new (a[2]) myClass, or xxx = a[2]. Of course right hand side usage is only legal after the object has been constructed via the new operator or via the New() method. To remove elements from the clones array use Remove() or RemoveAt().

Reimplemented from TObjArray.

Definition at line 859 of file TClonesArray.cxx.

◆ operator[]() [2/2]

TObject *
TClonesArray::operator[] (Int_t idx) const virtual

Return the object at position idx. Returns 0 if idx is out of bounds.

Reimplemented from TObjArray.

Definition at line 893 of file TClonesArray.cxx.

◆ Remove()

TObject *
TClonesArray::Remove (TObject * obj) virtual

Remove object from array.

Reimplemented from TObjArray.

Definition at line 570 of file TClonesArray.cxx.

◆ RemoveAt()

TObject *
TClonesArray::RemoveAt (Int_t idx) virtual

Remove object at index idx.

Reimplemented from TObjArray.

Definition at line 546 of file TClonesArray.cxx.

◆ RemoveRange()

void
TClonesArray::RemoveRange (Int_t idx1,
Int_t idx2
) virtual

Remove objects from index idx1 to idx2 included.

Reimplemented from TObjArray.

Definition at line 593 of file TClonesArray.cxx.

◆ SetClass() [1/2]

```
void TClonesArray::SetClass ( const char *  classname,
                              Int_t         size = 1000
                            )
```

see TClonesArray::SetClass(const TClass*)

Definition at line 664 of file TClonesArray.cxx.

◆ SetClass() [2/2]

```
void TClonesArray::SetClass ( const TClass *  cl,
                              Int_t           s = 1000
                            )
```

Create an array of clone objects of class cl.

The class must inherit fromTObject. The second argument s indicates an approximate number of objects that will be entered in the array. If more than s objects are entered, the array will be automatically expanded.

NB: This function should not be called in theTClonesArray is already initialized with a class.

Definition at line 627 of file TClonesArray.cxx.

◆ SetOwner()

void
TClonesArray::SetOwner (Bool_t enable = kTRUE) virtual

A TClonesArray is always the owner of the object it contains.

However the collection its inherits from (TObjArray) does not. Hence this member function needs to be a nop forTClonesArray.

Reimplemented from TCollection.

Definition at line 675 of file TClonesArray.cxx.

◆ Sort()

void
TClonesArray::Sort (Int_t upto = kMaxInt) virtual

If objects in array are sortable (i.e.

IsSortable() returns true for all objects) then sort array.

Reimplemented from TObjArray.

Definition at line 684 of file TClonesArray.cxx.

Member Data Documentation

◆ fClass

TClass* TClonesArray::fClass protected

Definition at line 35 of file TClonesArray.h.

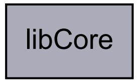
◆ fKeep

TObjArray* TClonesArray::fKeep protected

Pointer to the class of the elements.

Definition at line 36 of file TClonesArray.h.

Libraries for TClonesArray:



[legend]

The documentation for this class was generated from the following files:

- core/cont/inc/TClonesArray.h
- core/cont/src/TClonesArray.cxx