ROOT I/O

Introduction

The Root I/O system consists of several levels of standards which provide increasing levels of functionality. A core I/O level provides classes to read and write arbitrarily defined sets of bytes in a directory-like hierarchical structure. An object I/O level provides additional classes that allow reading and writing user objects in a self describing format. This level provides automatic schema evolution and provides for inter object references. A third level allows optimized storage of and access to large collections of objects. Additional classes exist for specific types of data in common use in HEP (such as histograms) but are not really part of the ROOT I/O system.

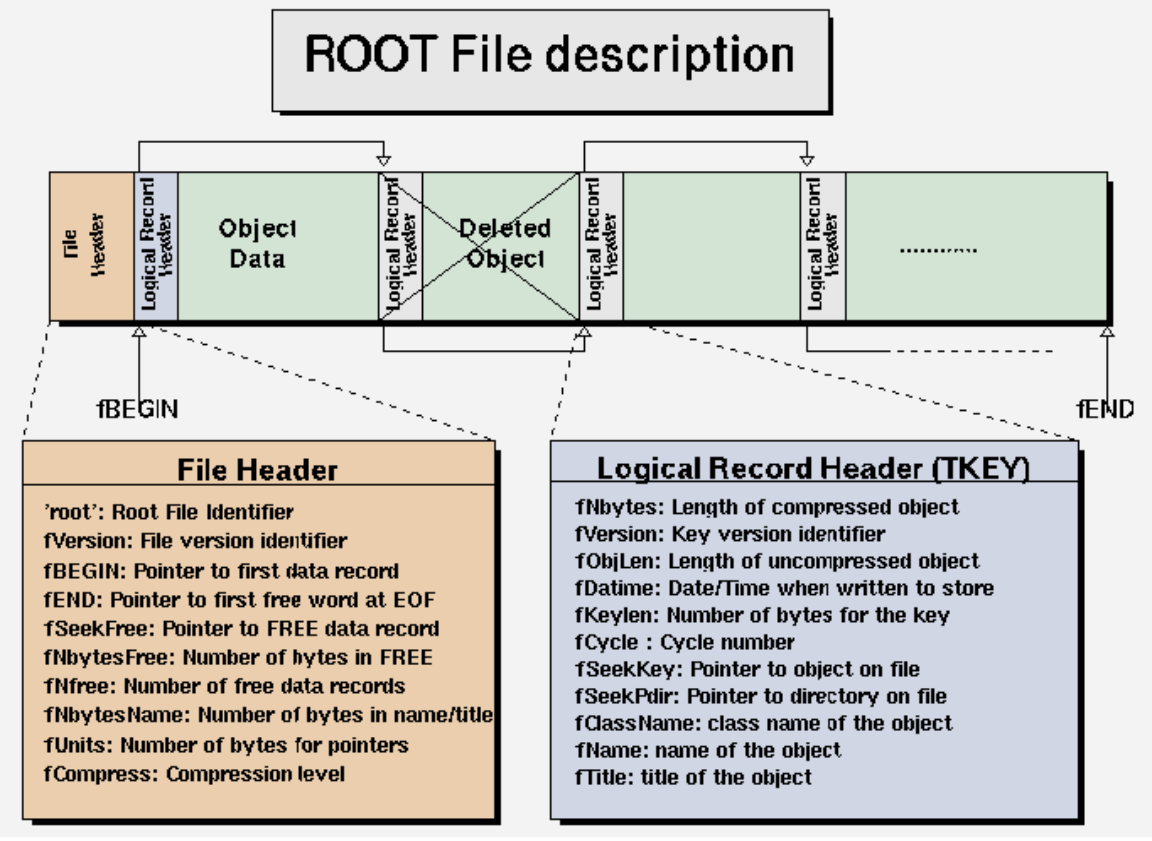
ROOT I/O can be used both from within the ROOT framework and in stand alone applications. In general ROOT I/O files written in other applications will also be able to be read from within ROOT. Full ROOT functionality depends on top level user objects either inheriting from the ROOT class TObject or being provided with some translation mechanism when they are input into ROOT.

We document here both the logical and physical file structure for each of these layers.

Overview

ROOTIO FILES

A ROOTIO file consists of one "file header", one or more "data records," and zero or more "free segments". The file header is always at the beginning of the file, while the data records and free segments may in principle appear in any order.



The file header is fixed length (64 bytes in the current release.) Its detailed format is given below in the section of detailed class descriptions.

A free segment is of variable length. One free segment is a set of contiguous bytes that are unused, and are available for ROOTIO to use for new or resized data records. The first four bytes of a a free segment contain the negative of the number of bytes in the segment. The contents of the remainder of the free segment are irrelevant.

A data record represents either user data or data used internally by ROOTIO. All data records have two portions, a "key" portion and a "data" portion. The key portion precedes the data portion. The format of the key portion is the same for all data. (The key portion corresponds to a class TKey object). The object name and they key cycle are together sufficient to uniquely determine the record within the file. The format of the data portion of a record for an object that uses the default streamer is given below in the section of detailed class descriptions.

In more detail, data records can be one of:

1. TFile record: One per file. Describes file as a whole or “root” of directory structure
2. TDirectory record(s): Zero or more per file. Each describes one (non-root) directory in tree
3. KeysList record(s): One per TFile or TDirectory Record. Contains keys to access records in the directory.
4. FreeSegments record: One per file. Controls access to free segments.
5. TStreamerInfo record: One per file. Contains info describing “self-describing” objects
6. TRef, TRefArray, and TProcessID: Used for inter-object references.
7. Records containing user objects or collections of user objects.

DATA RECORD TYPES

"core" record types

There are several types of data records used internally by ROOTIO to support the storage of byte sequences. These record types are "TFile", "TDirectory", "KeysList", and "FreeSegments". These types can be considered to be in the "core" layer of ROOTIO.

A file always contains exactly one "TFile" data record, which (nearly?) always immediately follows the file header. The TFile record consists of either data pertaining to the file as a whole, or data pertaining to the root "directory" of records in the file.

A file contains zero or more "TDirectory" data records, each representing a subdirectory in the directory tree that has the "TFile" record at its root.

A file contains one or more "KeysList" data records. There is one corresponding to the root directory (represented by the TFile record), and one corresponding to each (non-empty) subdirectory in the tree (each represented by a TDirectory record). The data portion of each KeysList record consists of the sequential keys of those data records in that directory. Note that keys for "TFile", "KeysList", "FreeSegments", and "StreamerInfo" data records never appear in the data portion of a KeysList data record.

A file always contains exactly one "FreeSegments" data record, which keeps track of the free segments in the file. Its detailed format is given in "freesegments.txt". Note that the list of free segments contains one additional free segment that is not in the file itself, because it represents the free space after the end of the file

Detailed formats of these record types are given below in the section of detailed class descriptions.

"streamer" layer record types

There is an additional data record type ("StreamerInfo") needed internally to support the storage of self-identifying objects. Its detailed format is given in "streamerinfo.txt". Note that the StreamerInfo data record itself and the "core" data records described above are not self-identifying objects. A ROOTIO file contains exactly one StreamerInfo record. The use of the "StreamerInfo" record is described under the "STREAMERINFO" heading below.

"pointer to persistent object" object types

There are three object types ("TProcessID", "TRef", and "TRefArray") used internally to support pointers to persistent objects. Their formats are given in "tprocessid.txt", "tref.txt", and "trefarray.txt" respectively. Of these three objects, only TProcessID objects necessarily comprise a complete data record (a "TProcessID" record). TRef and TRefArray objects typically are data members of larger objects, and therefore are only a part of the data portion of a record. In addition, objects that are referenced by such a pointer have an additional field in the base TObject. See "tobject.txt". A description of how these pointers work is given under the "POINTERS TO PERSISTENT OBJECTS" heading below.

"application" layer record types

These are either user defined record types, or record types supplied by ROOT that are not needed by ROOTIO. The format of such an object that uses the default streamer is given below.

DATA COMPRESSION

The user can set the data compression level for new or modified data records when creating or opening a file. When an existing file is opened for update, the compression level selected need not match that used previously. The compression level of existing records is not modified unless the record itself is modified.

There are ten compression levels, 0-9, ranging from 0 (no compression) to 9 (maximum compression), with level 1 being the default. The level chosen is a tradeoff between disk space and compression performance. The decompression speed is independent of level. Currently, in release 3.2.6, level 2 is not used. If level 2 is selected, level 1 is used with no notification to the user.

The chosen compression level is not applied to the entire file. The following portions of the file are not compressed, regardless of the compression level selected:

1) the file header
2) the KeysList data record
3) the FreeSegments data record
4) any data record (outside of a TTree) where the uncompressed size of the data portion is 256 bytes or less.
5) the key portion of any data record

Furthermore, the data portion of the StreamerInfo data record is always compressed at level 1 (if over 256 bytes uncompressed), regardless of the compression level selected (even if no compression is selected).

The compression algorithm used is an in memory ZIP compression written for the DELPHI collaboration at CERN. Its author is E. Chernyaev (IHEP/Protvino). The source code is internal to ROOTIO.

STREAMERINFO

The "StreamerInfo" data record is used by ROOTIO to support the storage of self-identifying objects. Its detailed format is given in "streamerinfo.txt". A ROOTIO file contains exactly one StreamerInfo record, which is written to disk automatically when a new or modified file is closed.

The StreamerInfo record is a list (ROOTIO class TList) of "StreamerInfo" objects (ROOTIO class TStreamerInfo). There is one StreamerInfo object in the list for every class used in the file in a data record, other than a core layer record. There is no streamerinfo object for a class used in a core layer record unless the class is also used elsewhere in a data record. When reading a self-identifying object from a file, the system uses the StreamerInfo list to decompose the object recursively into its simple data members.

Each streamerinfo object is an array of "streamer element" objects, each of which describes a base class of the object or a (non-static and non-transient) data member of the object. If the base class or data member is itself a class, then there will also be a streamerinfo object in the record for that class. In this way, each class is recursively decomposed into its atomic elements, each of which is a simple type (e.g. "int"). A "long" or "unsigned long" member is always written as an 8 byte quantity, even if it occupies only 4 bytes in memory.

A data member of a class is marked transient on the line of its declaration by a comment beginning with "//!". Such members are not written to disk, nor is there any streamerinfo for such a member.

A data member that is a C++ pointer (not to be confused with "pointers to persistent objects" described below) is never written to disk as a pointer value. If it is a pointer to an object, the object itself (or 0 (4 bytes) if the pointer value is NULL) is written. If the declaration line has a comment beginning with "//->", this indicates that the pointer value will never be null, which allows a performance optimization. Another optimization is that if two or more pointers pointing to the same object are streamed in the same I/O operation, the object is written only once. The remaining pointers reference the object through a unique object identifier. This saves space and avoids the infinite loop that might otherwise arise if the directed graph of object instance pointer references contains a cycle.

If a data member is a pointer to a simple type, the Streamer presumes it is an array, with the dimension defined in a comment of the form "//[<length>]", where length is either an integer constant or a variable that is an integer data member of the class. If a variable is used, it must be defined ahead of its use or in a base class.

The above describes the function of the StreamerInfo record in decomposing a self-identifying object if the user uses the streamer generated by "rootcint". There are two reasons why a user may need to write a specialized streamer for a class. One reason is that it may be necessary to execute some code before or after data is read or written, for example, to initialize some non-persistent data members after the persistent data is read. In this case, the custom streamer can use the StreamerInfo record to decompose a self-identifying object in the exact same manner as the generated streamer would have done. An example is given (for the Event class) in the Root User's Guide (URL below) (Input/Output chapter, Streamers subchapter). On the other hand, if the user needs to write a streamer for a class that ROOT cannot handle, the user may need to explicitly code the decomposition and composition of the object to its members. In this case, the StreamerInfo for that class might not be used. In any case, if the composition/decomposition of the class is explicitly coded, the user should include the byte count, class information, and version number of the class before the data on disk as shown in "dobject.txt".

The special method used for streaming a TClonesArray is described in the TClonesArray section below.

More information on the StreamerInfo record and its use is found in the Input/Output chapter of the Root Users Guide: http://root.cern.ch/root/RootDoc.html

NOTE: Some of the classes used internally in ROOTIO (e.g. TObject, TRef, TRefArray) have explicitly coded (de)compositions, and do not use the information in the StreamerInfo record to do the (de)composition. In this case, the StreamerInfo for the class may still be present in the StreamerInfo record, but may not match what is actually written to disk for those objects.

POINTERS TO PERSISTENT OBJECTS

Information on how these work in memory can be found at:
http://root.cern.ch/root/htmldoc/examples/Version302.news.html
These were introduced in release 3.02, so there is not yet a description in the current Root Users Guide, which is for a version release 3.1. Here we discuss only the information on disk.

A ROOT file contains zero or more TProcessID records. Each such record contains a globally unique ID defining a given ROOT job that wrote a referenced object (see tprocessid.txt). Each referenced object contains a "pidf" field referencing the corresponding TProcessID record and an "fUniqueID" field uniquely identifying the referenced object among those written by that process (see tobject.txt). Similarly, every persistent reference to that object (a TRef Object, see tref.txt) also contains "pidf" and "fUniqueID" fields with the same value, thereby uniquely determining the referenced object (which need not even be in the same file). In the case of an array of references (a TRefArray object, see "trefarray.txt"), there is one "pidf" value for the entire array, and a separate "fUniqueID" value for each reference. For further information, see the above URL.

SOME USEFUL CONTAINER CLASSES

1. TObjArray and TClonesArray

The TObjArray class can be used to support an array of objects. The objects need not be of the same type, but each object must be of a class type that inherits from TObject. We have already seen a specific example of the use of TObjArray, in the StreamerInfo record, where it is used to hold an array of TStreamerElement objects, each of which is of a class inheriting from TStreamerElement, which in turn inherits from TObject.

The TClonesArray class is a specialization of the TObjArray class for holding an array of objects that are all of the same type. The format of a TClonesArray object is given in "tclonesarray.txt".

There are two great advantages in the use of TClonesArray over TObjArray when the objects all will be of the same class:

1) Memory for the objects will be allocated only once for the entire array, rather than the per-object allocation for TObjArray. This can be done because all the objects are the same size.
2) In the case of TObjArray, the stored objects are written sequentially. However, in a TClonesArray, by default, each object is split one level deep into its base class(es) and data members, and each of these members is written sequentially for all objects in the array before the next member is written. This has two advantages:
    a) Greater compression can be achieved when similar data is consecutive.
    b) The object's data members can easily be split into different TTree branches (TTrees are discussed below).

2. TTree

A TTree is a highly specialized container class for efficient storage and retrieval of user data. The use of TTrees is discussed in detail in the Trees chapter of the Root Users Guide:
http://root.cern.ch/root/RootDoc.html

Here we discuss in particular how a TTree is stored in a ROOTIO file.

A TTree object is split into one or more branches (class TBranch), each of which may have its own (sub)branches, recursively to any depth. Each TBranch contains an array of zero or more leaves (class TLeaf), each corresponding to a basic variable type or a class object that has not been split. The TLeaf object does not actually contain variable values, only information about the variables. The actual data on each branch is physically stored in basket objects (class TBasket). The user can set the basket size on a per TBranch basis. The default basket size is 32000 bytes. This should be viewed as an approximate number.

There is one TTree data record per file for each tree in the file, corresponding to a TTree class object. The TTree class object recursively contains TBranch objects, each of which contains an array of TBasket objects to hold its data.

However, the TTree data record does not necessarily contain the entire TTree object. For each branch, exactly one TBasket object is contained in the TTree data record. If the data on a given branch fits in one basket, then all the data for that branch will be in the TTree record itself. Otherwise, there will be a separate TBasket data record for each additional basket used on the branch, each containing a TBasket object containing user data.

By default, the additional TBasket data records are stored in the same file as that of the corresponding TTree data record. However, the user may specify a separate file for a given branch. If the data for that branch fits into one basket, this option has no effect. Otherwise, the additional TBasket records are written into the specified file, rather than the file containing the TTree data record itself. In this case, a TBranch data record for the specified branch is also written to the specified file, containing the TBranch object for the specified branch.

The file "ttree.txt" shows the streamer information for the TTree, TBranch, TLeaf, and some related classes, together with some additional commentary. For writing to a ROOTIO file, the streamers for these three classes act exactly as those of default generated streamers, except that, if the user has specified a separate file for a branch, the TBranch streamer also writes the TBranch object as a keyed data record to the specified file.

There is no streamer information for the TBasket class. The custom written TBasket streamer internally handles the packing of data into fixed size TBasket objects.

Detailed Class Descriptions

Core I/O

File Header Format

```
// Here is the file header format as of release 3.02.06.  It is never compressed.
// ----------------------------------
// byte  0->3  "root"      = Identifies this file as a ROOT file.
//       4->7  Version     = File format version                          TFile::fVersion
//                         |  (10000*major+100*minor+cycle (e.g. 30203 for 3.2.3))
//       8->11 BEGIN       = Byte offset of first data record (64)
      TFile::fBEGIN
//      12->15 END         = Pointer to first free word at the EOF             TFile::fEND
//                         | (will be == to file size in bytes)
//      16->19 SeekFree    = Byte offset of FreeSegments record
      TFile::fSeekFree
//      20->23 NbytesFree  = Number of bytes in FreeSegments record
      TFile::fNBytesFree
//      24->27 nfree       = Number of free data records
```

```
//      28->31 NbytesName  = Number of bytes in TKey+TNamed for TFile at creation
TDirectory::fNbytesName
//      32->32 Units       = Number of bytes for file pointers (4)
      TFile::fUnits
//      33->36 Compress    = Zip compression level (i.e. 0-9)
      TFile::fCompress
//      37->40 SeekInfo    = Byte offset of StreamerInfo record
      TFile::fSeekInfo
//      41->44 NbytesInfo  = Number of bytes in StreamerInfo record
      TFile::fNbytesInfo
//      45->63             = Unused??
```

Data Record Format

```
//  Release 3.02.06
//  A ROOT file is mostly a suite of consecutive data records with the following format
//    <Name>;<Cycle> uniquely identifies the record in a directory
//    ----------TKey-(never compressed)--------------
//  byte 0->3  Nbytes    = Number of bytes in compressed record (Tkey+data)
      TKey::fNbytes
//       4->5  Version   = TKey class version identifier                TKey::fVersion
//       6->9  ObjLen    = Number of bytes of uncompressed data
      TKey::fObjLen
//      10->13 Datime    = Date and time when record was written to file
      TKey::fDatime
//                       | (year-1995)<<26|month<<22|day<<17|hour<<12|minute<<6|second
//      14->15 KeyLen    = Number of bytes in key structure (TKey)
      TKey::fKeyLen
//      16->17 Cycle     = Cycle of key    (e.g. 1)                    TKey::fCycle
//      18->21 SeekKey   = Byte offset of record itself (consistency check)
      TKey::fSeekKey
//      22->25 SeekPdir  = Byte offset of parent directory record      TKey::fSeekPdir
//      26->26 lname     = Number of bytes in the class name
      TKey::fClassName
//      27->.. ClassName = Object Class Name                           TKey::fClassName
//       0->0  lname     = Number of bytes in the object name
      TNamed::fName
//       1->.. Name      = lName bytes with the name of the object
      TNamed::fName
//       0->0  lTitle    = Number of bytes in the object title
      TNamed::fTitle
//       1->.. Title     = lTitle bytes with the title of the object
      TNamed::fTitle
//    ----------DATA---(may be compressed)-----------
//       0->..    The data object itself.  For an example, see dobject.txt
```

TDirectory

```
//  Format of a TDirectory record in release 3.02.06.  It is never compressed.
//    ----------TKey--------------
//  byte 0->3  Nbytes    = Number of bytes in compressed record (Tkey+data)
      TKey::fNbytes
//       4->5  Version   = TKey class version identifier                TKey::fVersion
//       6->9  ObjLen    = Number of bytes of uncompressed data
      TKey::fObjLen
//      10->13 Datime    = Date and time when record was written to file
      TKey::fDatime
//                       | (year-1995)<<26|month<<22|day<<17|hour<<12|minute<<6|second
//      14->15 KeyLen    = Number of bytes in key structure (TKey)
      TKey::fKeyLen
//      16->17 Cycle     = Cycle of key                                TKey::fCycle
```

```
//      18->21 SeekKey   = Byte offset of record itself (consistency check)
      TKey::fSeekKey
//      22->25 SeekPdir  = Byte offset of parent directory record
      TKey::fSeekPdir
//      26->26 lname     = Number of bytes in the class name (10)        TKey::fClassName
//      27->.. ClassName = Object Class Name ("TDirectory")              TKey::fClassName
//       0->0  lname     = Number of bytes in the object name
      TNamed::fName
//       1->.. Name      = lName bytes with the name of the object <directory name>
TNamed::fName
//       0->0  lTitle    = Number of bytes in the object title
      TNamed::fTitle
//       1->.. Title     = lTitle bytes with the title of the object <direcory title>
TNamed::fTitle
//    --------DATA----------------
//       0->0  Modified  = True if directory has been modified
      TDirectory::fModified
//       1->1  Writeable = True if directory is writeable
      TDirectory::fWriteable
//       2->5  DatimeC   = Date and time when directory was created
      TDirectory::fDatimeC
//                       | (year-1995)<<26|month<<22|day<<17|hour<<12|minute<<6|second
//       6->9  DatimeM   = Date and time when directory was last modified
      TDirectory::fDatimeM
//                       | (year-1995)<<26|month<<22|day<<17|hour<<12|minute<<6|second
//      10->13 NbytesKeys= Number of bytes in the associated KeysList record
      TDirectory::fNbyteskeys
//      14->17 NbytesName= Number of bytes in TKey+TNamed at creation
      TDirectory::fNbytesName
//      18->21 SeekDir   = Byte offset of directory record in file
      TDirectory::fSeekDir
//      22->25 SeekParent= Byte offset of parent directory record in file
      TDirectory::fSeekParent
//      26->29 SeekKeys  = Byte offset of associated KeysList record in file
      TDirectory::fSeekKeys
```

TFile

```
//  Here is the format of a TFile record for release 3.02.06.  It is never compressed.
//  It is probably not accessed by its key, but from its offset given in the file header.
//    ----------TKey---------------
//  byte 0->3  Nbytes    = Number of bytes compressed record (TKey+data)
      TKey::fNbytes
//       4->5  Version   = TKey class version identifier                 TKey::fVersion
//       6->9  ObjLen    = Number of bytes of uncompressed data
      TKey::fObjLen
//      10->13 Datime    = Date and time when record was written to file
      TKey::fDatime
//                       | (year-1995)<<26|month<<22|day<<17|hour<<12|minute<<6|second
//      14->15 KeyLen    = Number of bytes in key structure (TKey)
      TKey::fKeyLen
//      16->17 Cycle     = Cycle of key                                  TKey::fCycle
//      18->21 SeekKey   = Byte offset of record itself (consistency check) (64)
TKey::fSeekKey
//      22->25 SeekPdir  = Byte offset of parent directory record (0)
      TKey::fSeekPdir
//      26->26 lname     = Number of bytes in the class name (5)         TKey::fClassName
//      27->.. ClassName = Object Class Name ("TFile")
      TKey::fClassName
//       0->0  lname     = Number of bytes in the object name
      TNamed::fName
```

```
//       1->.. Name      = lName bytes with the name of the object <file name>
      TNamed::fName
//       0->0  lTitle    = Number of bytes in the object title
      TNamed::fTitle
//       1->.. Title     = lTitle bytes with the title of the object <file title>
TNamed::fTitle
//    --------DATA-----------------
//       0->0  lname     = Number of bytes in the TFile name
      TNamed::fName
//       1->.. Name      = lName bytes with the name of the TFile <file name>
      TNamed::fName
//       0->0  lTitle    = Number of bytes in the TFile title
      TNamed::fTitle
//       1->.. Title     = lTitle bytes with the title of the TFile <file title>
TNamed::fTitle
//       0->0  Modified  = True if directory has been modified
      TDirectory::fModified
//       1->1  Writeable = True if directory is writeable
      TDirectory::fWriteable
//       2->5  DatimeC   = Date and time when directory was created
      TDirectory::fDatimeC
//                       | (year-1995)<<26|month<<22|day<<17|hour<<12|minute<<6|second
//       6->9  DatimeM   = Date and time when directory was last modified
      TDirectory::fDatimeM

//                       | (year-1995)<<26|month<<22|day<<17|hour<<12|minute<<6|second
//      10->13 NbytesKeys= Number of bytes in the associated KeysList record
      TDirectory::fNbyteskeys
//      14->17 NbytesName= Number of bytes in TKey+TNamed at creation
      TDirectory::fNbytesName
//      18->21 SeekDir   = Byte offset of directory record in file (64)
      TDirectory::fSeekDir
//      22->25 SeekParent= Byte offset of parent directory record in file (0)
      TDirectory::fSeekParent
//      26->29 SeekKeys  = Byte offset of associated KeysList record in file
      TDirectory::fSeekKeys
```

Keys List Record Format

```
//  Format of KeysList record in release 3.02.06.  It is never compressed.
//  There is one KeysList record for the main (TFile) directory and one per non-empty
subdirectory.
//  It is probably not accessed by its key, but from its offset given in the directory
data.
//    ----------TKey---------------
//  byte 0->3  Nbytes    = Number of bytes in compressed record (TKey+data)
      TKey::fNbytes
//       4->5  Version   = TKey class version identifier                TKey::fVersion
//       6->9  ObjLen    = Number of bytes of uncompressed data         TKey::fObjLen
//      10->13 Datime    = Date and time when record was written to file
      TKey::fDatime
//                       | (year-1995)<<26|month<<22|day<<17|hour<<12|minute<<6|second
//      14->15 KeyLen    = Number of bytes in the key structure (TKey)
      TKey::fKeyLen
//      16->17 Cycle     = Cycle of key                                 TKey::fCycle
//      18->21 SeekKey   = Byte offset of record itself (consistency check)
      TKey::fSeekKey
//      22->25 SeekPdir  = Byte offset of parent directory record (directory)
      TKey::fSeekPdir
//      26->26 lname     = Number of bytes in the class name (5 or 10)
      TKey::fClassName
```

```
//      27->.. ClassName = Object Class Name ("TFile" or "TDirectory")
      TKey::fClassName
//       0->0  lname     = Number of bytes in the object name
      TNamed::fName
//       1->.. Name      = lName bytes with the name of the object <directory name>
TNamed::fName
//       0->0  lTitle    = Number of bytes in the object title
      TNamed::fTitle
//       1->.. Title     = lTitle bytes with the title of the object <directory title>
TNamed::fTitle
//    ----------DATA---------------
//       0->3  NKeys     = Number of keys in list (i.e. records in directory (non-
recursive))
//                       | Excluded:: The directory itself, KeysList, StreamerInfo, and
FreeSegments
//       4->.. TKey      = Sequentially for each record in directory,
//                       |  the entire TKey portion of each record is replicated.
//                       |  Note that SeekKey locates the record.
```

Format for free segments and gaps

```
//Format of FreeSegments record, release 3.02.06.  It is never compressed.
//It is probably not accessed by its key, but from its offset given in the file header.
//    ----------TKey---------------
//  byte 0->3  Nbytes    = Number of bytes in compressed record (TKey+data)
      TKey::fNbytes
//       4->5  Version   = TKey class version identifier                TKey::fVersion
//       6->9  ObjLen    = Number of bytes of uncompressed data
      TKey::fObjLen
//      10->13 Datime    = Date and time when record was written to file
      TKey::fDatime
//                       | (year-1995)<<26|month<<22|day<<17|hour<<12|minute<<6|second
//      14->15 KeyLen    = Number of bytes in key structure  (TKey)
      TKey::fKeyLen
//      16->17 Cycle     = Cycle of key                                TKey::fCycle
//      18->21 SeekKey   = Byte offset of record itself (consistency check)
      TKey::fSeekKey
//      22->25 SeekPdir  = Byte offset of parent directory record (TFile)
      TKey::fSeekPdir
//      26->26 lname     = Number of bytes in the class name (5)         TKey::fClassName
//      27->.. ClassName = Object Class Name ("TFile")
      TKey::fClassName
//       0->0  lname     = Number of bytes in the object name
      TNamed::fName
//       1->.. Name      = lName bytes with the name of the object <file name>
      TNamed::fName
//       0->0  lTitle    = Number of bytes in the object title
      TNamed::fTitle
//       1->.. Title     = lTitle bytes with the title of the object <file title>
TNamed::fTitle
//    ----------DATA---------------
//       0->1  Version   = TFree class version identifier
      TFree::Class_Version()
//       2->5  fFirst    = First free byte of first free segment         TFree::fFirst
//       6->9  fLast     = Byte after last free byte of first free segment
      TFree::fLast
//       ....  Sequentially, Version, fFirst and fLast of additional free segments.
//       ....  There is always one free segment beginning at file end and ending before
2000000000.

//  A gap (free segment in middle of file) has the following format.
//    -----------------------
```

```
//  byte 0->3  Nbytes    = Negative of number of bytes in gap
//       4->.. irrelevant
```

TBuffer

TMessage

Object I/O

Format for data objects using default streamers

```
//  Release 3.02.06
//    Here is the format of a class object in DATA that uses the default streamer.
//    Objects of many classes with custom streamers can have very similar formats.
//----------------
//       0->3  ByteCount = Number of remaining bytes in object (uncompressed)
//                 | OR'd with kByteCountMask (0x40000000)
//     4->.. ClassInfo = Information about class of object
//                 | If this is the first occurrence of an object of this class in the
      record
//                       |  4->7  -1        = New class tag (constant kNewClassTag =
0xffffffff)
//                       |  8->.. Classname = Object Class Name (null terminated string)
//                 | Otherwise
//                 |  4->7 clIdx      = Byte offset of new class tag in record, plus 2.
//                 | OR'd with kClassMask (0x80000000)
//       0->3  ByteCount = Number of remaining bytes in object (uncompressed)
//                 | OR'd with kByteCountMask (0x40000000)
//     4->5  Version   = Version of Class
//
//    The rest consists of objects of base classes and persistent non-static data
      members.
//    Data members marked as transient are not stored.
//
//     6->.. Sequentially, Objects of each base class from which this class is derived
//          (rarely more than one)
//     0->.. Sequentially, Objects of all non-static persistent data members.
//
//    Class objects are broken down recursively as above.
//
//      Built in types are stored as follows:
//    1 Byte: char, unsigned char
//      2 Bytes: short, unsigned short
//      4 Bytes: int, unsigned int, float
//      8 Bytes: long, unsigned long, double
//    Note that a long (signed or unsigned) is stored as 8 bytes even if it is only four
      bytes
//     in memory.  In that case, it is filled with leading zeros (or ones, for a negative
      value).
//
```

TObject

```
//    Here is the format of the DATA for a TObject object in Release 3.02.06.
//--------
//     0->1  Version   = Version of TObject Class
//     2->5  fUniqueID = Unique ID of object.  Currently, unless this object is or was
//                       | referenced by a TRef or TRefArray, or is itself a TRef or
TRefArray,
```

```
//                        | this field is not used by ROOT.
//     6->9  fBits     = A 32 bit mask containing status bits for the object.
//                        | The bits relevant to ROOTIO are:
//                  | 0x00000001 - if object in a list can be deleted.
//                  | 0x00000008 - if other objects may need to be deleted when this one
is.
//                  | 0x00000010 - if object is referenced by pointer to persistent
      object.
//                  | 0x00002000 - if object ctor succeeded but object shouldn't be used
//                  | 0x01000000 - if object is on Heap.
//                  | 0x02000000 - if object has not been deleted.
//    The "pidf" field below is present only if this TObject object (or an object
inheriting
//      from it) is referenced by a pointer to persistent object.
//    10->11 pidf  = An identifier of the TProcessID record for the process that wrote
      the
//                        | object. This identifier is an unsigned short.  The relevant
record
//                        | has a name that is the string "ProcessID" concatenated with
the ASCII
//                        | decimal representation of "pidf" (no leading zeros).  0 is a
valid pidf.
//-------
//    No object in the StreamerInfo record will be a reference or referenced, and all
objects
//      are on the heap.  So, for each occurrence in the StreamerInfo record, fUniqueID
will be 0,
//      fBits will be 0x03000000, and pidf will be absent.
```

TRef

```
//    Here is the format of the DATA for a TRef object in Release 3.02.06.
//--------
//     0->1  Version   = Version of TObject Class (base class of TRef)
//     2->5  fUniqueID = Unique ID of referenced object.  Typically, every referenced
//                        | object has an ID that is a positive integer set to a counter
//                  | of the number of referenced objects in the file, beginning at 1.
//                        | fUniqueID in the TRef object matches fUniqueID in the
//                        | referenced object.
//     6->9  fBits     = A 32 bit mask containing status bits for the TRef object.
//                        | The bits relevant to ROOTIO are:
//                  | 0x00000008 - Other objects may need to be deleted when this one is.
//                  | 0x00000010 - Object is referenced by pointer to persistent object.
//                  | 0x01000000 - Object is on Heap.
//                  | 0x02000000 - Object has not been deleted.
//    10->11 pidf  = An identifier of the TProcessID record for the process that wrote
the
//                        | referenced object. This identifier is an unsigned short.  The
relevant
//                        | record has a name that is the string "ProcessID" concatenated
with the
//                        | ASCII decimal representation of "pidf" (no leading zeros).
//                        | 0 is a valid pidf.
//-------
```

TRefArray

```
//    Here is the format of the DATA for a TRefArray object in Release 3.02.06.
//--------
//       0->3  ByteCount = Number of remaining bytes in TRefArray object (uncompressed)
//                        |   OR'd with kByteCountMask (0x40000000)
//       4->5  Version   = Version of TRefArray Class
```

```
//       6->15           = TObject object (Base class of TRefArray) (see tobject.txt).
//                       |   Will be two bytes longer (6->17) if TRefArray object is
//                       |   itself referenced (unlikely).
//      16->.. fName     = Number of bytes in name of TRefArray object, followed by the
//                       |   name itself.  (TCollection::fName). Currently, TRefArrays
//                       |   are not named, so this is a single byte containing 0.
//       0->3  nObjects  | Number of object references (fUIDs) in this TRefArray.
//       4->7  fLowerBound= Lower bound of array.  Typically 0.
//     8->9  pidf  = An identifier of the TProcessID record for the process that wrote
the
//                       | referenced objects. This identifier is an unsigned short.  The
relevant
//                       | record has a name that is the string "ProcessID" concatenated
with the
//                       | ASCII decimal representation of "pidf" (no leading zeros).
//                       | 0 is a valid pidf.
//      10->.. fUIDs     = Sequentially, object Unique ID's.
//                       | Each Unique ID is a four byte unsigned integer.
//                       | If non-zero, it matches the Unique ID in the referenced
//                       | object.  If zero, it is an unused element in the array.
//                       | The fUIDs are written out only up to the last used element,
//                       | so the last fUID will always be non-zero.
```

StreamerInfo (for self describing object format)

```
//  Format of StreamerInfo record in release 3.02.06.
//  It is probably not accessed by its key, but from its offset given in the file header.
//  The StreamerInfo record DATA consists of a TList (list) object containing elements
//  of class TStreamerInfo.
//    ----------TKey-(never compressed)----------------------
//  byte 0->3  Nbytes    = Number of bytes in compressed record (TKey+data)
      TKey::fNbytes
//       4->5  Version   = TKey class version identifier                TKey::fVersion
//       6->9  ObjLen    = Number of bytes of uncompressed data
      TKey::fObjLen
//      10->13 Datime    = Date and time when record was written to file
      TKey::fDatime
//                | (year-1995)<<26|month<<22|day<<17|hour<<12|minute<<6|second
//      14->15 KeyLen    = Number of bytes in key structure (TKey) (64)
      TKey::fKeyLen
//      16->17 Cycle     = Cycle of key                                TKey::fCycle
//      18->21 SeekKey   = Byte offset of record itself (consistency check)
      TKey::fSeekKey
//      22->25 SeekPdir  = Byte offset of parent directory record (TFile)
      TKey::fSeekPdir
//      26->26 lname     = Number of bytes in the class name (5)         TKey::fClassName
//      27->31 ClassName = Object Class Name ("TList")
      TKey::fClassName
//      32->32 lname     = Number of bytes in the object name (12)
      TNamed::fName
//      33->44 Name      = lName bytes with the name of the object ("StreamerInfo")
TNamed::fName
//      45->45 lTitle    = Number of bytes in the object title (18)
      TNamed::fTitle
//      46->63 Title     = lTitle bytes with the title of the object
      TNamed::fTitle
//                | ("Doubly linked list")
//    ----------TList-(always compressed at level 1 (even if compression level 0))----
//    The DATA is a TList collection object containing TStreamerInfo objects.
//    Below is the format of this TList data.
//
//    Here is the format of a TList object in Release 3.02.06.
```

```
//    Comments and offsets refer specifically to its use in the StreamerInfo record.
//--------
//      0->3  ByteCount = Number of remaining bytes in TList object (uncompressed)
//                |   OR'd with kByteCountMask (0x40000000)
//    4->5  Version   = Version of TList Class
//    6->15           = TObject object (a base class of TList) (see tobject.txt).
//                |   Objects in StreamerInfo record are not referenced.
//                |   Would be two bytes longer (6->17) if object were referenced.
//     16->16 fName     = Number of bytes in name of TList object, followed by the
//                |   name itself.  (TCollection::fName).  The TList object in
//                |   StreamerInfo record is unnamed, so byte contains 0.
//     17->20 nObjects  = Number of objects in list.
//     21->.. objects   = Sequentially, TStreamerInfo Objects in the list.
//                | In the StreamerInfo record, the objects in the list are
//                |   TStreamerInfo objects.  There will be one TStreamerInfo
//                |   object for every class used in data records other than
//                |   core records and the the StreamerInfo record itself.
//-------
//    Here is the format of a TStreamerInfo object in Release 3.02.06.
//    Note: Although TStreamerInfo does not use the default streamer, it has the same
//          format as if it did.  (compare with dobject.txt)
//      0->3  ByteCount = Number of remaining bytes in TStreamerInfo object
(uncompressed)
//                |   OR'd with kByteCountMask (0x40000000)
//    4->.. ClassInfo = Information about TStreamerInfo class
//                | If this is the first occurrence of a TStreamerInfo object in the
record
//                |  4->7  -1        = New class tag (constant kNewClassTag = 0xffffffff)
//                |  8->21 Classname = Object Class Name "TStreamerInfo" (null
terminated)
//                | Otherwise
//                |  4->7 clIdx      = Byte offset of new class tag in record, plus 2.
//                | OR'd with kClassMask (0x80000000)
//    0->3  ByteCount = Number of remaining bytes in TStreamerInfo object (uncompressed)
//                |   OR'd with kByteCountMask (0x40000000)
//    4->5  Version   = Version of TStreamerInfo Class
// -Begin TNamed object (Base class of TStreamerInfo)
//    6->9  ByteCount = Number of remaining bytes in TNamed object
//                |   OR'd with kByteCountMask (0x40000000)
//     10->11 Version   = Version of TNamed Class
//     12->21           = TObject object (Base class of TNamed) (see tobject.txt).
//                |   Objects in StreamerInfo record are not referenced.
//                |   Would be two bytes longer (12->23) if object were referenced.
//     22->.. fName     = Number of bytes in name of class that this TStreamerInfo object
//                |   describes, followed by the class name itself.  (TNamed::fName).
//    0->.. fTitle    = Number of bytes in title of class that this TStreamerInfo object
//                |   describes, followed by the class title itself.  (TNamed::fTitle).
//                |  (Class title may be zero length)
// -End TNamed object
//    0->3  fCheckSum = Check sum for class that this TStreamerInfo object describes.
//                |  This checksum is over all base classes and all persistent
//                |  non-static data members.  It is computed by TClass::GetCheckSum().
//                      |  (TStreamerInfo::fCheckSum)
//    4->7  fClassVersion = Version of class that this TStreamerInfo object describes.
//                      |   (TStreamerInfo::fClassVersion)
// -Begin TObjArray object (Data member of TStreamerInfo)
//      0->3  ByteCount = Number of remaining bytes in TObjArray object (uncompressed)
//                |   OR'd with kByteCountMask (0x40000000)
//    4->.. ClassInfo = Information about TObjArray class
//                | If this is the first occurrence of a TObjArray object in the record
//                |  4->7  -1        = New class tag (constant kNewClassTag = 0xffffffff)
//                |  8->17 Classname = Object Class Name "TObjArray" (null terminated)
//                | Otherwise
```

```
//                |  4->7 clIdx      = Byte offset of new class tag in record, plus 2.
//                | OR'd with kClassMask (0x80000000)
//      0->3  ByteCount = Number of remaining bytes in TObjArray object (uncompressed)
//                |   OR'd with kByteCountMask (0x40000000)
//    4->5  Version   = Version of TObjArray Class
//    6->15           = TObject object (a base class of TObjArray) (see tobject.txt).
//                |   Objects in StreamerInfo record are not referenced.
//                |   Would be two bytes longer (6->17) if object were referenced.
//     16->16 fName     = Number of bytes in name of TObjArray object, followed by the
//                |   name itself.  (TCollection::fName).  TObjArray objects in
//                |   StreamerInfo record are unnamed, so byte contains 0.
//     17->20 nObjects  = Number of objects (derived from TStreamerElement) in array.
//     21->24 fLowerBound = Lower bound of array.  Will always be 0 in StreamerInfo
record.
//     25->.. objects   = Sequentially, TStreamerElement objects in the array.
//                | In a TStreamerInfo object, the objects in the TObjArray are
//                |   of various types (described below), all of which inherit
//                |   directly from TStreamerElement objects.  There will be one
//                |   such object for every base class of the class that the
//                |   TStreamerInfo object describes, and also one such object for
//                      |   each persistent non-static data member of the class that the
//                      |   TStreamerInfo object describes.
// -End TObjArray object and TStreamerInfo object
//-------
//  The objects stored in the TObjectArray in TStreamerInfo are of various classes, each
of
//     which inherits directly from the TStreamerElement class.  The possible classes
(which
//     we refer to collectively as TStreamer<XXX>) are:
//
//  TStreamerBase:          Used for a base class.  All others below used for data
members.
//  TStreamerBasicType:     For a basic type
//  TStreamerString:        For type TString
//  TStreamerBasicPointer:  For pointer to array of basic types
//  TStreamerObject:        For an object derived from TObject

//  TStreamerObjectPointer: For pointer to an object derived from TObject
//  TStreamerLoop:          For pointer to an array of objects
//  TStreamerObjectAny:     For an object not derived from TObject
//  TStreamerSTL:           For an STL container (not yet used??)
//  TStreamerSTLString:     For an STL string (not yet used??)
//-------
//    Here is the format of a TStreamer<XXX> object in Release 3.02.06.
//    In description below,
//    0->3  ByteCount = Number of remaining bytes in TStreamer<XXX> object (uncompressed)
//                |   OR'd with kByteCountMask (0x40000000)
//    4->.. ClassInfo = Information about the specific TStreamer<XXX> class
//                | If this is the first occurrence of a TStreamerXXX object in the
record
//                |  4->7  -1        = New class tag (constant kNewClassTag = 0xffffffff)
//                |  8->.. Classname = Object Class Name "TStreamer<XXX>" (null
terminated)
//                | Otherwise
//                |  4->7 clIdx      = Byte offset of new class tag in record, plus 2.
//                | OR'd with kClassMask (0x80000000)
//    0->3  ByteCount = Number of remaining bytes in TStreamer<XXX> object (uncompressed)
//                |   OR'd with kByteCountMask (0x40000000)
//    4->5  Version   = Version of TStreamer<XXX> Class
// -Begin TStreamerElement object (Base class of TStreamerXXX)
//    0->3  ByteCount = Number of remaining bytes in TStreamerElement object
(uncompressed)
//                |   OR'd with kByteCountMask (0x40000000)
```

```
//    4->5  Version   = Version of TStreamerElement Class
// -Begin TNamed object (Base class of TStreamerElement)
//    6->9  ByteCount = Number of remaining bytes in TNamed object
//                  |   OR'd with kByteCountMask (0x40000000)
//     10->11 Version   = Version of TNamed Class
//     12->21           = TObject object (Base class of TNamed) (see tobject.txt).
//                  |   Objects in StreamerInfo record are not referenced.
//                  |   Would be two bytes longer (12->23) if object were referenced.
//     22->.. fName     = Number of bytes in class name of base class or member name of
//                        | data member that this TStreamerElement object describes,
//                        | followed by the name itself. (TNamed::fName).
//      0->.. fTitle    = Number of bytes in title of base class or data member that this
//                        | TStreamerElement object describes, followed by the title
itself.
//                        |  (TNamed::fTitle).
// -End TNamed object
//    0->3  fType     = Type of data described by this TStreamerElement.
//                        |   (TStreamerElement::fType)
//                        |   Built in types:
//                        |   1:char, 2:short, 3:int, 4:long, 5:float, 8:double
//                        |   11, 12, 13, 14:unsigned char, short, int, long respectively
//                        |   6: an array dimension (counter)
//                        |   15: bit mask (used for fBits field)
//                        |
//                        |   Pointers to built in types:
//                        |   40 + fType of built in type (e.g. 43: pointer to int)
//                        |
//                  |   Objects:
//                  |   65:TString, 66:TObject, 67:TNamed
//                  |   0: base class (other than TObject or TNamed)
//                  |   61: object data member derived from TObject (other than TObject or
TNamed)
//                  |   62: object data member not derived from TObject
//                  |   63: pointer to object derived from TObject (pointer can't be null)
//                  |   64: pointer to object derived from TObject (pointer may be null)
//                  |   501: pointer to an array of objects
//                        |   500: an STL string or container
//                        |
//                  |   Arrays:
//                        |   20 + fType of array element (e.g. 23: array of int)
//                        |
//    4->7  fSize     = Size of built in type or of pointer to built in type. 0
otherwise.
//                        |  (TStreamerElement::fSize).
//    8->11 fArrayLength = Size of array (0 if not array)
//                        |  (TStreamerElement::fArrayLength).
//     12->15 fArrayDim = Number of dimensions of array (0 if not an array)
//                        |  (TStreamerElement::fArrayDim).
//     16->35 fMaxIndex = Five integers giving the array dimensions (0 if not applicable)
//                        |  (TStreamerElement::fMaxIndex).
//     36->.. fTypeName = Number of bytes in name of the data type of the data member
that
//                        |  the TStreamerElement object describes, followed by the name
//                        |  itself.  If this TStreamerElement object defines a base class
//                        |  rather than a data member, the name used is 'BASE'.
//                        |  (TStreamerElement::fTypeName).
// -End TStreamerElement object
//     The remaining data is specific to the type of TStreamer<XXX> class.
//      For TStreamerInfoBase:
//      0->3  fBaseVersion   = Version of base class that this TStreamerElement
describes.
//      For TStreamerBasicType:
//            No specific data
```

```
//      For TStreamerString:
//            No specific data
//      For TStreamerBasicPointer:
//      0->3  fCountVersion = Version of class with the count (array dimension)
//      4->.. fCountName= Number of bytes in the name of the data member holding
//                      | the count, followed by the name itself.
//      0->.. fCountName= Number of bytes in the name of the class holding the
//                      | count, followed by the name itself.
//      For TStreamerObject:
//            No specific data
//      For TStreamerObjectPointer:
//            No specific data
//      For TStreamerLoop:
//      0->3  fCountVersion = Version of class with the count (array dimension)
//      4->.. fCountName= Number of bytes in the name of the data member holding
//                      | the count, followed by the name itself.
//      0->.. fCountClass= Number of bytes in the name of the class holding the
//                      | count, followed by the name itself.
//      For TStreamerObjectAny:
//            No specific data
//      For TStreamerSTL:
//      0->3  fSTLtype  = Type of STL container:
//                      | 1:vector, 2:list, 3:deque, 4:map, 5:set, 6:multimap, 7:multiset
//      4->7  fCType    = Type contained in STL container:
//                      | Same values as for fType above, with one addition: 365:STL
string
//      For TStreamerSTLString:
//            No specific data
```

TProcessID (for referenced objects)

```
//    Format of TProcessID record in release 3.02.06.
//    Will be present if there are any referenced objects.
//    ----------TKey--------------
//   byte 0->3  Nbytes   = Number of bytes in compressed record (Tkey+data)
      TKey::fNbytes
//        4->5  Version  = TKey class version identifier                TKey::fVersion
//        6->9  ObjLen   = Number of bytes of uncompressed data
      TKey::fObjLen
//       10->13 Datime   = Date and time when record was written to file
      TKey::fDatime
//                      | (year-1995)<<26|month<<22|day<<17|hour<<12|minute<<6|second
//       14->15 KeyLen   = Number of bytes in key structure (TKey)
      TKey::fKeyLen
//       16->17 Cycle    = Cycle of key                                TKey::fCycle
//       18->21 SeekKey  = Byte offset of record itself (consistency check)
      TKey::fSeekKey
//       22->25 SeekPdir = Byte offset of parent directory record      TKey::fSeekPdir
//       26->26 lname    = Number of bytes in the class name (10)      TKey::fClassName
//       27->36 ClassName= Object Class Name ("TProcessID")            TKey::fClassName
//       37->37 lname    = Number of bytes in the object name
      TNamed::fName
//       38->.. Name     = lName bytes with the name of the object
      TNamed::fName
//                      | (e.g. "ProcessID0")
//        0->0  lTitle   = Number of bytes in the object title
      TNamed::fTitle
//        1->.. Title    = lTitle bytes with the title of the object
      TNamed::fTitle
//                      | (Identifies processor, time stamp, etc.)
//                | See detailed explanation below.
//    ----------DATA-------------
```

```
//        0->3 ByteCount = Number of remaining bytes in TProcessID object (uncompressed)
//                       |   OR'd with kByteCountMask (0x40000000)
//      4->5 Version   = Version of TProcessID Class
// -Begin TNamed object (Base class of TProcessID)
//        6->9 ByteCount = Number of remaining bytes in TNamed object (uncompressed)
//                       |   OR'd with kByteCountMask (0x40000000)
//       10->11 Version  = Version of TNamed Class
//       12->21          = TObject object (Base class of TNamed) (see tobject.txt).
//                       |   The TProcessID object is not itself referenced.
//       22->22 lname    = Number of bytes in the object name
      TNamed::fName
//       23->.. Name     = lName bytes with the name of the object
      TNamed::fName
//                       | The name will be "ProcessID" concatenated with
//                       | a decimal integer, or "pidf".
//        0->0  lTitle   = Number of bytes in the object title
      TNamed::fTitle
//        1->.. Title    = lTitle bytes with the title of the object
      TNamed::fTitle
//                       | (Identifies processor, time stamp, etc.)
//                 | See detailed explanation below.
// -End TNamed object
//
//     ----------Explanation of the title of a TProcessID object-----------------
//    The title of a TProcessID object is a globally unique identifier of the
//    ROOTIO process that created it.  It is derived from the following quantities.
//
//    1) The creation time ("fTime) of the TProcessID record.  This is a 60 bit time
//    in 100ns ticks since Oct. 15, 1582.
//
//    2) A 16 bit random unsigned integer ("clockeq") generated from a seed that is the
//      job's process ID.  The highest two bits are not used.
//
//    3) A six byte unsigned quantity ("fNode") identifying the machine.  If the machine
has a
//    valid network address, the first four bytes are set to that address, and the last
two bytes
//    are stuffed with 0xbe and 0xef respectively.  Otherwise a six byte quantity is
generated
//    from the time and random machine statistics. In this case, the high order bit of
the
//    first byte is set to 1, to distinguish it from a network ID, where the bytes can be
//    no larger than 255.
//
//    We the define the following quantities (class TUUID):
        UInt_t    fTimeLow;               // 60 bit time, lowest 32 bits
        UShort_t  fTimeMid;               // 60 bit time, middle 16 bits
        UShort_t  fTimeHiAndVersion;      // 60 bit time, highest 12 time bits (low 12
bits)
                                          // + 4 UUID version bits (high 4 bits)
                                    // version is 1 if machine has valid network address
                                    // and 3 otherwise.
        UChar_t   fClockSeqHiAndReserved; // high 6 clockseq bits (low 6 bits)
                                          // + 2 high bits reserved (currently set to
binary 10)
        UChar_t   fClockSeqLow;           // low 8 clockseq bits
        UChar_t   fNode[6];               // 6 node (machine) id bytes

//    Then the following sprintf() call defines the format of the title string:
      sprintf(Title, "%08x-%04x-%04x-%02x%02x-%02x%02x%02x%02x%02x%02x",
           fTimeLow, fTimeMid, fTimeHiAndVersion, fClockSeqHiAndReserved,
           fClockSeqLow, fNode[0], fNode[1], fNode[2], fNode[3], fNode[4],
           fNode[5]);
```

```
//    Since the title written to disk is preceded by its byte count, the delimiting null
is not written.
```

Optimization Layer

TTree

```
Here is the streamer information for TTree related classes in release 3.02.06:
(For the explanation of the meaning of the type, see "fType" in "streamerinfo.txt".)
------------------------------------------------------
StreamerInfo for class: TTree, version=6
  BASE          TNamed          offset=  0 type=67 The basis for a named object (name,
title)
  BASE          TAttLine        offset=  0 type= 0 Line attributes
  BASE          TAttFill        offset=  0 type= 0 Fill area attributes
  BASE          TAttMarker      offset=  0 type= 0 Marker attributes
  Stat_t        fEntries        offset=  0 type= 8 Number of entries
  Stat_t        fTotBytes       offset=  0 type= 8 Total number of bytes in all branches
before compression
  Stat_t        fZipBytes       offset=  0 type= 8 Total number of bytes in all branches
after compression
  Stat_t        fSavedBytes     offset=  0 type= 8 Number of autosaved bytes
  Int_t         fTimerInterval  offset=  0 type= 3 Timer interval in milliseconds
  Int_t         fScanField      offset=  0 type= 3 Number of runs before prompting in
Scan
  Int_t         fUpdate         offset=  0 type= 3 Update frequency for EntryLoop
  Int_t         fMaxEntryLoop   offset=  0 type= 3 Maximum number of entries to process
  Int_t         fMaxVirtualSize offset=  0 type= 3 Maximum total size of buffers kept in
memory
  Int_t         fAutoSave       offset=  0 type= 3 Autosave tree when fAutoSave bytes
produced
  Int_t         fEstimate       offset=  0 type= 3 Number of entries to estimate
histogram limits
  TObjArray     fBranches       offset=  0 type=61 List of Branches
  TObjArray     fLeaves         offset=  0 type=61 Direct pointers to individual branch
leaves
  TArrayD       fIndexValues    offset=  0 type=62 Sorted index values
  TArrayI       fIndex          offset=  0 type=62 Index of sorted values
  TList*        fFriends        offset=  0 type=64 pointer to list of friend elements

StreamerInfo for class: TAttLine, version=1
  Color_t       fLineColor      offset=  0 type= 2 line color
  Style_t       fLineStyle      offset=  0 type= 2 line style
  Width_t       fLineWidth      offset=  0 type= 2 line width

StreamerInfo for class: TAttFill, version=1
  Color_t       fFillColor      offset=  0 type= 2 fill area color
  Style_t       fFillStyle      offset=  0 type= 2 fill area style

StreamerInfo for class: TAttMarker, version=1
  Color_t       fMarkerColor    offset=  0 type= 2 Marker color index
  Style_t       fMarkerStyle    offset=  0 type= 2 Marker style
  Size_t        fMarkerSize     offset=  0 type= 5 Marker size

StreamerInfo for class: TBranch, version=7
  BASE          TNamed          offset=  0 type=67 The basis for a named object (name,
title)
  Int_t         fCompress       offset=  0 type= 3 (=1 branch is compressed, 0 otherwise)
  Int_t         fBasketSize     offset=  0 type= 3 Initial Size of  Basket Buffer
```

```
  Int_t         fEntryOffsetLen offset=  0 type= 3 Initial Length of fEntryOffset table
in the basket buffers
  Int_t         fWriteBasket    offset=  0 type= 3 Last basket number written
  Int_t         fEntryNumber    offset=  0 type= 3 Current entry number (last one filled
in this branch)
  Int_t         fOffset         offset=  0 type= 3 Offset of this branch
  Int_t         fMaxBaskets     offset=  0 type= 6 Maximum number of Baskets so far
  Int_t         fSplitLevel     offset=  0 type= 3 Branch split level
  Stat_t        fEntries        offset=  0 type= 8 Number of entries
  Stat_t        fTotBytes       offset=  0 type= 8 Total number of bytes in all leaves
before compression
  Stat_t        fZipBytes       offset=  0 type= 8 Total number of bytes in all leaves
after compression
  TObjArray     fBranches       offset=  0 type=61 -> List of Branches of this branch
  TObjArray     fLeaves         offset=  0 type=61 -> List of leaves of this branch
  TObjArray     fBaskets        offset=  0 type=61 -> List of baskets of this branch
  Int_t*        fBasketBytes    offset=  0 type=43 [fMaxBaskets] Length of baskets on
file
  Int_t*        fBasketEntry    offset=  0 type=43 [fMaxBaskets] Table of first entry in
eack basket
  Seek_t*       fBasketSeek     offset=  0 type=43 [fMaxBaskets] Addresses of baskets on
file
  TString       fFileName       offset=  0 type=65 Name of file where buffers are stored
("" if in same file as Tree header)

StreamerInfo for class: TBranchElement, version=7
  BASE          TBranch         offset=  0 type= 0 Branch descriptor
  TString       fClassName      offset=  0 type=65 Class name of referenced object
  TString       fParentName     offset=  0 type=65 Name of parent class
  TString       fClonesName     offset=  0 type=65 Name of class in TClonesArray (if any)
  Int_t         fClassVersion   offset=  0 type= 3 Version number of class
  Int_t         fID             offset=  0 type= 3 element serial number in fInfo
  Int_t         fType           offset=  0 type= 3 branch type
  Int_t         fStreamerType   offset=  0 type= 3 branch streamer type
  Int_t         fMaximum        offset=  0 type= 3 Maximum entries for a TClonesArray or
variable array
  TBranchElement*fBranchCount    offset=  0 type=64 pointer to primary branchcount branch
  TBranchElement*fBranchCount2   offset=  0 type=64 pointer to secondary branchcount
branch

StreamerInfo for class: TLeaf, version=2
  BASE          TNamed          offset=  0 type=67 The basis for a named object (name,
title)
  Int_t         fLen            offset=  0 type= 3 Number of fixed length elements
  Int_t         fLenType        offset=  0 type= 3 Number of bytes for this data type
  Int_t         fOffset         offset=  0 type= 3 Offset in ClonesArray object (if one)
  Bool_t        fIsRange        offset=  0 type=11 (=kTRUE if leaf has a range, kFALSE
otherwise)
  Bool_t        fIsUnsigned     offset=  0 type=11 (=kTRUE if unsigned, kFALSE otherwise)
  TLeaf*        fLeafCount      offset=  0 type=64 Pointer to Leaf count if variable
length

StreamerInfo for class: TLeafElement, version=1
  BASE          TLeaf           offset=  0 type= 0 Leaf: description of a Branch data
type
  Int_t         fID             offset=  0 type= 3 element serial number in fInfo
  Int_t         fType           offset=  0 type= 3 leaf type
```

TClonesArray

```
// -Here is the format (release 3.02.06)  of the DATA for a TClonesArray object in a
ROOTIO file.
```

```
//      0->3  ByteCount = Number of remaining bytes in TClonesArray object (uncompressed)
//                |   OR'd with kByteCountMask (0x40000000)
//      4->.. ClassInfo = Information about TClonesArray class
//                | If this is the first occurrence of a TClonesArray object in the
record
//                |  4->7  -1        = New class tag (constant kNewClassTag = 0xffffffff)
//                |  8->17 Classname = Object Class Name "TClonesArray" (null terminated)
//                | Otherwise
//                |  4->7 clIdx      = Byte offset of new class tag in record, plus 2.
//                | OR'd with kClassMask (0x80000000)
//      0->3  ByteCount = Number of remaining bytes in TClonesArray object (uncompressed)
//                |   OR'd with kByteCountMask (0x40000000)
//    4->5  Version   = Version of TClonesArray Class
//    6->15           = TObject object (a base class of TClonesArray) (see tobject.txt).
//                |   Would be two bytes longer (6->17) if object were referenced.
//     16->.. fName     = Number of bytes in name of TClonesArray object, followed by the
//                |   name itself.  (TCollection::fName).  This name will be the
//                |   class name of the cloned object, appended with an 's'
//                      |   (e.g. "TXxxs")
//      0->..           = Number of bytes in name and version of the cloned class,
followed
//                      |   by the name and version themselves (e.g. "TXxx;1")
//      0->3  nObjects  = Number of objects in clones array.
//      4->7  fLowerBound= Lower bound of clones array.
//      8->.. objects   = Sequentially, objects in the clones array.  However, the data
//                      |   ordering depends on whether or not kBypassStreamer (0x1000)
is
//                |   set in TObject::fBits.   By default, it is set.  If it is not set,
//                |   the objects are streamed sequentially using the streamer of the
//                |   cloned class (e.g. TXxx::Streamer()).
//                      |
//                      |   If it is set, the cloned class is split into its base classes
and
//                      |   persistent data members, and those streamers are used.  So,
if the
//                      |   base classes and persistent data members of class TXxx are
TXxxbase,
//                      |   TXxxdata0, TXxxdata1, etc.,  all the TXxxbase data from the
entire
//                      |   clones array is streamed first, followed by all the TXxxdata0
data,
//                |   etc.  This breakdown is not recursive, in that the member objects
//                      |   are not again split.
// -End TClonesArray object
```

Global Setup