ROOT I/O Improvements

Non-intrusive I/O

Class Template

Namespace and Nested Class

Table Of Content

- Motivations
- Difficulties
- New features
- Limitations
- How to …
- Implementations
- What’s next?

Motivations

- Being able to save in a ROOT file objects from library that you can NOT modify at all.
- Being able to easily save objects that do not inherit from TObject.
- Being able to handle more complex templates and namespaces.
- Streamline *ClassDef/ClassImp* family of macros.
- Follow standard C++ rules

Difficulties

- How to recognize the class type
  - TObject or ClassDef:

    myobject->IsA();
  - New implementation:

    gROOT->GetClass(typeid(myobject));
- How to discover the address of protected and private data members
  - 'Trick' *#define private public*
  - Shadow classes

New Features

- ‘;’ after ClassDef
- Saving non-instrumented Classes
- Unlimited number of template parameters
- Replace definition of custom *ClassDef/ClassImp* macros by overloading of *ROOT::DefineBehavior*

Limitations

- ClassImp now declares a variable whose uniqueness is only due to the line number.
  - To work around the problem use:

    ClassImpUnique
- On windows, can not handle (yet) protected/private nested classes

How To: Non-intrusive I/O

- Create linkdef with

  #pragma link C++ class LibClass+;
- Possibly add a source file containing

  RootClassVersion(ClassName,3)
- Generate Dictionary
- Use in other classes

How To: Templates

- Replacing ClassDefT2

  ```
  typedef MyPairTemplate<int, double> type;
  ClassDef(type,2)
  ```

- Replacing ClassImpT2

  ```
  typedef  MyPairTemplate<int, double> type001;
  ClassImp( type001 )
  ```

- Global ClassImp

  ```
  templateClassImp(myTemplate)
  ```

- To be implemented:

  ```
  template <...template param> GetClassVersion(MyTemplate<...> *) {
     return VersionNumber; }
  ```

How To: Namespace and Nested Classes

- Now allow infinite nesting of namespace and classes
- ClassImp must be used outside namespace and be passed the fully qualified class name.

How To: ROOT::DefineBehavior

- Interface for objects used during the *TClass* initialization process:

```
class TInitBehavior
```

- Created by call to *DefineBehavior:*

```
const TInitBehavior *DefineBehavior(
    void * /*parent_type*/, void * /*actual_type*/)
```

- Should be overloaded to return the proper init object;

```
const TQObjectInitBehavior *DefineBehavior(
  TQObject*, TQObject*);
template <RootClass>
  const ROOT::TTableInitBehavior< RootClass> *DefineBehavior(
  TTable*, RootClass*);
```

Implementation

- Table of *typeid*'s
  - In *TClassTable* for the *ClassRec_t* objects
  - In TROOT for *TClass* objects
- Shadow class
  - Duplicate the class data member and existence (or not) of virtual table
  - (will) duplicate nested *protected/private* types

Implementation

- Bootstrap class information objects
  - *TGenericClassInfo*
  - Contains the information formally defined in *ClassDef/ClassImp* and generated methods
  - Acts as a place holder for the class information, (declaration file and line, version number)
  - Creates the *TClass* object when needed

    We can not use a *TClass* object because some implementation requires a different class (for example *TQClass*)

Implementation

- Bootstrap functions
  - Used to create a *TGenericClassInfo* object and insure that it is both unique and always available when needed.
- Bootstrap variables
  - Used to provoke the execution of customization of the *TGenericClassInfo*. Used in *RootClassVersion* and *ClassImp*.

Implementation

- Default operators:

```
template <class Tmpl> TBuffer &operator>>(TBuffer &buf,
    Tmpl *&obj);
template <class Tmpl> TBuffer &operator<<(TBuffer &buf,
    Tmpl *&hobj);
```

- Overload:

```
#if defined R__TEMPLATE_OVERLOAD_BUG
template <>
#endif
inline TBuffer &operator<<(TBuffer &buf,
    const TArrayI *obj)
```

Possible additional extensions

- Add proper interface to *TDirectory* to save/retrieve object which are NOT *TObjects*
- Allowing *custom streamers* for non-instrumented classes
- Registering *more than one* implementation file
- Replace Shadow classes by compiler and platform dependent offset calculation.