Extension of ROOT I/O customization framework

Anna Smagina

Mentor: Philippe Canal

1 September 2015

Reminder about the Evolution Schema

Evolution Schema allows to support objects of old versions classes without original compiled code.

Provides support of changed class definition by applying **the customization rules**.

Example of the rule:

```
♯ pragma read sourceClass=’’oldname’’
version="[1-]"checksum=‘‘[12345,23456]’’
source=’’type1 var; type2 var2;"
targetClass=‘‘newname’’ target=’’var3"
include=‘‘<cmath> <myhelper>’’
code=’’{ ... ‘code calculating var3 from var1 and var2’ ... }"
```

Reminder about the task

- **Implement support for JIT-compilation of I/O rules.**
  With CINT (and also with Cling) rules are written into dictionaries and compiled as a part of the user shared library. But with Cling it is a possible to operate with rules directly.
- **Add support for I/O rules for nested objects.**
  The same version of a containing class can hold several versions of the nested object's class.

Support for JIT-compilation

What was done:

- enabled reading of rules from file and check on target members,
- introduce a wrapper-function for the rule,
- introduce JIT-compilation of the rule.

Also, I've tried to improve the perfomance of rules consistency check – check on already existing in a memory rules in a case when user works with several files with the same rules.
But results are not promissing :(

Improving the performance of rules consistency check

Previous implementation: linear search.
Current implementation: binary search. I've introduced a map with the key composed of rules attributes – *source class* and *target*, or even *source class*, *target*, *version* and *checksum*.

But it brings gain in effeciency only about 5-20% (depends on different cases) with test on 20 files with 10 rules.

The following ideas could be tested:

- use as key the hash value of rule presented as string,
- when adding a rule into system, do fast check on already existing rules, only if the rule is not yet loaded, do the full procedure of adding a rule.

Support for nested objects

Required:

- updates in the rule wrapper function,
- extension of the TVirtualObject class (a proxy for representing target in-memory object and input data).

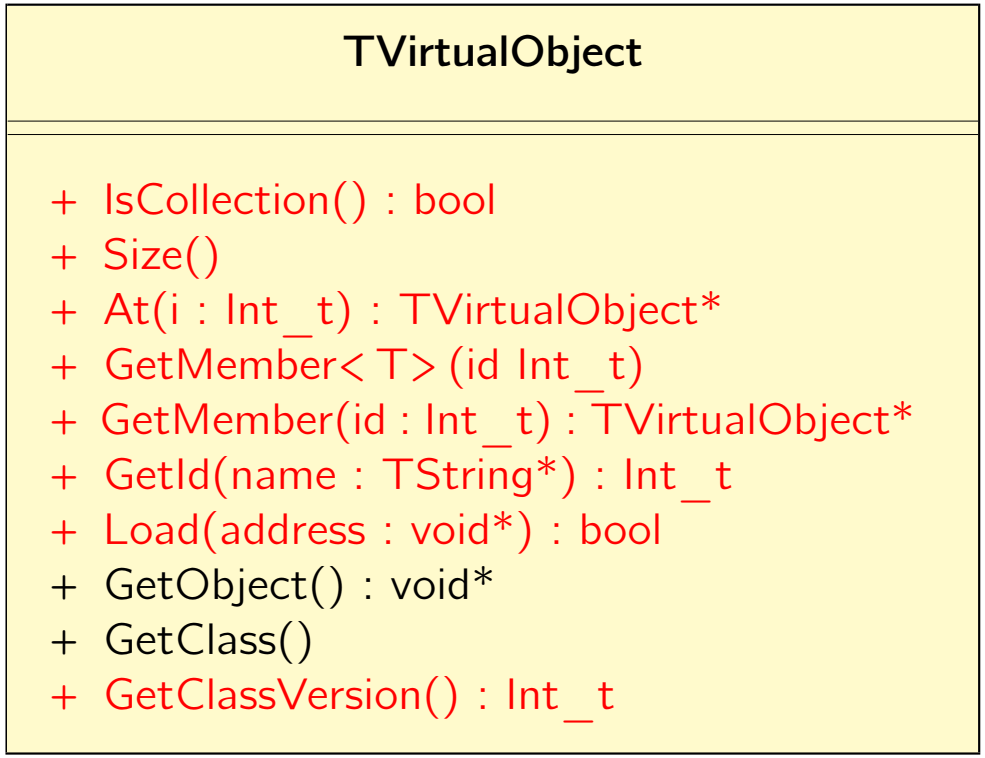
Example of the rule:

```
#pragma read sourceClass=‘‘Event"version="[2]"targetClass=‘‘Event"
source=‘‘Track fTrack;"target=‘‘fId; fCompactTrack;"
code="{ if( onfile.fTrack->GetVersion() == 3 )
{
fId = onfile.fTrack->GetMember<double>( id_fTrack_fB) +
onfile.fTrack->GetMember<double>( id_fTrack_fC );
onfile.fTrack->Load( fCompactTrack );
}
else if ( onfile.fTrack->GetVersion() == 4 )
{
fId = onfile.fTrack->GetMember<double>( id_fTrack_fB);
onfile.fTrack->Load( fCompactTrack );
}; }"
```

Updates in the rule wrapper function

- Nested object type is replaced by TVirualObject.
- Source object members are accessed via TVirtualObject methods.
- Source object members are accessed by id (to avoid doing string comparison while accessing proxified data).

Extension of TVirtualObject class



- before GSOC
- after GSOC