Table of Contents

ROOT and POOL persistency, IO and IOHelper

This twiki describes the tools to make the migration of POOL to ROOT in LHCb as was needed some time ago. These days all files are read/written with ROOT or MDF.

Foreword

Hopefully the switch from **POOL** to **ROOT** will be so transparant, you will never know anything has happened.

In case that is not true, or you need to work outside of the box, you should read this TWiki.

Quick-start guide, POOL->ROOT

Do you want to test out **ROOT** persistency in an application which doesn't come with it as default?

If you're an expert and just want to give this a go or have a reminder, here's the recipie

If you want to switch from **POOL** to **ROOT** , and it **is** supported in your application

- add `DaVinci().Persistency="ROOT"` to your options, (e.g. if DaVinci is your application)
- In Ganga add `job.application.args=["--option='from GaudiConf import IOHelper; IOHelper("+'"ROOT", "ROOT"'+ ").postConfigServices()'"]` to your job

If you want to switch from **POOL** to **ROOT** , but it **is not** supported in your application

- see task #20701
- Due to a bug in Gaudi, and missing services, no stacks older than the DaVinci v29r1 stack support ROOT.
- You can approximately acheive support by doing the following, but it will only work in some limited cases.
- `getpack GaudiConf v15r5`
- `getpack Online/RootCnv v1r12`
- `getpack GaudiSvc v18r16`
- `make` (obviously)
- if the build fails, see below
- add `from GaudiConf import IOHelper; IOHelper("ROOT","ROOT").postConfigServices()` to your options if you are running gaudi directly
- or in Ganga add `job.application.args=["--option='from GaudiConf import IOHelper; IOHelper("+'"ROOT", "ROOT"'+ ").postConfigServices()'"]` to your job

If you want to find out if **ROOT** is supported or not:

- see Is Root Supported? below

If you're not an expert, and/or want to do something strange, or the above recipie does not work in your tiny corner of phase space, see below.

Introduction

- POOL

POOL is the Gaudi-developed service for storing objects to files.

- It was in use as default by LHCb up until the middle of 2011, Gaudi v22r5.
- It is a Root-based file structure, where every event data object is stored inside its own tree.
  - Initially this was thought to be an efficient storage system, since knowing the size of one object enables you to quickly navigate to any of the objects in the tree.
  - Unfortunately Root's underlying IO was then re-optimised to be faster on other types of data structures, namely where you have one tree or a small number of trees per file.
  - this makes the **POOL** implementation incredibly inefficient in terms of IO and memory usage.
  - **POOL** was deprecated from Gaudi v23r0.

The **POOL** format is very inefficient on **Reading**, especially from remote storage e.g. over Castor. As long as your input file is in **POOL** format, you will have memory usage and IO problems.

- ROOT

ROOT is the name we give to a simpler service more in-keeping with the data format for which Root is optimized.

- It is the new default file format, as of Stripping17, i.e. from the end of 2011, Gaudi v22r5, DaVinci v29r1.
- It is a Root-based file structure where there is only one tree containing the entire event information, then another tree containing FSRs etc.
- **ROOT** was trialed with LHCb versions LHCb v32r4 onwards, but can only be considered usable from the v33r0 stack with Gaudi v22r5.
- **ROOT** is the only available persistency service from Gaudi v23r0

When a file containing the same data has been stored in **ROOT** format, reading with the **ROOT** services will be much more efficient.

If the file has been stored in **POOL** format, it can also be read in with **ROOT**, but for no gain in performance.

- Backwards/Forwards/Sideways Compatibility

- **POOL** files can be read with **ROOT** services.
- **ROOT** files cannot be read with **POOL** services.
- **POOL** files can only be written by **POOL** services.
- **ROOT** files can only be written by **ROOT** services.
- **ROOT** and **POOL** services cannot co-exist within the same Gaudi job.

- Limitations of migratability

For LHCb versions greater than v36r4, and GaudiConf versions greater than v18r0 we completely deprecated pool. This twiki is only valid up until those versions.

IOHelper and IOExtension

We have written two helper classes which are there not just for the experts, to help you configuring your jobs regardless of persistency.

IOHelper doxygen svn is the main way applications, and you, will interact with input/output it can handle:

- "dressing" filenames so that they are understood by Gaudi
- Setting up persistency services in almost any situation
- Writing output files and FSRs correctly
- Appending input files
- Added from LHCb v32r4

IOExtension doxygen svn is an alternative class which uses the three letter extension at the end of the file to decide which services to use. It can also handle:

- "dressing" filenames so that they are understood by Gaudi
- Writing output files and FSRs correctly
- Appending input files
- Added from LHCb v32r4

These classes live in the same module in the GaudiConf package.

The inbuilt python help and documentation is the best place to start with these classes.

```
from GaudiConf import IOHelper
help(IOHelper)
from GaudiConf import IOExtension
help(IOExtension)
```

Migration

Many of our applications come with **ROOT** as an option. LHCb v32r4 and higher come with the possibility to switch to **ROOT** already embedded. Older applications will gradually be instrumented with **ROOT** to make sure we can still, for example, run the trigger on older data. From Gaudi v23r0, LHCb v34r0 **ROOT** is the only option.

In order to use **ROOT**, if it is not the default persistency, requires the correct combination of three things:

1. packages,
2. options,
3. datacards.

- What type is a certain file?

A little script in AppConfig will tell you, given a file PFN, whether it is **POOL** or **ROOT** format. If it is a remote file, you may wish to copy an example locally first to check.

```
$ SetupProject SomeProject SomeVersion
$ $APPCONFIGROOT/scripts/DetectFileType.py <PFN>
```

- Is Root Supported?

For most purposes, the asnwer to this question is the same as "is my version part of the same stack as DaVinci v29r1, or later"

To tell whether you have all the correct items to use **ROOT** persistency.

```
$ SetupProject SomeProject SomeVersion
```

```
$ python
from GaudiConf import IOHelper
print IOHelper().isRootSupported()
```

If this script succeeds without raising an exception and prints "True", then Root is supported in this application.

Unfortunately for some versions this may return true, when actually root is not supported due to bugs in the underlying Gaudi. It will give you some idea and we are working on a better solution, but eventually it comes down to "is my version part of the same stack as DaVinci v29r1, or later"

Updating your local version of IOHelper will pick up the check on the underlying version of GaudiSvc. Only GaudiSvc >=v18r3 supports Root properly.

- What is the default?

The default persistency is controlled by IOHelper. If IOHelper does not exist, then only **POOL** is supported.

```
SetupProject SomeProject SomeVersion
python
from GaudiConf import IOHelper
print IOHelper().defaultPersistency()
```

If this script throws an exception, then assume **POOL**, otherwise the default is the persistency of the defaultly constructed IOHelper.

- What packages do I need?

Assuming **ROOT** isn't already supported, you need to get:

```
getpack GaudiConf v15r5
getpack Online/RootCnv v1r12
getpack GaudiSvc v18r16
```

Once these are compiled and correctly on your path, checking if Root is supported will return True.

-> What if the build fails

- **The head of GaudiSvc and also RootCnv cannot be combined for certain Gaudi versions which are sufficiently old...**
- Try `echo $GAUDISVCROOT` to find the version of older Gaudi stacks
- For **Gaudi >= v22r4**, the build should work as given.
  - see the method above, this is for LHCb stacks >=v33r0
- For **Gaudi >= v22r0**, GaudiSvc will not compile, because of the missing msgLevel function, so you'll need to getpack the version of GaudiSvc which was actually released in that gaudi version, and patch it yourself:
  - This is LHCb versions >=v32r0
  - e.g.

```
getpack GaudiSvc
WARNING : Version not specified for package 'GaudiSvc'
Select a version (v19r9_v16r6, v18r17, v18r16, v18r15, v18r14, v18r13, v18r12, v18r11, v18r10, v1
...
cd GaudiSvc
svn merge svn+ssh://svn.cern.ch/reps/gaudi/Gaudi/trunk/GaudiSvc -c6649 .  #this will probably giv
Conflict discovered in 'doc/release.notes'.
Select: (p) postpone, (df) diff-full, (e) edit,
        (mc) mine-conflict, (tc) theirs-conflict,
```

```
        (s) show all options: p
--- Merging r6649 into '.':
C    doc/release.notes
U    src/PersistencySvc/OutputStream.cpp
Summary of conflicts:
  Text conflicts: 1
```

- For Gaudi < v22r0, RootCnv will not compile, due to the missing `DataObject.update()` method. There is no way to get around that right now, the entire stack would need to be rebuilt up to your version with a later/better Gaudi version.
  - ♦ contact your release manager

- What about the options?

1. If **ROOT** is supported, is the default, and the application is a recent one, you do not need to change anything in order to pick up the **ROOT** persistency.
2. If **ROOT** is supported, without you needing to getpack anything, but POOL is the default, you can probably set the application to pick up **ROOT** using the configurable for the application such as:

```
DaVinci().Persistency="ROOT"
#or
Brunel().Persistency="ROOT"
```

1. If **ROOT** is supported, but it is because you did the getpacking of the correct packages, you will instead need to force the switch of the services by appending a PostConfigAction::

```
from GaudiConf import IOHelper
IOHelper("ROOT","ROOT").postConfigServices()
```

using `postConfigServices` is a catch-all way of always ensuring you switch to **ROOT**, or back to **POOL**, no matter of the other options in your file are/were, and no matter what your datacard was.

- What about the?

Datacards have hard-coded within them the type of services they should be read with. This is not usually a problem, since the python configuration can and does take care of swapping backwards and forwards between **POOL** and **ROOT** services.

However, if you have a use case where IOHelper is not involved, such as:

- some obscure application
- some GaudiPython script
- Data added with its own postConfigAction
- Data added or edited during run time

It is advisable to first convert your datacard into a better format, with the correct persistency.

- Old format Gaudi Cards

the old format is to set directly the property of the EventSelector:

```
from Gaudi.Configuration import *

EventSelector().Input   = [
"   DATAFILE='LFN:/lhcb/MC/MC09/DST/00004871/0000/00004871_00000001_1.dst' TYP='POOL_ROOTTREE' OP
...
]
```

In **ROOT** this becomes:

```
from Gaudi.Configuration import *

EventSelector().Input   = [
"   DATAFILE='LFN:/lhcb/MC/MC09/DST/00004871/0000/00004871_00000001_1.dst' SVC='Gaudi::RootEvtSel
...
]
```

Notice how this overwrites any previous content of the EventSelector, and remember that for the next step.

- New format Gaudi Cards

With IOHelper the setting of input files is much easier:

```
from GaudiConf import IOHelper
IOHelper().inputFiles([
    "LFN:/lhcb/MC/MC09/DST/00004871/0000/00004871_00000001_1.dst",
...
   ])
```

You only set the file name, and do not worry about the persistency itself. IOHelper handles that for you.

NB: The files in inputFiles are ADDED to the EventSelector, no previous content is removed

If you want to remove the previous content, set the keyword `clear=True`

```
from GaudiConf import IOHelper
IOHelper().inputFiles([
    "LFN:/lhcb/MC/MC09/DST/00004871/0000/00004871_00000001_1.dst",
...
   ], clear=True)
```

- Modifying the format or persistency of

There is a simple script again in AppConfig which will parse any number of options files and translate the data to IOHelper format. This only will not work if you have data set inside a different configurable or inside a postConfigAction.

```
$ SetupProject LHCb v33r0 --use AppConfig
$ $APPCONFIGROOT/scripts/ConvertGaudiCard.py --help
usage: ConvertGaudiCard.py <inputfile> [<inputfile>...] [--new/--ext/--old] [--persistency=<POOL,
   inputfile: gaudi card or options file to parse
   --new: (default) output in new IOHelper style
   --ext: output in new IOExtension style
   --old: output in old GaudiCard style
   --persistency: Convert persistency to given format, otherwise use default
   --setpersistency: explicitly give the persistency in the card, even if it is the default
```

This will help if you need to revert new-style cards to an older format or even fix problems with broken GaudiCards.

- Inside Ganga?

You do not need to do anything if you are using the default persistency, with the most recent Ganga/DaVinci

- Ganga v506r8 is the minimum version required to support ROOT persistency.
- https://savannah.cern.ch/bugs/?82532

If you need to use a non-default persistency:

- Ganga flattens options files, this can mean IOHelper gets in a spot of bother when trying to interpret your input.
- There are two ways potentially to circumvent this within Ganga

1. **Command line arguements to `gaudirun.py`, which can be used to circumvent the flattening and set the correct options.**
    - ♦ This method is a catch-all which should always work, for both ROOT and POOL
    - ♦ If you are migrating to a non-default persistency inside of Ganga, you should set the command line arguement to do the migration:

```
job.application.args=["--option='from GaudiConf import IOHelper; IOHelper("+'"ROOT", "ROOT"'+ ").
```

1. **Directly in the LHCbDataset**
    - ♦ This method may not work in all cases, it relies that you have already setup the correct services, and only need to change the datacard
    - ♦ In this case Ganga allows the dataset itself to know about it's persistency, so you can specify:

```
job.inputdata.Persistency="ROOT"
```

What if running fails (`R_unzip: error in header`)?

- Depending on the file type, and how the DSTs were generated you will not be able to run sufficiently old software on sufficiently new files.
- Recently LHCb chose to change compression algorithm, which saves up to 10% of grid-space usage for DSTs, however this alg is not understood be very old versions of ROOT.
- Running Gaudi versions <v22r4, LHCb < v33r0 on files from at or after Stripping 17, LHCb v35r0.
- If you get `R_unzip: error in header` printed for every file you have three choices:
    1. Regenerate the DSTs if possible, with the older gzip compression.
        ◊ Add `from Configurables import RootCnvSvc; RootCnvSvc().GlobalCompression = "ZLIB:1"`, to your generation options at the final stage
    2. Run a conversion step before your older version which writes out the same files in the older compression, probably to local disk.
        ◊ Same options as choice 1
    3. OR, contact your release manager to get a release of the software based on newer Root/Gaudi, this won't necessarily do what you want, though.

What can possibly go wrong?

Because you need the combination of DataCard, Packages and Options all to be correct, there are many permutations of many different scenarios, most of which lead to failure of one sort or another. Attatched is a flow chart of everything which may go wrong.
https://twiki.cern.ch/twiki/pub/LHCb/PersistencyMigration/InputData.pdf

---

FAQ

- TypeErrors in Ganga when parsing dataset.

Errors of the type: `!TypeError: Type of file xxxx could not be determined use IOHelper with specified persistency instead`

- **If the file name has a "?" somewhere** in it to designate the svcClass
  - this is a known bug. Getpack a version of GaudiConf >= v15r2
- **If the file extension is wierd** such as .fishpastesandwiches or .spamandeggs .xmsdst
  - IOExtension cannot detemine the file type from weird extensions
  - Extensions are gradually being added to IOExtension contact your release manager if you find one is missing, but first try the latest GaudiConf

---

-- RobLambert - 05-Aug-2011

This topic: LHCb > PersistencyMigration
Topic revision: r19 - 2014-01-20 - RobLambert