dunetpc - Task #21259

Update ArtServiceHelper for art 3

10/30/2018 11:54 AM - David Adams

Status:	Accepted	**Start date:**	10/30/2018
Priority:	Normal	**Due date:**	
Assignee:	David Adams	**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			

Description

Kyle reports that art version 3 breaks the dunetpc class ArtServiceHelper. We should fix that before migrating to art 3.

History

#1 - 10/30/2018 11:56 AM - David Adams

From Kyle:

I have migrated ArtServiceHelper to support art 3; you can see the result on the head of the feature/team_for_art_v3 branch in dunetpc. I may not have gotten all of the tests worked out, so Lynn please let me know if I have more work to do there.

David, I made quite a few changes to the class:

- Since it now is a layer on top of art's service manager, I removed explicit calls to addService, and I just let the framework classes handle all of that. The only accessible interface is ArtServiceHelper::load_services.
- I've removed the close() function since that behavior is not emulated in the framework. Of course, you may choose to restore it as you like.
- I also removed several "debugging" functions since everything is entirely managed by the framework classes. Again, feel free to restore what you think is necessary.
- The test_TFileService test has been removed because it relies on the close() behavior, which is no longer there
- The test_RandomNumberGenerator test has been removed because it really cannot be made to work outside of art any more—i.e. the RNG interface has changed a bit to support multi-threaded execution, and in art 3.02 the RandomNumberGenerator::getEngine function will be deprecated, and then removed in art 3.03. I've discussed this at a few LArSoft coordination meetings.

The goal of this exercise was to be a help in getting experiment repositories ready for art 3—not to remove facilities that experiments need. I believe I've gotten things to a point where meaningful tests that rely on dune/ArtSupport are still being done. Modulo any issues with the tests, I believe DUNE is now pretty close to being ready for art 3. Of course, everything is up for discussion. Don't hesitate to get ahold of me…phone calls/Zoom meetings are fine if it makes communication easier.

Cheers,
Kyle

#2 - 12/13/2018 09:15 AM - David Adams

Kyle:

I am testing the new ArtServiceHelper using dunetpc branch feature/team_for_art_v3 with art v08_00_00_rc2. I added ArtServiceHelper::load for backward compatibility and tried to load services from a root session. I call ArtServiceHelper::load(...) without error but when I exit Root, I get a crash with the following trace:

```
#5  0x0000000001502f600 in ?? ()
#6  0x00007fa6ab05e7ad in util::LArFFT::~LArFFT() () at /scratch/workspace/build-larsoft/v08_00_00_rc2/SLF6/pr
of/build/lardata/v08_00_00_rc2/src/lardata/Utilities/LArFFT_service.cc:70
#7  0x00007fa6ab060019 in art::detail::ServiceWrapper<util::LArFFT, (art::ServiceScope)0>::~ServiceWrapper() (
) at /scratch/workspace/build-larsoft/v08_00_00_rc2/SLF6/prof/build/gcc/v7_3_0/Linux64bit+2.6-2.12/include/c++
/7.3.0/bits/shared_ptr_base.h:154
#8  0x00007fa6ac5ee529 in std::_Rb_tree<art::TypeID, std::pair<art::TypeID const, art::detail::ServiceCacheEnt
ry>, std::_Select1st<std::pair<art::TypeID const, art::detail::ServiceCacheEntry> >, std::less<art::TypeID>, s
td::allocator<std::pair<art::TypeID const, art::detail::ServiceCacheEntry> > >::_M_erase(std::_Rb_tree_node<st
d::pair<art::TypeID const, art::detail::ServiceCacheEntry> >*) () at /scratch/workspace/art-release-build/SLF6
/prof/build/gcc/v7_3_0/Linux64bit+2.6-2.12/include/c++/7.3.0/bits/shared_ptr_base.h:154
#9  0x00007fa6ac5ee36f in std::_Rb_tree<art::TypeID, std::pair<art::TypeID const, art::detail::ServiceCacheEnt
ry>, std::_Select1st<std::pair<art::TypeID const, art::detail::ServiceCacheEntry> >, std::less<art::TypeID>, s
td::allocator<std::pair<art::TypeID const, art::detail::ServiceCacheEntry> > >::_M_erase(std::_Rb_tree_node<st
d::pair<art::TypeID const, art::detail::ServiceCacheEntry> >*) () at /scratch/workspace/art-release-build/SLF6
/prof/build/gcc/v7_3_0/Linux64bit+2.6-2.12/include/c++/7.3.0/bits/stl_tree.h:1856
#10 0x00007fa6ac5ee36f in std::_Rb_tree<art::TypeID, std::pair<art::TypeID const, art::detail::ServiceCacheEnt
ry>, std::_Select1st<std::pair<art::TypeID const, art::detail::ServiceCacheEntry> >, std::less<art::TypeID>, s
td::allocator<std::pair<art::TypeID const, art::detail::ServiceCacheEntry> > >::_M_erase(std::_Rb_tree_node<st
```

```
d::pair<art::TypeID const, art::detail::ServiceCacheEntry> >*) () at /scratch/workspace/art-release-build/SLF6
/prof/build/gcc/v7_3_0/Linux64bit+2.6-2.12/include/c++/7.3.0/bits/stl_tree.h:1856
#11 0x00007fa6ac5ee36f in std::_Rb_tree<art::TypeID, std::pair<art::TypeID const, art::detail::ServiceCacheEnt
ry>, std::_Select1st<std::pair<art::TypeID const, art::detail::ServiceCacheEntry> >, std::less<art::TypeID>, s
td::allocator<std::pair<art::TypeID const, art::detail::ServiceCacheEntry> > >::_M_erase(std::_Rb_tree_node<st
d::pair<art::TypeID const, art::detail::ServiceCacheEntry> >*) () at /scratch/workspace/art-release-build/SLF6
/prof/build/gcc/v7_3_0/Linux64bit+2.6-2.12/include/c++/7.3.0/bits/stl_tree.h:1856
#12 0x00007fa6ac5e7c0e in art::ServicesManager::~ServicesManager() () at /scratch/workspace/art-release-build/
SLF6/prof/build/gcc/v7_3_0/Linux64bit+2.6-2.12/include/c++/7.3.0/bits/stl_tree.h:1171
#13 0x00007fa6b0d884a0 in ArtServiceHelper::~ArtServiceHelper() () at /nashome/d/dladams/dev/dudev-art3/workdi
r/srcs/dunetpc/dune/ArtSupport/ArtServiceHelper.h:109
#14 0x0000003f17035a02 in exit () from /lib64/libc.so.6
#15 0x00007fa6beabb650 in TUnixSystem::Exit(int, bool) () from /cvmfs/larsoft.opensciencegrid.org/products/roo
t/v6_12_06a/Linux64bit+2.6-2.12-e17-prof/lib/libCore.so
#16 0x00007fa6bea08261 in TApplication::Terminate(int) () from /cvmfs/larsoft.opensciencegrid.org/products/roo
t/v6_12_06a/Linux64bit+2.6-2.12-e17-prof/lib/libCore.so
<pre>

Although I cannot be certain this is the problem here, this sort of error was the reason for adding the close
method to the original ArtServiceHelper. Would it be easy to do something equivalent in art3?

Thanks.

da
```

#3 - 12/13/2018 12:21 PM - David Adams

I think the important difference for LArFFT is that the art3 build instantiates all services while the art2 version only instantiated those that were retrieved (via ServiceHAndle). If I make a retrieval in my art2 root session, then I also get a crash in the LArFFT dtor:

```
============================================================
#5  0x0000000012e89a70 in ?? ()
#6  0x00007fc08ce065fd in util::LArFFT::~LArFFT() () at
/scratch/workspace/build-larsoft/v07_13_00/SLF6/prof/build/lardata/v07_01_01/src/lardata/Utilities/LArFFT_serv
ice.cc:70
#7  0x00007fc08ce07e19 in art::detail::ServiceWrapper<util::LArFFT,
(art::ServiceScope)0>::~ServiceWrapper() () at
/scratch/workspace/build-larsoft/v07_13_00/SLF6/prof/build/gcc/v7_3_0/Linux64bit+2.6-2.12/include/c++/7.3.0/bi
ts/shared_ptr_base.h:154
#8  0x00007fc08e71ba96 in art::ServicesManager::~ServicesManager() () at
/scratch/workspace/art-release-build/SLF6/prof/build/gcc/v7_3_0/Linux64bit+2.6-2.12/include/c++/7.3.0/bits/sha
red_ptr_base.h:154
#9  0x00007fc08e71b329 in art::ServiceRegistry::~ServiceRegistry() () at
/scratch/workspace/art-release-build/SLF6/prof/build/gcc/v7_3_0/Linux64bit+2.6-2.12/include/c++/7.3.0/bits/sha
red_ptr_base.h:154
#10 0x0000003d4e235a02 in exit () from /lib64/libc.so.6
#11 0x00007fc0a0e85650 in TUnixSystem::Exit(int, bool) () from
/home/dladams/ups/root/v6_12_06a/Linux64bit+2.6-2.12-e17-prof/lib/libCore.so
#12 0x00007fc0a0dd2261 in TApplication::Terminate(int) () from
/home/dladams/ups/root/v6_12_06a/Linux64bit+2.6-2.12-e17-prof/lib/libCore.so
============================================================
```

#4 - 12/13/2018 12:40 PM - Kyle Knoepfel

Hi David, I intentionally redesigned the ArtServiceHelper class so that it uses the same mechanism for loading services as the *art* framework does. The framework constructs all configured services, and it does not issue any close command because it relies on C++ lifetime to do that naturally. The ArtServiceHelper behavior is the same. It may be possible to adjust the ArtServiceHelper so that it can work the same way with *art* 3 as it did in *art* 2. If you would like to use it within ROOT, and you think that loading extraneous services is the culprit, a judicious use of 'services.ExtraService: @erase' may solve the problem, but I can't promise that. An *art*-based service is not created/maintained with the intention of being used within an interactive ROOT environment. Trying to provide that use case may prove very difficult, unless extra constraints are imposed on service authors.

What is important for testing outside of the framework is not the service itself, but the functionality that the *art* service (hopefully) wraps around. LArSoft's approach to this has been to distinguish between services themselves and providers, which do not rely on the art core framework at all, but provide the required functionality outside of it, can still be configured by FHiCL, and can be used within (e.g.) *gallery*. *art*'s service system simply makes such providers accessible from within the framework, calling appropriate callbacks at the right transitions, and enabling the creation of ServiceHandle objects.

#5 - 12/13/2018 03:07 PM - David Adams

Indeed, removing LArFFt from the fcl does "solve" the problem--there is no longer a crash at closeout although I do see a warning message:

```
Warning in <TFile::Write>: file ./TFileService-dde7-c660-aea2-3837.root not opened in write mode
```

I understand that services used outside the framework will not see state transitions but most don't need this and they should be easy to use outside the art framework. Other than that, I don't see much burden on service (or tool or utility) developers to write code that runs equally well in the art framework, Root interactive or standalone.

I looked at bit at the Root FFT code. Although one check suggests we have avoided the pitfall of deleting an FFT Object owned by Root, I see the Root FFT classes do make use of some global data. I suspect we are again running into problems because the Root cleanup is done before we delete our services. Our FFT service deletes a Root FFT object after this global data is cleaned leading to the crash.

I think the best solution is the one we had in the art2 code: add a close method to the service helper that deletes the services and we can make sure that is called before C++ (most importantly Root) cleanup. So, I would like to restore this function. As you know, I had to do something pretty ugly in art2. Will that still work? Or can you provide a nicer interface in art that has this effect?

Thanks.

#6 - 12/18/2018 11:59 AM - David Adams

Kyle:

Can you advise on how to restore ArtServiceHelper::close() in art 3?

Thanks.

da

#7 - 12/18/2018 12:00 PM - Kyle Knoepfel

Sorry, David--been busy. I should have some time to respond on this later today.

#8 - 12/19/2018 04:01 PM - Kyle Knoepfel

David, can you give me specific instructions to reproduce the ROOT error?

#9 - 12/20/2018 11:19 AM - David Adams

Set up dunetpc, clone git@github.com:dladams/protodune-adc-calib.git, source setup.sh in the latter from and empty directory, start and exit root.

#10 - 12/21/2018 03:38 PM - Kyle Knoepfel

Thank you for the instructions, David. I am able to reproduce the segmentation violation. A double-delete is happening for the TF1 object pointed to by fPeakFit in LArFFT. This is due to ROOT's self-registration of objects. I am not certain that adding a close function will fix this problem, but if you have encountered this type of thing before, let's assume it will for now.

No changes are necessary in *art* to provide an ArtServiceHelper::close() function. *art* services are owned by the ServicesManager object, which has no static member data. So as long as the ServicesManager object is destroyed, all services will be destroyed and you can start over. Try the following pattern:

```
#include <cassert>
#include <iostream>
#include <memory>
#include <string>

namespace {
  // Arbitrary struct to demonstrate:
  struct ServicesManager {
    std::string filename; // As a proxy for a bunch of owned services
  };

  class ArtServiceHelper {
  public:
    static void load(std::string const&);
    static void close();

    static auto const& current_file()
    {
      assert(instance);
      return instance->manager_.filename;
    }

  private:
    explicit ArtServiceHelper(std::string const& file) : manager_{file} {}
    ServicesManager manager_;
    static std::unique_ptr<ArtServiceHelper> instance;
  };

  std::unique_ptr<ArtServiceHelper> ArtServiceHelper::instance{nullptr};
```

```
  void
  ArtServiceHelper::load(std::string const& file)
  {
    instance.reset(new ArtServiceHelper{file});
  }

  void
  ArtServiceHelper::close()
  {
    instance.reset();
  }
}

int main()
{
  ArtServiceHelper::load("my.fcl");
  std::cerr << ArtServiceHelper::current_file() << '\n';
  ArtServiceHelper::close();

  ArtServiceHelper::load("my_other.fcl");
  std::cerr << ArtServiceHelper::current_file() << '\n';
  ArtServiceHelper::close();
}
```

Compiling with GCC 7.2 (e17), this printed out:

```
my.fcl
my_other.fcl
```

I think this will get you where you want to go.

I will be unavailable over the holidays, but I can pick up this thread again in January.

#11 - 12/21/2018 03:43 PM - Kyle Knoepfel

P.S. In the close function, you'll probably want to call art::EventProcessor::set_services_manager(nullptr).