JANA2 External Wiring Update

Nathan Brei

October 27, 2025

ePIC Reconstruction Working Group Meeting

Agenda

1. Previously: JWiredFactoryGenerator
2. Rewirable JFactoryGenerator- and JOmniFactoryGenerator- created JFactories
3. Partial rewirings
4. Rewirable JEventProcessors, JEventUnfolders, etc
5. Enabling and disabling JComponents
6. Ongoing work

Previously: JWiredFactoryGenerator

Previously: JWiredFactoryGenerator

- Leave JOmniFactory exactly the same
- Introduce JWiredFactoryGenerator
- The generator will create new factory instances for each wiring listed in the wiring file belonging to the current plugin
- Users can still use JOmniFactoryGenerator for other factories as long as the same wirings aren't specified both places
- Problem: We want JANA2 to be usable without a wiring file, and we want to be able to gradually migrate
- EICrecon has exactly two configurations 'presently': normal mode and timeframe-splitting mode

```C++
extern "C"
void InitPlugin(JApplication *app) {
    InitJANAPlugin(app);
    app->Add(new JWiredFactoryGeneratorT<ClusterFac>());
}
```

```toml
[[wiring]]
action = "add"
plugin_name = "clustering"
type_name = "ClusterFac"
prefix = "simple_clusterizer"
input_names = ["protoclusters"]
output_names = ["simple_clusters"]
    [wiring.configs]
    offset = "22"

[[wiring]]
action = "add"
plugin_name = "clustering"
type_name = "ClusterFac"
prefix = "weird_clusterizer"
input_names = ["protoclusters"]
output_names = ["weird_clusters"]
    [wiring.configs]
    offset = "100"
```

Rewirable JFactoryGenerator- and JOmniFactoryGenerator- created JFactories

Rewirable JFactoryGenerator- and JOmniFactoryGenerator-created JFactories

- The wiring file now overrides values set in any factory class, JFactoryGenerator, or JOmniFactoryGenerator
- For this to work, JFactory, JFactoryT, JMultifactory, and JOmniFactory all have a new Wire() method which gets called by JANA in the background
- Furthermore, all factory classes now inherit from JFactory and use the same machinery to provide Inputs, (multiple) Outputs, Parameters, and Services.
- This makes the implementations of JMultifactory and JOmniFactory much simpler. The complexity is moved into JDatabundle.

C++

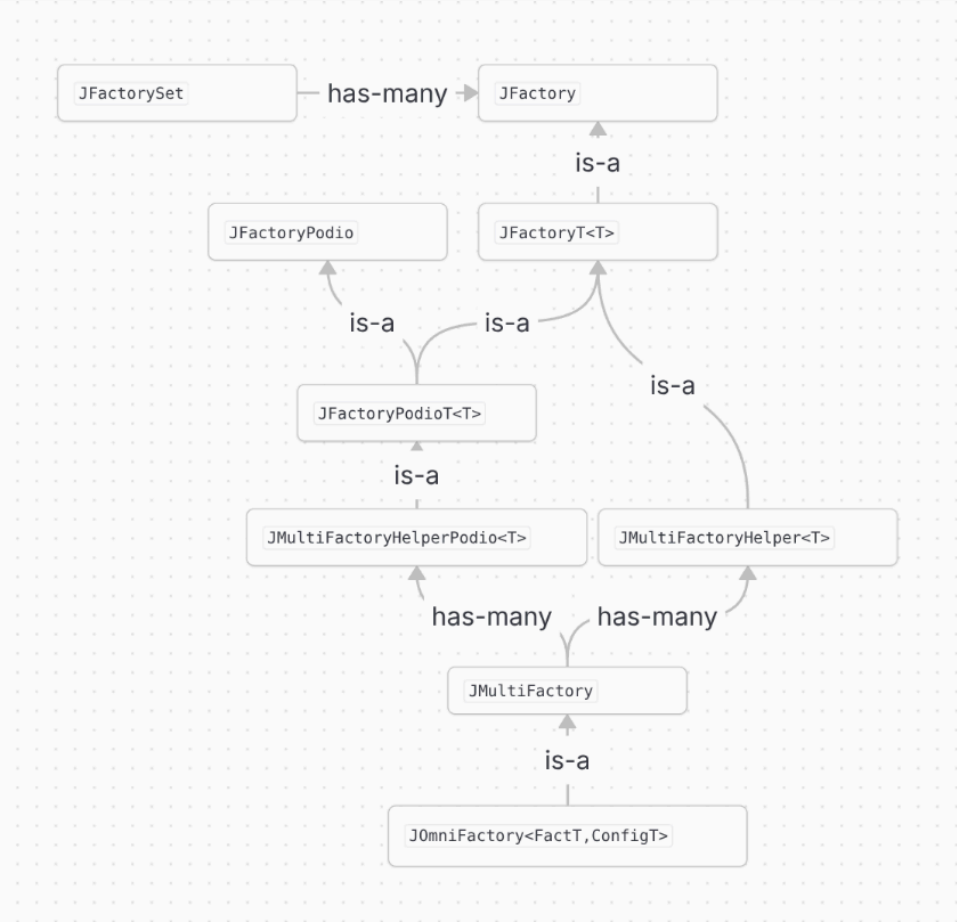
```
app->Add(new JOmniFactoryGeneratorT<SiliconTrackerDigi_factory>(
  "TOFBarrelRawHits", {"EventHeader", "TOFBarrelHits"},
  {"TOFBarrelRawHits", "TOFBarrelRawHitAssociations"},
  {
      .threshold      = 6.0 * dd4hep::keV,
      .timeResolution = 0.025, // [ns]
  },
  app));
```

toml

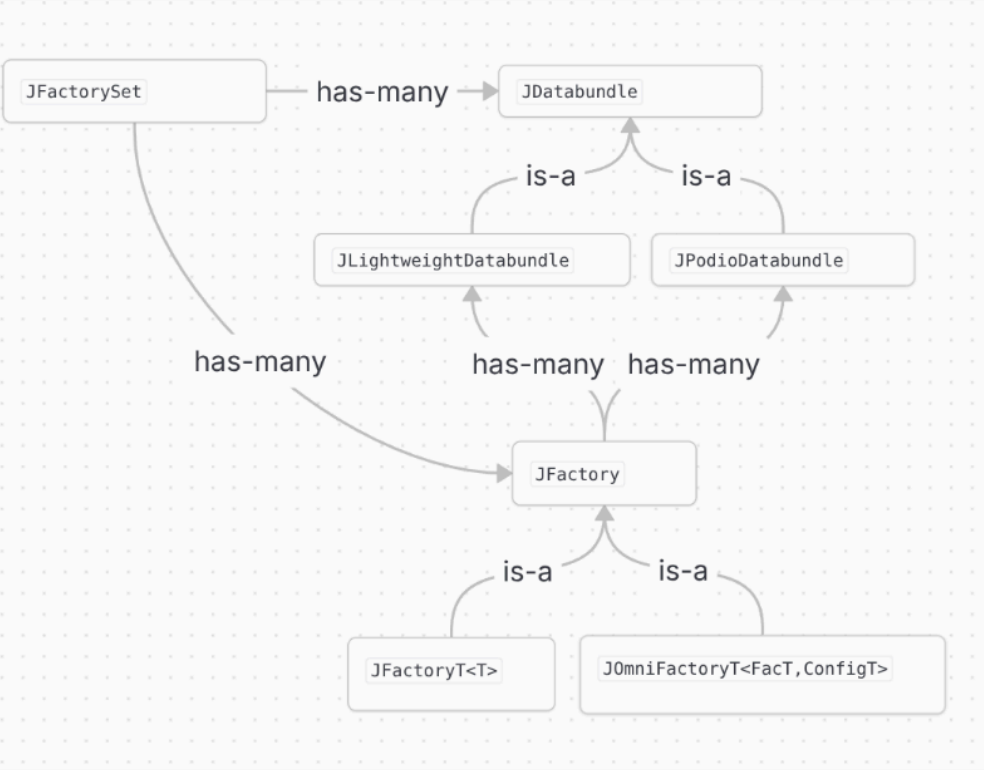
```
[[wiring]]
action = "update"
plugin_name = "BTOF"
type_name = "SiliconTrackerDigi_factory"
prefix = "TOFBarrelRawHits"
level = "Timeslice"
input_names = ["EventHeader", "TOFBarrelHits"]
output_names = ["TOFBarrelRawHits", "TOFBarrelRawHitAssociations"]

    [factory.configs]
    threshold      = "6000.0"    # dd4hep::keV,
    timeResolution = "0.025"     # [ns]
```

JFactory, JFactoryT, JMultifactory, JOmniFactory refactoring

Before



After



Partial rewirings

Partial Rewirings

- Previously, the wiring file contained a complete description of inputs, outputs, parameters, event levels, etc for each factory
- Unlike JWiredFactoryGenerator, JOmniFactoryGenerator is designed to capture the exact same information
- For the sake of gradual migrations, I don't want to force all JOmniFactoryGenerators to be converted to JWiredFactoryGenerators
- I **don't** believe that moving all wiring info out of the code entirely is a good idea
- For usability, it is better for the (input) wiring file to be a diff with respect to the default configuration, which lives in the code

C++

```
1
2     app->Add(new JOmniFactoryGeneratorT<SiliconTrackerDigi_factory>(
3       "TOFBarrelRawHits", {"EventHeader", "TOFBarrelHits"},
4       {"TOFBarrelRawHits", "TOFBarrelRawHitAssociations"},
5       {
6           .threshold      = 6.0 * dd4hep::keV,
7           .timeResolution = 0.025, // [ns]
8       },
9       app));
10
```

toml

```
1
2   [[wiring]]
3   action = "update"
4   plugin_name = "BTOF"
5   type_name = "SiliconTrackerDigi_factory"
6   prefix = "TOFBarrelRawHits"
7   level = "Timeslice"
8
```

Rewirable JEventProcessors, JEventUnfolders, etc

Rewirable JEventProcessors, JEventUnfolders, etc

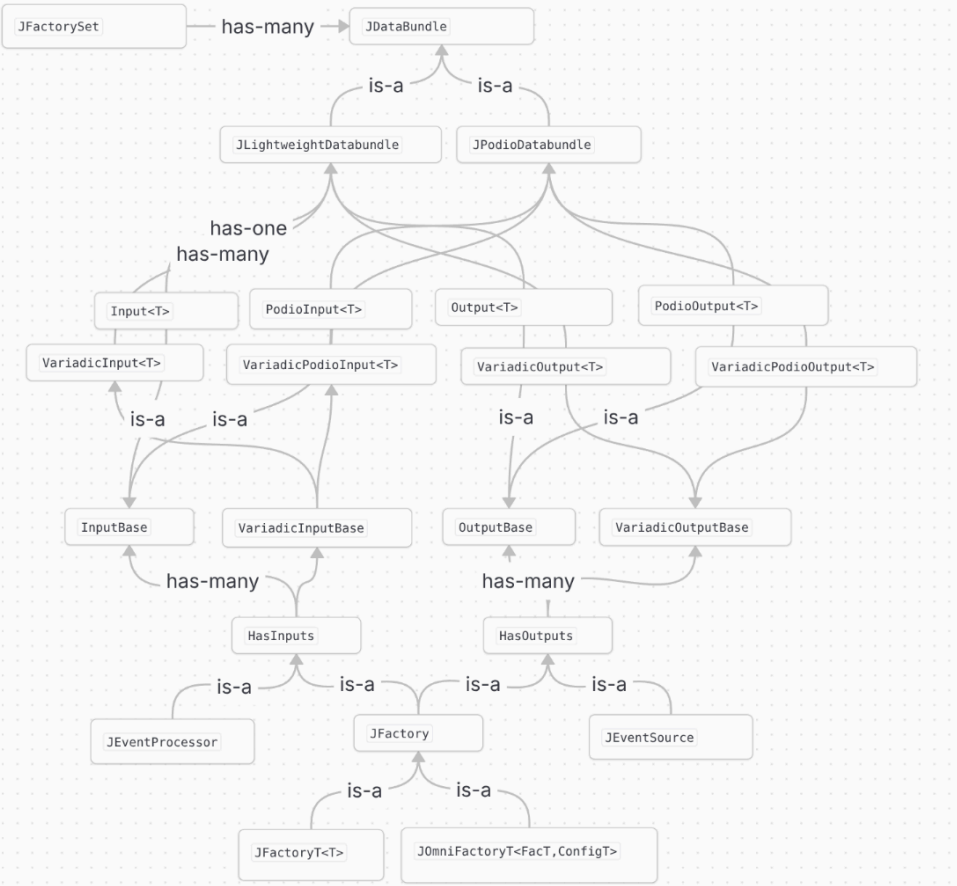
- JEventSources, JEventUnfolders, JEventFolders, and JEventProcessors similarly have a Wire() method now
- Note that Wire() only works with declared Inputs/Outputs/Parameters, not with JEvent::GetCollection() or JApplication::RegisterParameter() calls
- Note that `variadic_{inputs,outputs}` has been separated from `{inputs,outputs}`. This allows the number of collections in each variadic input/output to vary independently.

C++
```
extern "C" {
void InitPlugin(JApplication* app) {
  if (app->RegisterParameter<bool>("split_timeframes", false,
                                   "Enable timeframe splitting")) {
    app->Add(new TimeframeSplitter());
    // ... Add additional Timeframe-level factories
  }}
```

toml
```
[[wiring]]
action = "update"
plugin_name = "splitting"
type_name = "TimeframeSplitter"
prefix = "timeframe_splitter"
input_names = ["EventHeader", "MCParticles"]
output_names = ["EventHeader", "MCParticles"]
variadic_input_names = [
  ["B0TrackerRecHits_aligned", ...],           # SimTrackerHits
  ["B0ECalHits", "EcalBarrelImagingHits", ...], # SimCalorimeterHits
  ["B0ECalHitsContributions", ...]             # CaloHitContribution
]
variadic_output_names = [
  ["B0TrackerRecHits", ...],                   # SimTrackerHits
  ["B0ECalHits", "EcalBarrelImagingHits", ...], # SimCalorimeterHits
  ["B0ECalHitsContributions", ...]             # CaloHitContribution
]
```

Refactoring JComponent

- All components inherit from `JComponent`, which provides `Parameter` and `Service` helpers
- Factories and unfolders inherit from `JHasInputs`, `JHasOutputs`, etc.
- JHasInputs provides `{Variadic}{Podio}Input` helpers; JHasOutputs provides `{Variadic}{Podio}Output` helpers
- Each `Input` and `Output` refers to one or more `JDatabundles` which abstract a collection of data products
- Databundles come in `Lightweight` or `Podio` varieties
- This design allows factoring out the Podio dependency completely, can be extended to support additional datamodel facilities



Enabling and disabling JComponents

Enabling and disabling JComponents: Motivation

- Previously, there were two ways to enable/or disable a JANA component:
  - By including or omitting its plugin
  - By checking a parameter during InitPlugin()
- Both of these approaches have problems:
  - Rewiring (particularly for timeframe splitting) crosses plugin boundaries
  - Parameters aren't sufficient to switch between configurations, nor independent from one another
  - External wiring is particularly desired when no plugins are used at all

C++

```cpp
extern "C" {
void InitPlugin(JApplication* app) {
  if (app->RegisterParameter<bool>("split_timeframes", false,
                                   "Enable timeframe splitting")) {
    app->Add(new TimeframeSplitter());
    // ... Add additional Timeframe-level factories
}}
```

toml

```toml
[[wiring]]
action = "update"
plugin_name = "splitting"
type_name = "TimeframeSplitter"
prefix = "timeframe_splitter"
input_names = ["EventHeader", "MCParticles"]
output_names = ["EventHeader", "MCParticles"]
variadic_input_names = [
  ["B0TrackerRecHits_aligned", ...],           # SimTrackerHits
  ["B0ECalHits", "EcalBarrelImagingHits", ...], # SimCalorimeterHits
  ["B0ECalHitsContributions", ...]             # CaloHitContribution
]
variadic_output_names = [
  ["B0TrackerRecHits", ...],                   # SimTrackerHits
  ["B0ECalHits", "EcalBarrelImagingHits", ...], # SimCalorimeterHits
  ["B0ECalHitsContributions", ...]             # CaloHitContribution
```

Enabling and disabling JComponents: New design

- Now, components have an `is_enabled` flag, so that they can be registered with JANA regardless of whether they will be needed or not
- The plugin may choose to enable/disable each component directly, and the wiring file may choose to override that choice
- In the wiring file, the `add` and `remove` actions override the `is_enabled` flag for all non-factory components
- Note: Factories still require a JWiredFactoryGenerator in order to `add` (but not to `remove`)

C++
```
extern "C" {
void InitPlugin(JApplication* app) {
  auto splitter = new TimeframeSplitter();
  splitter->SetEnabled(false);
  app->Add(splitter);  // Registers with JANA but disables by default
  // ... Add additional Timeframe-level factories
}}
```

toml
```
[[wiring]]
action = "add"
plugin_name = "splitting"
type_name = "TimeframeSplitter"
prefix = "timeframe_splitter"
input_names = ["EventHeader", "MCParticles"]
output_names = ["EventHeader", "MCParticles"]
variadic_input_names = [
  ["B0TrackerRecHits_aligned", ...],           # SimTrackerHits
  ["B0ECalHits", "EcalBarrelImagingHits", ...], # SimCalorimeterHits
  ["B0ECalHitsContributions", ...]             # CaloHitContribution
]
variadic_output_names = [
  ["B0TrackerRecHits", ...],                   # SimTrackerHits
  ["B0ECalHits", "EcalBarrelImagingHits", ...], # SimCalorimeterHits
  ["B0ECalHitsContributions", ...]             # CaloHitContribution
```

Ongoing work

Ongoing design work

Parameter values

- Still represented in TOML file as strings
- Needs to handle units: `threshold = 6.0 * dd4hep::keV`
- Parameter manager needs to know which file each value came from

Inheriting/overlaying multiple configuration files

- Main functionality implemented already, but hasn't been tested beyond the level of unit tests
- Not on the critical path, but a good task for a motivated junior developer

Reporting the accumulated "ultimate wiring"

- Shows the complete wiring configuration for all components after all initialization finishes
- Important for auditability
- This overlaps with (and might someday replace) JComponentSummary

Plan for integrating with EICrecon

- Everything mentioned here (except "Ongoing design work") is in master
- Currently being tested in conjunction with Takuya's timeframe splitter (`tkuma_timeframe_splitting`)
- Codesign hopefully converges this week
- I will cut a release once the codesign converges
- This release needs extra careful validation because it introduces some very large refactorings under the hood
- Incorporating the new release into EICrecon is possibly the only remaining prerequisite for merging Takuya's timeframe splitter

Thank you!