Workspace Factories — Old and New

Simon Heybrock

```
// Quiz: What does this print?
for (const auto &type :
     {"Workspace2D", "EventWorkspace", "MaskWorkspace"}) {
  auto parent =
      WorkspaceFactory::Instance().create(type, 1, 1, 1);
  auto ws = WorkspaceFactory::Instance().create(parent);
  std::cout << typeid(*ws).name() << '\n';
}
```

Old workspace factory

```cpp
#include "MantidAPI/WorkspaceFactory.h"
auto ws = WorkspaceFactory::Instance().create(
    "Workspace2D", 1, 3, 2);
auto out = WorkspaceFactory::Instance().create(
    ws, nHist, ysize + 1, ysize);
```

Shortcomings

- Need to specify X and Y length explicitly.
- Hidden conversions ⇒ cannot create `EventWorkspace`.
- 3 `int` parameters in a row.
- Very often multi-stage initialization of workspace necessary.
  - Set instrument.
  - Set spectrum numbers and spectrum mapping.
  - Set histogram data such as bin edges.

```cpp
#include "MantidDataObjects/WorkspaceCreation.h"
auto ws = DataObjects::create<T>(/* see below */);
```

New workspace factory

```cpp
#include "MantidDataObjects/WorkspaceCreation.h"
using namespace DataObjects;

auto ws = create<T>(IndexInfo,  Histogram);
auto ws = create<T>(Instrument, NumSpectra, Histogram);
auto ws = create<T>(Instrument, IndexInfo,  Histogram);
auto ws = create<T>(ParentWS);
auto ws = create<T>(ParentWS, Histogram);
auto ws = create<T>(ParentWS, NumSpectra, Histogram);
auto ws = create<T>(ParentWS, IndexInfo, Histogram);
```

- `T` gives the pointer type (`MatrixWorkspace` subclass).
- The held type is determined from input workspace and `T`.
- Workspace size set directly or via `Indexing::IndexInfo` to set spectrum numbers and mapping to detectors.
- ‘Histogram‘ argument determines whether created workspace contains point data or histogram data and at the same time provides a way to initialize the histograms.

Examples (1)

Create directly from instrument

Before:

```
auto ws = WorkspaceFactory::Instance().create(
    "Workspace2D", instrument->getDetectorIDs.size(), 3,
    2);
ws->setInstrument(instrument);
BinEdges binEdges{1.0, 2.0, 4.0};
for (size_t i = 0; i < ws->getNumberHistograms(); ++i)
  ws->setBinEdges(i, binEdges);
```

After:

```
auto ws = create<Workspace2D>(instrument,
                              BinEdges{1.0, 2.0, 4.0});
```

Examples (2a)

Same type as parent, same number of histograms, X copied

Before:

```
if (boost::dynamic_pointer_cast<EventWorkspace>(parent)) {
  goto someOtherSlide;
} else {
  auto ws = WorkspaceFactory::Instance().create(parent);
  for (size_t i = 0; i < ws->getNumberHistograms(); ++i)
    ws->setSharedX(i, parent->sharedX(i));
}
```

After:

```
auto ws = create<MatrixWorkspace>(*parent);
```

Examples (2b)

Same type as parent, same number of histograms, change X

Before:

```
if (boost::dynamic_pointer_cast<EventWorkspace>(parent)) {
  goto someOtherSlide;
} else {
  auto ws = WorkspaceFactory::Instance().create(
      parent, parent->getNumberHistograms(), 2, 2);
  Points points{1.5, 2.5};
  for (size_t i = 0; i < ws->getNumberHistograms(); ++i)
    ws->setPoints(i, points);
}
```

After:

```
auto ws =
    create<MatrixWorkspace>(*parent, Points{1.5, 2.5});
```

Examples (2c)

Same type as parent, change number of histograms, change X

Before:

```cpp
if (boost::dynamic_pointer_cast<EventWorkspace>(parent)) {
  goto someOtherSlide;
} else {
  auto ws = WorkspaceFactory::Instance().create(parent,
                                                 17, 2, 2);
  Points points{1.5, 2.5};
  for (size_t i = 0; i < ws->getNumberHistograms(); ++i)
    ws->setPoints(i, points);
}
```

After:

```cpp
auto ws = create<MatrixWorkspace>(*parent, 17,
                                   Points{1.5, 2.5});
```

Special case for example 2a

Create EventWorkspace from parent EventWorkspace

Before:

```
someOtherSlide:
const int size =
    static_cast<int>(parent->getNumberHistograms());
const int YLength = static_cast<int>(parent->blocksize());
// Make a brand new EventWorkspace
auto ws = boost::dynamic_pointer_cast<EventWorkspace>(
    API::WorkspaceFactory::Instance().create(
        "EventWorkspace", size, YLength + 1, YLength));
// Copy geometry over.
API::WorkspaceFactory::Instance().initializeFromParent(
    *parent, *ws, false);
for (int i = 0; i < size; ++i)
  ws->setSharedX(i, parent->sharedX(i));
```

After:

```
auto ws = DataObjects::create<MatrixWorkspace>(*parent);
// auto ws = DataObjects::create<EventWorkspace>(*parent);
```

Type conversions: Create `Workspace2D` from `EventWorkspace`

Undocumented feature of `WorkspaceFactory::create(parent)`

`EventWorkspace` is always converted to `Workspace2D`

- Need to provide this feature
- Need to make this more explicit

⇒ new helper class `HistoWorkspace` as base class of non-event `MatrixWorkspace` classes. Before:

```
auto ws = API::WorkspaceFactory::Instance().create(
    parent, nHist, ysize + 1, ysize);
```

After:

```
auto ws = DataObjects::create<API::HistoWorkspace>(
    *parent, nHist, BinEdges(ysize));
```

Compare (NOT dropping events):

```
auto maybeEventWs = create<API::MatrixWorkspace>(
    *parent, nHist, BinEdges(ysize));
```