MDEventWorkspace Code Overview

Janik Zikovsky
Jul 20, 2011

Class: MDEvent<nd>

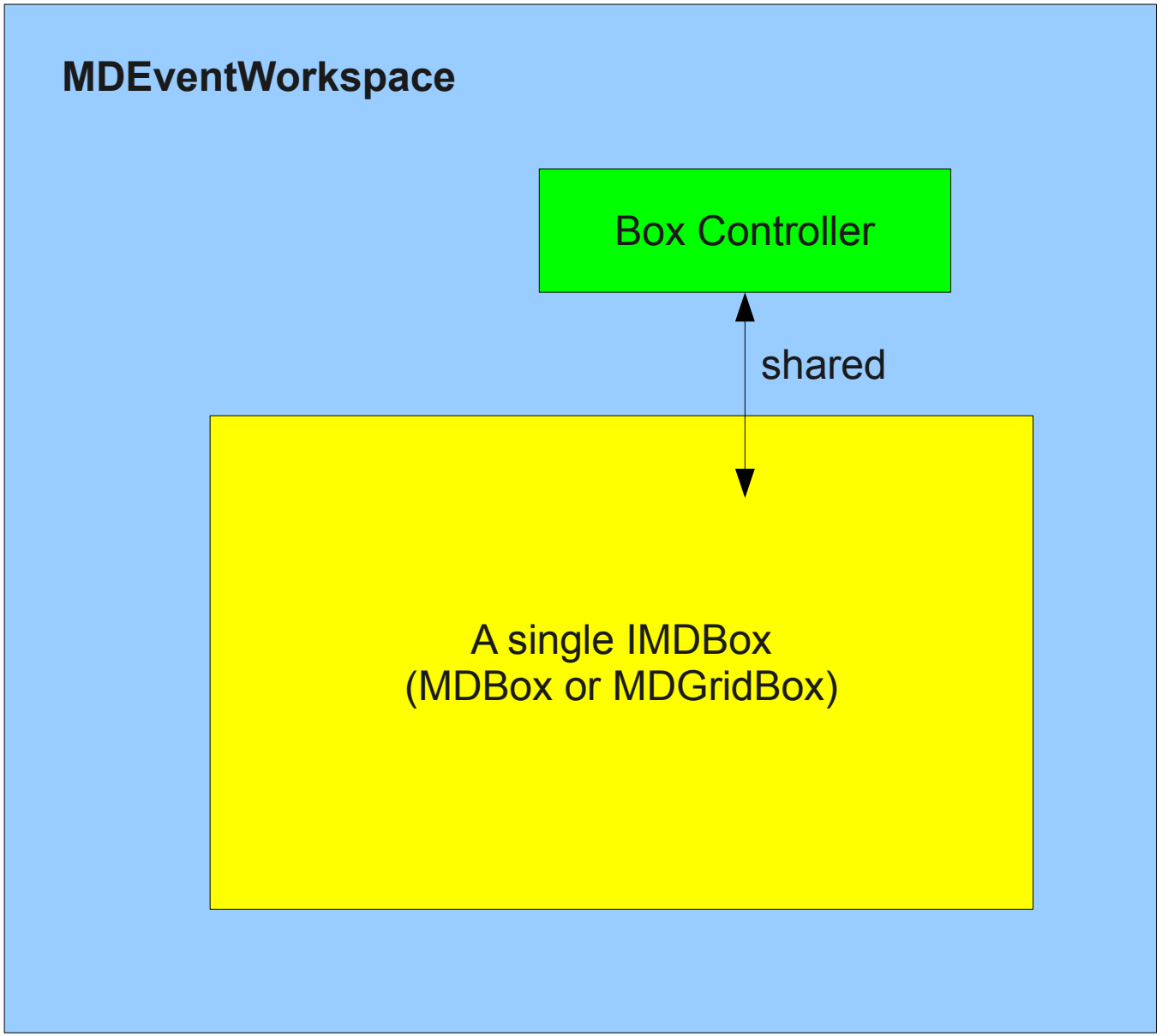
- Basic data type for a single event
- Templated for the number of dimensions, nd
  - This allows the use of a fixed-size array for the coordinates of the event, and other compiler optimizations
- Members:
  - center[nd] : coordinate of the event
  - signal and errorSquared.
- Can be subclassed if you need extra information.

IMDEventWorkspace

- A non-templated abstract interface with common functionality.
- **This is the type to passed around to algorithms** since it is a common type for all MDEventWorkspaces, e.g. WorkspaceProperty<IMDEventWorkspace>
  - The MDEventFactory.h file has a macro to use to cast to a particular MDEventWorkspace:
    - CALL_MDEVENT_FUNCTION(funcname, workspace)
    - See algorithms for examples.
- Holds:
  - A description of each of the *nd* dimensions.
  - Some common methods.

MDEventWorkspace<MDE,nd>

- The concrete workspace.
- Templated for *MDE* (typically MDEvent) and the number of dimensions *nd*.
- It basically contains only a box (of type IMDBox) which we will see next.
- Each MDEventWorskpace is associated with a **BoxController** instance that determines a lot of the behavior of the MDEW. We will see it later.


MDEventWorkspace
Box Controller
shared
A single IMDBox
(MDBox or MDGridBox)

IMDBox<MDE,nd>

- The real MDEvent storage units are MDBoxes.
- IMDBox is an abstract class at the base of MDBoxes.
- Contains:
  - The extents (min/max) in nd-dimensions.
  - A cached total signal/error for all the events in the box (very useful for speeding up some calculations)
  - A link to a BoxController. All boxes in a workspace have the same controller!

MDBox<MDE,nd>

- Subclass of IMDBox
- The simple “box” of events; it holds an unsorted list of events.
- **All events in the MDBox fit in its extents.**
- It is at the MDBox level that the file-based back-end will operate, saving/loading lists of events to file.
  - Methods: getEvents()

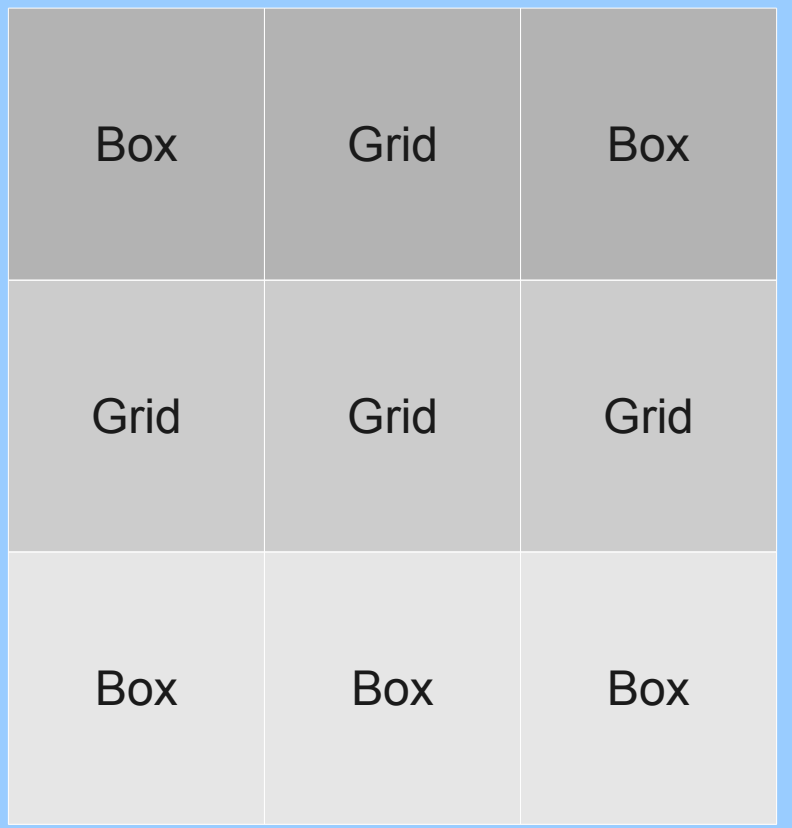
MDGridBox<MDE,nd>

- Subclass of IMDBox
- A special box – it contains *no events directly*. Instead, it holds a regular grid of sub-boxes (all of type IMDBox).
  - These sub-boxes are its *children*. Methods: getChild(), getNumChildren()
- The *BoxController* determines how it will get split.
- The MDGridBox is normally formed by passing a MDBox * to its *constructor*. The new MDGridBox *replaces* the old MDBox.

MDBox

MDEvent
MDEvent
MDEvent
...

MDGridBox



BoxController

- A BoxController pointer is shared by **ALL** MDBoxes and MDGridBoxes of a given workspaces.
- The BoxController determines how a MDEventWorskpace will *behave*.

BoxController: Splitting behavior

- The BoxController:
  - Has a threshold for determining how many events before splitting up a MDBox into a MDGridBox
    - get/setSplitThreshold()
  - Determines into how many children a MDBox will get split when forming a MDGridBox.
    - get/setSplitInto()
  - How deep splitting should go, a maximum recursion depth.
    - get/setMaxDepth()

BoxController: other uses

- It gives a unique ID to each box created.
  - This is used in file saving/loading.
- It tracks statistics of the number of boxes.
- In the future, it will be used to coordinate file-based back-end.
- **Basically anything the MDEventWorkspace needs to do.**
  - Instead of putting a pointer to the parent MDEventWorkspace in each MDBox contained within.

Using MDEventWorkspaces in Algorithms 1.

- Pass MDEventWorkspaces around as IMDEventWorkspace:

  declareProperty(WorkspaceProperty<IMDEventWorkspace>(...));

  IMDEventWorkspace_sptr ws = getProperty("InputWorkspace");

- Declare a method like:

  template<typename MDE, size_t nd> void SaveMDEW::doSave(typename MDEventWorkspace<MDE, nd>::sptr ws);

- Call that method using:

  CALL_MDEVENT_FUNCTION(this->doSave, ws);

Using MDEventWorkspaces in Algorithms 2.

- You can get at the base box of a MDEventWorkspace with:
  - IMDBox<MDE,nd> * box = ws->getBox();
  - Remember, all your code is templated by *MDE* event type and by *nd* dimensions, so these classes have to have the same types declared.
- A good way to go through the workspace is to use the MDBoxIterator (that's next).

MDBoxIterator

- Can be used to iterate through (all/part) of the boxes in a MDEventWorkspace.
  - Can be restricted to only go down to a given depth.
  - Can be set to return only “leaf” nodes, e.g. unsplit boxes, which will be MDBoxes.
    - Except if you limit to a max depth, in which case a leaf node might be a MDGridBox.
- Coming soon:
  - Restrict to boxes that touch a region of interest (ImplicitFunction)

Using MDBoxIterator

- Declare it with:
  - MDBoxIterator<MDE,nd> it(ws->getBox(), 1000 /*max depth */, false /*leaf Only*/);

```
while (true)
{
   // Start at the first box returned
   IMDBox<MDE,nd> * box = it.getBox();
   // Do stuff with the box
   // Until you run out of boxes
   if (!it.next()) break;
}
```

Alternate Algorithms

- Another way to go through boxes is to use recursive calls. IMDBox::centerpointBin() is a good example of that.
  - The method will “bin” all the events that fit within the extents of a MDBin object passed to it.
  - That method is implemented differently by the subclasses:
    - MDGridBox tries to find which of its children might be in the area and only calls centerpointBin() on *those* children.
      - If the child is also MDGridBox, then the call is made recursively.
      - Boxes that completely fit within a bin use the cached total signal value.
    - MDBox simply loops through all events and sums those that do fit within the bin.