FLIP-2: Extending Window Function Metadata

Status

Discussion thread	https://www.mail-archive.com/dev%40flink.apache.org/msg09285.html
Vote thread	
JIRA	**FLINK-4997** - Getting issue details... STATUS
Release	1.3

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

Right now, in Flink a WindowFunction does not get a lot of information when a window fires.

The signature of WindowFunction is this:

```
public interface WindowFunction<IN, OUT, KEY, W extends Window> extends Function, Serializable {
    void apply(KEY key, W window, Iterable<IN> input, Collector<OUT> out);
}
```

i.e , the user code only has access to the key for which the window fired, the window for which we fired and the data of the window itself. In the future, we might like to extend the information available to the user function. We initially propose this as additional information:

- Why/when did the window fire. Did it fire on time, i.e. when the watermark passed the end of the window. Did it fire early because of a speculative early trigger or did it fire on late-arriving data.
- How many times did we fire before for the current window. This would probably be an increasing index, such that each firing for a window can be uniquely identified.

Public Interfaces

- New class ProcessWindowFunction with extensible interface
- New overload of WindowedStream.apply() that takes the new window function

Proposed Changes

New ProcessWindowFunction

We propose to add a new window function that has an extensible interface so that we can easily add more meta information in the future. The initial signature would be this:

```
public abstract class ProcessWindowFunction <IN, OUT, KEY, W extends Window> implements Function {

    public abstract void process(KEY key, Context ctx, Iterable<IN> elements, Collector<OUT> out) throws
Exception;

    public abstract class Context {
        public abstract W window();
    }
}
```

The context object has the same information as *WindowFunction* but can be extended in the future.

Required Changes

Internally the WindowFunction that a user uses is already decoupled from the windowing internals. Internally an InternalWindowFunction is used. Normally, an InternalWindowFunction wraps a WindowFunction. We propose to change InternalWindowFunction to match the newly proposed interface. It can still be used to present the current interface to WindowFunction while we can now also support the new ProcessWindowFunction. This is necessary because we cannot break the current API around WindowFunction.

Future Changes based on this FLIP

Adding a Window-Firing Counter

This requires to extend the Context given to the *ProcessWindowFunction* like this:

```
public abstract class Context {
   public abstract W window();
   public abstract int id();
}
```

where the new *id()* method gives the id/count of the current window firing.

For keeping track of the count, the WindowOperator will need to keep additional state. Right now, the window elements are kept in a state that is accessed using this state descriptor:

```
protected final StateDescriptor<? extends AppendingState<IN, ACC>, ?>
windowStateDescriptor;
```

The additional state can be a simple counter, such as:

```
protected final ValueStateDescriptor<Integer> windowIdDescriptor;
```

that will have to be updated when firing windows and that will have to be garbage collected when the window state is being garbage collected.

Adding a Window-Firing "Reason"

This requires to extend the Context given to the *ProcessWindowFunction* like this:

```
public abstract class Context {
   public abstract W window();
   public abstract int id();
   public abstract FiringInfo firingInfo();
}
```

FiringInfo would be an enum that *EARLY*, *ON_TIME*, *LATE*. That information could be derived from changes in the watermark and does not require keeping extra state in the window operator.

Compatibility, Deprecation, and Migration Plan

- The new interface will not impact users of the existing *WindowFunction*
- We can deprecate the old *WindowFunction* in Flink 2.0, if we want

Test Plan

Describe in few sentences how the FLIP will be tested. We are mostly interested in system tests (since unit-tests are specific to implementation details). How will we know that the implementation works as expected? How will we know nothing broke?

Rejected Alternatives

If there are alternative ways of accomplishing the same thing, what were they? The purpose of this section is to motivate why the design is the way it is and not some other way.