Consumer API changes

New consumer API

```
  /**
   *  Create a list of MessageStreams for each topic.
   *
   *  @param topicCountMap  a map of (topic, #streams) pair
   *  @param decoder Decoder to decode each Message to type T
   *  @return a map of (topic, list of KafkaStream) pairs.
   *          The number of items in the list is #streams. Each stream supports
   *          an iterator over message/metadata pairs.
   */
  def createMessageStreams[T](topicCountMap: Map[String,Int],
                              decoder: Decoder[T] = new DefaultDecoder)
    : Map[String,List[KafkaStream[T]]]


  /**
   *  Create a list of message streams for all topics that match a given filter.
   *
   *  @param topicFilter Either a Whitelist or Blacklist TopicFilter object.
   *  @param numStreams Number of streams to return
   *  @param decoder Decoder to decode each Message to type T
   *  @return a list of KafkaStream each of which provides an
   *          iterator over message/metadata pairs over allowed topics.
   */
  def createMessageStreamsByFilter[T](topicFilter: TopicFilter,
                                      numStreams: Int = 1,
                                      decoder: Decoder[T] = new DefaultDecoder)
    : Seq[KafkaStream[T]]
```

Questions/discussion

What is a `TopicFilter`?

`TopicFilter` can be either a whitelist or a blacklist. e.g.,

- Example of a whitelist `TopicFilter`: `new Whitelist("white.*")`
- Example of a blacklist `TopicFilter`: `new Blacklist("black.*")`

Although Java regex allows you to specify anything with a single regex (i.e., you don't really need a blacklist option per se), negating a whitelist in Java regex is clumsy. It is convenient to be able to easily specify a whitelist and blacklist. Although right now `TopicFilter` supports only one of whitelist /blacklist in future we may want to support a chain of filters to do more elaborate topic selection.

What is a `MessageAndMetadata`?

```
case class MessageAndMetadata[T](message: T, topic: String = "", offset: Long = -1L)
```

The `KafkaStream[T]`'s iterator is a `ConsumerIterator[T]` which is an iterator over `MessageAndMetadata[T]` objects.

Can we eliminate the need for two methods in the API? Also, providing a topic-count-map in the `createMessageStreams` API is burdensome. Can we get rid of that?

If we don't support the one-shot approach of creating multiple streams for multiple topics, then the most obvious alternative is:

```
  def createMessageStreams[T](topic: String, numStreams: Int,
                              decoder: Decoder[T] = new DefaultDecoder)
    : Seq[KafkaStream[T]]
```

Advantages

- Simpler, more intuitive API which is consistent with the `createMessageStreamsByFilter` API as well.
- It may be possible to combine the `createMessageStreams` and `createMessageStreamsByFilter` calls into one API. This would need to be fleshed out in some detail, but we could have a higher-level `Topic` class that can either be a static topic, or a `TopicFilter`. Another advantage of this is that the high-level `Topic` class it could do things like validate topic names. However, the consumer code would need to explicitly call (`new Topic(new Whitelist("white.*")`) or (`new Topic("topicname")`) but that does not seem so bad.

Disadvantages

- The above also reveals an advantage of creating multiple streams for multiple topics at one-shot . If you want to create multiple streams for multiple topics, then you would need to make a `createMessageStreams` call for each topic, which would trigger one rebalance for each topic. With the one-shot call (which receives atopic-count-map), only one rebalance (for all topics) will be required.
- Not a disadvantage, but additional work: the consumer connector code is currently broken with respect to supporting multiple calls to `createMessageStreams` on the same connector object. For example, the `consumerIdString` is per connector object, and not per call. There are a bunch of other global variables that may need to become per-call instances. Anyway, the point is that we would need to completely fix that if we want to deprecate the option to provide a topic-count-map.

Impact to clients

Client code will need to change, since the current pattern of:

```
for (message <- stream) {
  // process(message)
}
```

will change to:

```
for (msgAndMetadata <- stream) {
  // processMessage(msgAndMetadata.message)
  // can also access msgAndMetadata.offset, topic, etc. if appropriate
}
```

Existing API

```
  /**
   *  Create a list of MessageStreams for each topic.
   *
   *  @param topicCountMap  a map of (topic, #streams) pair
   *  @return a map of (topic, list of  KafkaMessageStream) pair. The number of items in the
   *          list is #streams. Each KafkaMessageStream supports an iterator of messages.
   */
  def createMessageStreams[T](topicCountMap: Map[String,Int],
                              decoder: Decoder[T] = new DefaultDecoder)
    : Map[String,List[KafkaMessageStream[T]]]
```

References

1. https://issues.apache.org/jira/browse/KAFKA-249
2. Mailing list discussion