KIP-276 Add StreamsConfig prefix for different consumers

Status

Current state: *released in 2.0*

Discussion thread: here

JIRA: JIRA 6657

Motivation

Kafka Streams allows to pass in different configs for different clients by prefixing the corresponding parameter with `producer.` or `consumer.`.

However, Kafka Streams internally uses multiple consumers, (1) the main consumer (2) the restore consumer and (3) the global consumer

For some use cases, it's required to set different configs for different consumers. Thus, we should add three new prefixes for main, restore and global consumer.

We might also consider to extend `KafkaClientSupplier` and add a `getGlobalConsumer()` method.

Public Interfaces

The changes affected classes including **StreamsConfig** and **KafkaClientSupplier**. Existing APIs look like below:

StreamsConfig.java

```
// Prefix setter for consumer config
public static final String CONSUMER_PREFIX = "consumer.";
public static String consumerPrefix(final String consumerProp) {
    return CONSUMER_PREFIX + consumerProp;
}
...
// API to get consumer configs
public Map<String, Object> getConsumerConfigs(final String groupId, final String clientId);
public Map<String, Object> getRestoreConsumerConfigs(final String clientId);
```

KafkaClientSupplier.java

```
// Current consumer APIs:
Consumer<byte[], byte[]> getConsumer(final Map<String, Object> config);
Consumer<byte[], byte[]> getRestoreConsumer(final Map<String, Object> config);
```

Currently, user could use function **consumerPrefix()** to add specific config for stream consumers. There are no way to differentiate configuration for main consumer, restore consumer and global consumer, which may have different behavior from base settings.

Proposed Changes

We first add APIs in the **StreamsConfig** class to generate prefix for main consumer, restore consumer and global consumer, and functions to retrieve them as override configs:

StreamsConfig.java

```
// New Prefixes and setters for main consumer, restore consumer and global consumer
public static final String MAIN_CONSUMER_PREFIX = "main.consumer.";
public static final String RESTORE_CONSUMER_PREFIX = "restore.consumer.";
public static final String GLOBAL_CONSUMER_PREFIX = "global.consumer.";

public static String mainConsumerPrefix(final String consumerProp) {
    return MAIN_CONSUMER_PREFIX + consumerProp;
}
public static String restoreConsumerPrefix(final String consumerProp) {
    return RESTORE_CONSUMER_PREFIX + consumerProp;
}
public static String globalConsumerPrefix(final String consumerProp) {
    return GLOBAL_CONSUMER_PREFIX + consumerProp;
}
...
// API to get different types of consumer configs.
public Map<String, Object> getMainConsumerConfigs(final String clientId);
public Map<String, Object> getRestoreConsumerConfigs(final String clientId);
public Map<String, Object> getGlobalConsumerConfigs(final String clientId);
```

And add a new public API in the KafkaClientSupplier class to get global consumer during init.

KafkaClientSupplier.java

```
// New API for global consumer:
Consumer<byte[], byte[]> getGlobalConsumer(final Map<String, Object> config);
```

By deprecating the **getConsumerConfigs** function (from now user should use **getMainConsumerConfigs** instead), rewriting the **getRestoreConsumerConfigs** function and adding the **getGlobalConsumerConfigs** function, if one user uses **restoreConsumerPrefix** or **globalConsumerPrefix** when adding new configurations, the configs shall overwrite base consumer config. If one just wants to change main consumer behavior without actually affecting other consumers, using **mainConsumerPrefix** would make sure the change only apply to main consumer. If not specified, main consumer, restore consumer and global consumer shall share the same config with base consumer.

Example

if user writes:

Consumer Config

```
consumer.max.poll.record = 5
main.consumer.max.poll.record = 100
restore.consumer.max.poll.record = 50
```

During initialization, consumers would get:

consumer type	max.poll.record	Reason
main consumer	100	Target assignment with "main.consumer" prefix
restore consumer	50	Get override config 50 by prefixing "restore-consumer"
global consumer	5	Since no "global.consumer" prefix is used, default config will be applied.

Compatibility, Deprecation, and Migration Plan

There is no backward compatibility issue as we are not deprecating any public API, and the underlying change should be transparent to the user.

Rejected Alternatives

None.