Platform Configuration Guide

Terminology

The following terminology will be used throughout this guide;

Term	Definition
Attribute	An application or component setting whose value may be derived from configuration
Property	A key string in the config.properties file for the application or component
Current Property	A property currently intended for use in configuring an attribute
Replaced Property	A property, previously used for configuring an attribute, that is still supported by the code although the attribute should be configured through a different current property
Deprecated Property/Attribute	A property, previously used for configuring an attribute, that is still supported by the code, although the attribute should now be configured through a different current property.
Defunct Property	A property, previously used for configuring an attribute, that is no longer supported by the code.

Annotations

The following annotations are available for annotating configuration classes

Annotation	Package	Fields	Target	Usage Notes
@Config	com.proofpoint.configuration	• value : String	Method	• Specifies the current property for an attribute • Should be placed on the setter for the attribute
@LegacyConfig	com.proofpoint.configuration	• value : String array • replacedBy : String	Method	• Provides replaced properties for an attribute • Should be placed on a setter • If the settter's attribute name is no longer correct, the replacedBy field can be used to provide linkage to the current property
@Deprecated	java.lang	-	Method (in this case)	• Marks an attribute and all associated properties as deprecated • Placed an a getter and all associated setters

@DefunctConfig	com.proofpoint.configuration	• value : String array	Type	• Specifies defunct configuration properties that used to be used to configure a class • Placed on the class

Usage Examples

Initial Config creation

1. An attribute is defined by a setter function annotated witih @Config (“setX()” defines the attribute “X”).
2. An attribute must have exactly one getter/is function. This function for attribute X must be named getX/isX.

```
class IdConfig
{
    private int id;

    public int getId()
    {
        return id;
    }

    @Config(“my.id”)
    public void setId(int id)
    {
        this.id = id;
    }
}
```

Or

```
class EnabledConfig
{
    private boolean enabled;

    public boolean isEnabled()
    {
        return enabled;
    }

    @Config(“i.am.enabled”)
    public void setEnabled(boolean enabled)
    {
        this.enabled = enabled;
    }
}
```

Changing the property used to configure an attribute

1. A new current property name should be selected
2. The setter annotated with @Config should be updated with the new property name. The replaced property name should be moved into a @LegacyConfig annotation on the same setter (creating if required)
3. Any setters annotated with @LegacyConfig, with replacedBy referencing the replaced property name should be updated to be replacedBy the new current property name

E.g. For the following original class, where we want to change the name of the property “my.id” to “id-number”

```
class IdConfig
{
    private int id;

    public int getId()
    {
        return id;
    }

    @Config(“my.id”)
    public void setId(int id)
    {
        this.id = id;
    }
}
```

Result:

```
class IdConfig
{
    private int id;

    public int getId()
    {
        return id;
    }

   @Config(“id-number”)
   @LegacyConfig(“my.id”)
   public void setId(int id)
    {
        this.id = id;
    }
}
```

Or, for the following original class, where we want to change the name of the property “timeout-ms” to “timeout”

```
class TimeConfig
{
    private int duration;

    public int getDurationMs()
    {
        return duration;
    }

    @Config(“timeout-ms”)
    @LegacyConfig(“timeout-milliseconds”)
    public void setDurationMs(int millis)
    {
        this.duration = millis;
    }

    @Deprecated
    @LegacyConfig(value = ”timeout-secs”, replacedBy = “timeout-ms”)
    public void setDurationInSeconds(int secs)
    {
        setDurationMs(secs * 1000);
    }
}
```

Result:

```
class TimeConfig
{
    private int duration;

    public int getDurationMs()
    {
        return duration;
    }

    @Config(“timeout”)
    @LegacyConfig(value = {“timeout-milliseconds”, “timeout-ms”})
    public void setDurationMs(int millis)
    {
        this.duration = millis;
    }

    @Deprecated
    @LegacyConfig(value = ”timeout-secs”, replacedBy = “timeout”)
    public void setDurationInSeconds(int secs)
    {
        setDurationMs(secs * 1000);
    }
}
```

Changing the property and name of an attribute

E.g. For the following original class, where we want to replace duration in seconds (attribute DurationInSeconds) with duration in milliseconds (attribute DurationMs)

```
class TimeConfig
{
    private int duration;

    public int getDurationInSeconds()
    {
        return duration;
    }

    @Config(”timeout-secs”)
    public void setDurationInSeconds(int secs)
    {
        setDuration(secs);
    }
}
```

1. A new current property name and attribute name should be selected
2. The getter should be updated to use the new attribute name.
3. A new setter, with the new attribute name, should be created. This setter should be annotated with @Config, with the new current property name
4. The setter that used to be annotated with @Config must be instead annotated with @LegacyConfig; the replacedBy field should be set to the new property name
5. The old setter should also be annotated with @Deprecated.
6. Any setters that used to be annotated with @LegacyConfig, with replacedBy referencing the replaced property name should be updated to be replacedBy the new current property name

Result:

```
class TimeConfig
{
    private int duration;

    public int getDurationMs()
    {
        return duration;
    }

    @Config(“timeout-ms”)
    public void setDurationMs(int millis)
    {
        this.duration = millis;
    }

    @Deprecated
    @LegacyConfig(value = ”timeout-secs”, replacedBy = “timeout-ms”)
    public void setDurationInSeconds(int secs)
    {
       setDurationMs(secs * 1000);
    }
}
```

Or, after a further change to replace Millis with Nanos:

```
class TimeConfig
{
    private int duration;

    public int getDurationNanos()
    {
        return duration;
    }

    @Config(“timeout-nanos”)
    public void setDurationNanos(int nanos)
    {
        this.duration = nanos;
    }

    @Deprecated
    @LegacyConfig(value = “timeout-ms”, replacedBy = “timeout-nanos”)
    public void setDurationMs(int millis)
    {
        setDurationNanos(millis * 1000000);
    }

    @Deprecated
    @LegacyConfig(value = ”timeout-secs”, replacedBy = “timeout-nanos”)
    public void setDurationInSeconds(int secs)
    {
        setDurationMs(secs * 1000);
    }
}
```

Changing the type (but not the name) of an attribute

1. A new current property name should be selected
2. The getter should be updated to return the new type.
3. A new setter, with the same attribute name, taking the new type should be created. This setter should be annotated with @Config, with the new current property name
4. The setter that used to be annotated with @Config must be instead annotated with @LegacyConfig, and should be annotated with @Deprecated
5. Any setters that used to be annotated with @LegacyConfig, with replacedBy referencing the replaced property name should be updated

to be replacedBy the new current property name

E.g. For an original class, replacing int with string for Id:

```
class IdConfig
{
    private int id;

    public int getId()
    {
        return id;
    }

    @Config(“my.id”)
    public void setId(int id)
    {
        this.id = id;
    }
}
```

Change to

```
class IdConfig
{
    private String id;

    public String getId()
    {
        return id;
    }

    @Deprecated
    @LegacyConfig(value = “my.id”)
    public void setId(int id)
    {
       this.id = Integer.toString(id);
    }

    @Config(“my.id-string”)
    public void setId(String id)
    {
        this.id = id;
    }
}
```

Or for original class, replacing int with Duration for Duration

```
class TimeConfig
{
    private int duration;

    public int getDuration()
    {
        return duration;
    }

    @Config(“timeout-ms”)
    public void setDuration(int millis)
    {
        this.duration = millis;
    }

    @Deprecated
    @LegacyConfig(value = ”timeout-secs”, replacedBy = “timeout-ms”)
    public void setDurationInSeconds(int secs)
    {
        setDuration(secs * 1000);
    }
}
```

Replace with

```
class TimeConfig
{
    private Duration duration;

    public Duration getDuration()
    {
        return duration;
    }

    @Deprecated
    @LegacyConfig(value = “timeout-ms”)
    public void setDuration(int millis)
    {
       this.duration = new Duration(millis, MILLIS);
    }

    @Deprecated
    @LegacyConfig(value = ”timeout-secs”, replacedBy = “timeout”)
    public void setDurationInSeconds(int secs)
    {
        setDuration(secs * 1000);
    }

    @Config(“timeout”)
    public void setDuration(Duration duration)
    {
        this.duration = duration;
    }
}
```

Deprecating a configuration property/attribute

E.g. For the following configuration class, where we no longer want Id (“my.id”) to be configurable – it is no longer used in the code..

```
class IdConfig
{
    private int id;

    public int getId()
    {
        return id;
    }

    @Config(“my.id”)
    public void setId(int id)
    {
        this.id = id;
    }
}
```

1. Mark the getter and all setters as @Deprecated.

Result:

```
class IdConfig
{
    private int id;

    @Deprecated
    public int getId()
    {
        return id;
    }

    @Deprecated
    @Config(“my.id”)
    public void setId(int id)
    {
        this.id = id;
    }
}
```

Or for original class:

```
class TimeConfig
{
    private Duration duration;

    public Duration getDuration()
    {
        return duration;
    }

    @Deprecated
    @LegacyConfig(names = “timeout-ms”)
    public void setDuration(int millis)
    {
        this.duration = new Duration(millis, MILLIS);
    }

    @Deprecated
    @LegacyConfig(value = ”timeout-secs”, replacedBy = “timeout”)
    public void setDurationInSeconds(int secs)
    {
        setDuration(secs * 1000);
    }

    @Config(“timeout”)
    public void setDuration(Duration duration)
    {
        this.duration = duration;
    }
}
```

Result:

```
class TimeConfig
{
    private Duration duration;

    @Deprecated
    public Duration getDuration()
    {
        return duration;
    }

    @Deprecated
    @LegacyConfig(“timeout-ms”)
    public void setDuration(int millis)
    {
        this.duration = new Duration(millis, MILLIS);
    }

    @Deprecated
    @LegacyConfig(value = ”timeout-secs”, replacedBy = “timeout”)
    public void setDurationInSeconds(int secs)
    {
        setDuration(secs * 1000);
    }

    @Deprecated
    @Config(“timeout”)
    public void setDuration(Duration duration)
    {
        this.duration = duration;
    }
}
```

Defuncting a replaced configuration property

Once a property has been replaced for a reasonable length of time, it is reasonable to defunct it – remove support for it from the code. In order to do this

1. Remove the defunct property from any @LegacyConfig annotation in which it occurs. If this removes the last annotation from the property, remove the entire annotation. If this removes all config-related annotations from the setter function, consider removing the setter function.
2. Add the defunct property to the class annotation @DefunctConfig, creating it if required.

E.g. To defunct the property (“my.id”) in the following class

```
class IdConfig
{
    private String id;

    public String getId()
    {
        return id;
    }

    @Deprecated
    @LegacyConfig(“my.id”)
    public void setId(int id)
    {
        this.id = Integer.toString(id);
    }

    @Config(“my.id-string”)
    public void setId(String id)
    {
        this.id = id;
    }
}
```

Result:

```
@DefunctConfig(“my.id”)
class IdConfig
{
    private String id;

    public String getId()
    {
        return id;
    }

    // Deleted 5 lines

    @Config(“my.id-string”)
    public void setId(String id)
    {
        this.id = id;
    }
}
```

Defuncting a deprecated configuration attribute

Once a property has been deprecated for a reasonable length of time, it is reasonable to defunct it – remove support for it from the code. In order to do this

1. For each function related to the attribute that is annotated as @Deprecated:
    a. Remove the function
    b. Add all property names from the @Config or @LegacyConfig annotations to the class @DefunctConfig annotation

E.g. To defunct the deprecated Duration attribute in the following class

```
class TimeConfig
{
    private Duration duration;
    private int id;

    public int getId()
    {
        return id;
    }

    @Config(“id”)
    public void setId(int id)
    {
        this.id = id;
    }

    @Deprecated
    public Duration getDuration()
    {
        return duration;
    }

    @Deprecated
    @LegacyConfig(names = “timeout-ms”)
    public void setDuration(int millis)
    {
        this.duration = new Duration(millis, MILLIS);
    }

    @Deprecated
    @LegacyConfig(value = ”timeout-secs”, replacedBy = “timeout”)
    public void setDurationInSeconds(int secs)
    {
        setDuration(secs * 1000);
    }

    @Deprecated
    @Config(“timeout”)
    public void setDuration(Duration duration)
    {
        this.duration = duration;
    }
}
```

Result:

```
@DefunctConfig({“timeout”, “timeout-ms”, “timeout-secs”})
class TimeConfig
{
    // deleted 1 line
    private int id;

    public int getId()
    {
        return id;
    }

    @Config(“id”)
    public void setId(int id)
    {
        this.id = id;
    }

    // Deleted 26 lines
}
```

Splitting a configuration property

It is permissible to configure more than one attribute through a single property. E.g. In the following class, all 3 timeout attributes are configured through the “timeout” property.

```
class ConnectionConfig
{
    private int readTimeout;
    private int writeTimeout;
    private int connectTimeout;

    public int getReadTimeout()
    {
        return readTimeout;
    }

    @Config(”timeout”)
    public void setReadTimeout(int secs)
    {
        this.readTimeout = secs;
    }

    public int getWriteTimeout()
    {
        return writeTimeout;
    }

    @Config(”timeout”)
    public void setWriteTimeout(int secs)
    {
        this.writeTimeout = secs;
    }

    public int getConnectTimeout()
    {
        return connectTimeout;
    }

    @Config(”timeout”)
    public void setConnectTimeout(int secs)
    {
        this.connectTimeout = secs;
    }
}
```

In order to split the timeout property, so that, e.g. The connect timeout was configurable independently of the read/write timeouts:

1. Select new property names for each of the resulting property groups (note that the old property name MUST be replaced for all attributes)
2. For each attribute, follow the instructions above to replace its property name

Result, when splitting connectTimeout (new “connect-timeout”) from read/writeTimeout (new “activity-timeout”) for the above class

```
class ConnectionConfig
{
    private int readTimeout;
    private int writeTimeout;
    private int connectTimeout;

    public int getReadTimeout()
    {
        return readTimeout;
    }

    @Config(”activity-timeout”)
    @LegacyConfig(names = “timeout”)
    public void setReadTimeout(int secs)
    {
        this.readTimeout = secs;
    }

    public int getWriteTimeout()
    {
        return writeTimeout;
    }

    @Config(”activity-timeout”)
    @LegacyConfig(names = “timeout”)
    public void setWriteTimeout(int secs)
    {
        this.writeTimeout = secs;
    }

    public int getConnectTimeout()
    {
        return connectTimeout;
    }

    @Config(”connect-timeout”)
    @LegacyConfig(names = “timeout”)
    public void setConnectTimeout(int secs)
    {
        this.connectTimeout = secs;
    }
}
```

Testing

Two types of tests are required of all configuration classes, three if there are any deprecated properties.

Default Test

Verify the default values for all properties. Doing so ensures no one unintentionally changes a default in the code. For example for a property with setter "setTtl" and default of 1 hour:

```
@Test
public void testDefaults()
{
    ConfigAssertions.assertRecordedDefaults(ConfigAssertions.recordDefaults(StoreConfig.class)
            .setTtl(new Duration(1, TimeUnit.HOURS)));
}
```

Round Trip Test

Verify that values from getters are the same as those passed to the setter.

```
@Test
public void testExplicitPropertyMappings()
{
    Map<String, String> properties = new ImmutableMap.Builder<String, String>()
            .put("store.ttl", "2h")
            .build();

    StoreConfig expected = new StoreConfig()
            .setTtl(new Duration(2, TimeUnit.HOURS));

    ConfigAssertions.assertFullMapping(properties, expected);
}
```

Deprecated Properties Test

Verify that the new property can set an equivalent value as the legacy property, and that the two are properly tied together with a @LegacyConfig annotation described above.

```
@Test
public void testDeprecatedProperties()
{
    Map<String, String> currentProperties = new ImmutableMap.Builder<String, String>()
            .put("store.ttl", "1h")
            .build();

    Map<String, String> oldProperties = new ImmutableMap.Builder<String, String>()
            .put("store.ttl-in-ms", "3600000")
            .build();

    ConfigAssertions.assertDeprecatedEquivalence(StoreConfig.class, currentProperties,
oldProperties);
```