Developing with XFP

by Flavio Tordini

1. Writing Your Own Filter

It is very easy to write your own filter:

- Create a Java class that extends `org.xfp.Filter`.

```
public class MyFilter extends Filter {
  ...
}
```

- For each property, write a setter method. The setter method must be a public void method that takes a single argument. The name of the method must begin with `set`, followed by the property name, with the first character of the name in uppercase, and the rest in lowercase. That is, to support an property named `type` you create a method `setType`. Depending on the type of the argument, XFP will perform some conversions for you (refer to the BeanUtils documentation).

```
/**
 * Sets my property.
 * @param type The property to set
 * @xfp.property.required
 */
public void setMyProperty(String myProperty) {
    this.myProperty = myProperty;
}
```

Properties can also be `mapped`, i.e. be in a key/value format:

```
/**
 * Sets my mapped property.
 * @param propertyname Name of the property to set
 * @param propertyvalue Value for the property
 */
public void setMyMappedProperty(java.lang.String propertyname, Object propertyvalue)
    this.myMappedProperties.put(propertyname, propertyvalue);
}
```

- Optionally, write a public void `initialize` method, with no arguments. Use this method to initialize any resource used by the filter. This method will be called after property values have been setted.

```
/**
```

```
 * @see org.xfp.Filter#initialize()
 */
public void initialize() throws Exception {
    // your logic here
}
```

- Write one or more public `execute` methods returning any type (but primitives) or void, with one or zero arguments (of any type but primitives), that throws Exception. These methods implement the filter itself. The type of the method argument is the input data type, while the returned type is the output type. If an `execute` method has no argument, the filter will not use any input data (we may call it a *Source*). This kind of filter is suitable to be a root filter. If the method returns void, the filter will have no output (a *Sink*). Such a filter will always be a leaf in the filter tree. All methods must return the same type.

```
/**
 * Execute this filter.
 * @return The filter output.
 */
public MyOutputType execute() throws Exception {
    // your logic here
}
```

Note:

Because of its variable signature, the `execute` method is not declared in `org.sourceforge.xfp.Filter`. It is discovered at runtime using Java reflection. For this reason the compiler will not throw any error if you do not declare any `execute` method, but XFP will.

- Write a public void `reset` method, with no arguments. Use this method to reset any property to its default value.

```
/**
 * @see org.xfp.Filter#reset()
 */
public void reset() {
    myProperty = null;
    myMappedProperties.clear();
}
```

You can create filters that accept multiple input types by overloading the `execute` method. Try supporting as much data types as possible, so that users can easily chain filters with matching data types.

If a filter `execute` method returns null, children filter will not be executed.

See `org.xfp.filters.*` for some real-world implementations.

2. Writing Your Own Component

XFP uses a subset of the Avalon Framework in order to manage the attribute lifecycle. However, the support of the Avalon Framework is not complete.

- Create a Java class that optionally implements one or more of the following interfaces:
  - `org.apache.avalon.framework.parameters.Parameterizable`
  - `org.apache.avalon.framework.activity.Initializable`
  - `org.apache.avalon.framework.activity.Startable`
  - `org.apache.avalon.framework.activity.Disposable`

See `org.xfp.components.*` for some concrete examples.