Upgrading styles with GeoAPI Interfaces

Motivation:	*Upgrading styles for more functionalities*
Contact:	Johann Sorel
Tagline:	Style Upgrade

Description

Currently, GeoTools styles package is based on OGC SLD 1.0.0.

This proposal will upgrade GeoTools style package to follow newer specifications :

- OGC Symbology Encoding 1.1
- SLD 1.1.0 (only the style class)
- ISO 19117 "portrayal"

JavaDoc of the GeoAPI interfaces :

> http://geoapi.sourceforge.net/snapshot/javadoc/index.html
> package :
> org.opengis.style

A Basic Implementation :

> To tests on the go-renderer, I committed a basic implementation in the unsupported GO module.
> package :
> org.geotools.styling2

What is new :

- styles can be expressed in display units (pixels for screen) or in land units like meters, foots ...
- styles interfaces are more accurate, some interfaces have been splitted (like graphic that can be graphicFill or GraphicStroke with more details)
- styles handle inline objects.
- Java5 Generics and Enumerations helps do avoid mistakes
- symbolizers are immutable
- FeatureTypeStyle is mutable and has add/remove listeners

Some changes may be more or less a problem like :

- ~~arrays[] replaced by lists or collections~~
- ~~raster symbolizer functions~~

> **Solution for array versus collection**
> A solution is to rename GeoAPI " getRules() " method to " rules() " .

> **Raster function test**
> Jody provided a version of Categorize function and I tryed to use it.
> It looks usable, we can get back all attributs from the function.
> We can also extends the function with a special "categorizeFunction" interface
> which provide more convinient methods to grab parameters.

The GeoAPI interfaces are new and have never been implemented before, if while implementing some interface raise problems then a solution is to backport changes in GeoAPI.

API

Style Creation

We will be adding additional methods as needed to the GeoTools StyleFactoryImpl. When we are happy with the result we will create a GeoAPI StyleFactory with the needed methods.

```
class StyleFactoryImpl extends AbstractStyleFactory implements StyleFactory2,
                                                               /**
org.opengis.style.StyleFactory*/ {
    ...
    /**
     * Create an LineSymbolizer using stroke to draw the indicated geometry.
     * @param stoke Description of how to draw the line segments
     * @param geometryExpression Expression evaluating to a Geometry.
     *        (For SLD 1.1 documents this should be a PropertyName expression)
     * @since 2.6.0
     */
    LineSymbolizer lineSymbolizer( Stroke stroke, Expression geometryExpression );

    Rule rule();
    /**
     * ...
     * @param symbolizers List of symbolizers to use; these symbolizers will be copied
into
     *                    an internal list so we can issue events.
     * ...
     */
    Rule rule( Description description, Filter filter, List<Symbolizer> symbolizers,
LegendGraphic, double minScale, double maxScale, boolean isElse );

    ...

}
```

We will be using single words (no "create" prefix) similar to FilterFactory. We must be careful to provide each and every parameter during construction of the immutable objects. For mutable objects we will have two methods; one taking only the required parameters; and one taking each and every parameter.

Integration with existing Interfaces

We are making use of type narrowing to allow the GeoTools implementation to implement **both** org.geotools.styling and org.opengis.style interfaces. The acceptance test is to have existing code still compile; and to migrate the renderers to the new interfaces.

BEFORE:

```
public interface LineSymbolizer extends Symbolizer {
    Stroke getStroke();

    /** @deprecated Not thread safe - please use DuplicatingStyleVisitor */
    void setStroke(Stroke stroke);

    String getGeometryPropertyName();

    /** @deprecated Not thread safe - please use DuplicatingStyleVisitor */
    void setGeometryPropertyName(String geometryPropertyName);
}
```

AFTER:

```
public interface LineSymbolizer extends Symbolizer implements
org.opengis.style.LineSymbolizer {

    @Override Type narrow org.opengis.style.Stoke to Stroke
    Stroke getStroke();

    /** @deprecated Not thread safe - please use DuplicatingStyleVisitor */
    void setStroke(Stroke stroke);

    String getGeometryPropertyName();

    /** @deprecated Not thread safe - please use DuplicatingStyleVisitor */
    void setGeometryPropertyName(String geometryPropertyName);
}
```

Direct Collection Use

We are setting up new methods for direct manipulation of collection based access. Modifications made to the collection will issue change events.
BEFORE:

```
public interface FeatureTypeStyle {
    ...
    Rule[] getRules();
    void setRules(Rule[] rules);
    void addRule(Rule rule);
}
public interface Rule {
    ...
}
```

AFTER:

```
public interface FeatureTypeStyle extends org.opengis.FeatureTypeStyle {
    ...
    /** Same as: rules().toObjectArray( new Rule[0] ) */
    @Deprecated
    Rule[] getRules();

    /** Same as:
     * rules().clear()
     * rules().addAll( Arrays.asList( rules ) );
    @Deprecated
    void setRules(Rule[] rules);

    /** same as rules().add( Rule ) */
    @Deprecated
    void addRule(Rule rule);

    /** Offer direct access to the rules list; contents are assumed to be
org.geotools.styling.Rule */
    @override
    List<org.opengis.style.Rule> rules();
}
public interface Rule extends org.opengis.Rule {
    ...
}
```

NOTES:

- We cannot use the List in the Factory - we will need to use our own collection internally that can issue events as contents are added / removed / and moved up and down.

Listeners for Mutable Classes

AFTER:

```
public interface FeatureTypeStyle extends org.opengis.FeatureTypeStyle {
   addStyleListener( StyleListener listener );
   removeStyleListener( StyleListener listener );
}
public interface Rule extends org.opengis.Rule {
   addStyleListener( StyleListener listener );
   removeStyleListener( StyleListener listener );
}
/** Utility class for Style implementors - issues events when asked */
public StyleList extends ArrayList {
   addStyleListener( StyleListener listener );
   removeStyleListener( StyleListener listener );
}
```

Note each data structure is having its own set of listeners.

Notes:

- Be sure to include oldValue and newValue in the StyleEvent

New Methods for FeatureTypeStyle

AFTER:

FeatureTypeStyle.java

```
NEW METHODS
    /**
     * Returns a collection of Object identifying features object.
     *
     * <p>
     * ISO 19117 extends FeatureTypeStyle be providing this method.
     * This method enable the possibility to use a feature type style
     * on a given list of features only, which is not possible in OGC SE.
     * </p>
     *
     * @return Collection<String>
     */
    @UML(identifier="definedForInst", obligation=OPTIONAL, specification=ISO_19117)
    Id getFeatureInstanceID();

    /**
     * Returns a semanticType that identifies the more general "type" of geometry
     * that this style is meant to act upon.
     * In the current OGC SE specifications, this is an experimental element and
     * can take only one of the following values:
     * <p>
     * <ul>
     *   <li>{@code generic:point}</li>
     *   <li>{@code generic:line}</li>
     *   <li>{@code generic:polygon}</li>
     *   <li>{@code generic:text}</li>
     *   <li>{@code generic:raster}</li>
     *   <li>{@code generic:any}</li>
     * </ul>
     * <p>
     *
     */
    Set<SemanticType> getSemanticTypeIdentifiers();
```

New Methods for Symbolizer

AFTER:

Symbolizer.java

```
NEW METHOD
    /**
     * Returns a string of containing the measure unit name.
     * This parameter is herited from GML.
     * Renderers shall use the unit to correctly render symbols.
     *
     * recommended uom definitions are :
     * <p>
     * <ul>
     *     <li>{@code metre}</li>
     *     <li>{@code foot}</li>
     *     <li>{@code pixel}</li>
     * </ul>
     * <p>
     *
     * @return String name for the measure unit to use,
     * can be null. If the unit is null than we shall use a the pixel unit
     */
    @XmlElement("uom")
    Unit getUnitOfMeasure();
```

API Change for ColorMap

BEFORE :

ColorMap.java

```
public interface ColorMap {
    public static final int TYPE_RAMP = 1;
    public static final int TYPE_INTERVALS = 2;
    public static final int TYPE_VALUES = 3;

    public void addColorMapEntry(ColorMapEntry entry);

    public ColorMapEntry[] getColorMapEntries();

    public ColorMapEntry getColorMapEntry(int i);

    public int getType();

    public void setType(int type);

    void accept(StyleVisitor visitor);

    public void setExtendedColors(boolean extended);

    public boolean getExtendedColors();
}
```

AFTER:

ColorMap.java

```
public interface ColorMap {
    /** This is a live view of the ColorMap in function form*/
    Function getFunction();
}
```

NOTES:

- Is the Function a copy? Or is the Function another "view" on the same data structure? Can ColorMap implement Function and get it over with?

Those are only the most interesting/problematic changes, feel free to explore the interfaces to find other changes.

Status

This proposal was started by Johann Sorel but never completed; please see Cleanup Style Interface Deprecations for a plan to clean up after this incomplete work.

Voting on this proposal:

- Andrea Aime
- Ian Turton
- Justin Deoliveira
- Jody Garnett +1
- Martin Desruisseaux +1
- Simone Giannecchini

Community support:

- Johann Sorel +1

Tasks

This section is used to make sure your proposal is complete (did you remember documentation?) and has enough paid or volunteer time lined up to be a success

	no progress		done		impeded		lack mandate/funds/time		volunteer needed

July 15th deadline:

1. API change before GeoTools 2.5 goes out; deprecate symbolizer setter methods
2. Update wiki upgrade to to 2.5 pages; linking to instructions for DuplicatingStyleVisitor
3. Start 2.5.x branch
4. API changed based on BEFORE / AFTER
5. Update default implementation
6. Add factory methods
7. Deploy for other team members

After July 15th deadline:

1. Remove deprecated code from GeoTools project (yeah!)
2. Create org.opengis.style.StyleFactory based on experience above
3. Update or provided sample code in demo
4. Update the user guide examples (there are only two pages; copy and paste from demo)