

Dynamic SLD Graphic objects

Motivation:	SLD 1.0 and SE 1.1 do not address a range of common symbolization needs. We need ways to extend the SLD Mark and ExternalGraphic capabilities in a pluggable way
Contact:	Andrea Aime
Tracker:	http://jira.codehaus.org/browse/GEOT-1819
Tagline:	Programmatically extending SLD Graphic capabilities

This page represents the **current** plan; for discussion please check the tracker link above.

- [Description](#)
 - [Symbol naming](#)
 - [An SLD example](#)
 - [Proposal MarkFactory interfaces](#)
 - [API](#)
 - [Original Suggestion](#)
- [Set of factories being developed](#)
- [Status](#)
- [Tasks](#)
- [Documentation Changes](#)

Description

Map creation often calls for usage of symbols that are not supported directly by Mark and ExternalGraphic SLD elements.

Mark represent a resizable outline that can be filled and stroked at will. Unfortunately the SLD spec limits its usage to a set of well known names, and GeoTools has not clear path for extension. We need a way for people to specify the want to use externally defined vector symbols, like:

- the [cartographic symbols](#) supported by MapServer
- the true type fonts that do embed graphic icons, like Wingdings, or like many others that can be found on the internet like this [catalog](#).
- AutoCAD shapes and hatches, the Internet is full of them (some [examples here](#)).

External graphic represents complete images that can be reached using an URL. This approach fits symbols such as PNG or SVG images, but it's a good approach also for dynamic symbols that do include full styling (color, line width) like the MIL2525 ones (for a short description see [this mail](#)).

Symbol naming

Depending on the case, the symbol/external graphic name may be a pure static string, but often it's appealing to pass Feature attributes as parameter to drive the symbol creation. Standard SLD achieves this by using multiple Rule elements, each one filtering out certain features and assigning them a statically defined set of symbolizers. This unfortunately may lead to a need for hundreds of rules. In this proposal we'll use the power and simplicity of

CQL to circumvent this limitation.

The typical Mark/ExternalGraphic name will become:

```
protocol://path
```

Where the `protocol` most of the time identifies the factory that will generate the symbol, and the path is going to be a standard URL GET request. Single factories may decide to accept only static strings, whilst others may decide to allow for full blown CQL expressions, thus allowing the user to pass in Feature attribute values or to dynamically compute request parameters.

URL styles may assume various styles. Some could be:

- `name`: plain static name. This is proposed as a backwards compatible approach for well known names, and should be handled only by the factory handling those symbols (it is suggested that all of the others do use a `protocol://path` approach).
- `${cqlExpression}`: again a plain string, but this time the `${...}` part contains a CQL expression that can refer to Feature attributes. The CQL expression is evaluated to build the symbol name
- `http://host:port/real/path/to/symbol`: a standard URL that will be used by symbol factories that are able to retrieve and interpret as specific symbol type. This would allow, for example, to split SVG icon support to another module (the `renderer` module now has a heavy dependency on Batik for that).
- `http://host:port/real/path/to/symbol?param1=${cql1}¶m2=${cql2}`: same as above, but with the twist of being able to use CQL expressions as parameters to a remote symbol server (CQL expression will be evaluated in order to build the URL)
- `factoryId://symbolName`: an URL pointing directly to a specific factory with a plain symbol name. Something like `ttf://wingdigs#0x7B` for the flower symbol contained in the Wingdigs TTF font
- `factoryId://${expression}`: same as above, with the twist of being able to pass an attribute name or CQL expression as a param, so that the factory can perform different evaluations based on the current Feature attributes.

An SLD example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<PointSymbolizer version="1.1.0"
```

```
xsi:schemaLocation="http://www.opengis.net/se/1.1.0/Symbolizer.xsd"
```

```
xmlns="http://www.opengis.net/se"
```

```
xmlns:ogc="http://www.opengis.net/ogc"
```

```
xmlns:xlink="http://www.w3.org/1999/xlink"
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-  
instance"
```

```
uom="http://www.opengeospatial.org/se/units/  
pixel">
```

```
<Name>MyPointSymbolizer</Name>
```

```
<Description>
```

```
<Title>Example Pointsymbolizer</Title>
```

```
<Abstract>This is just a simple example of  
a point symbolizer.</Abstract>
```

```
</Description>
```

```
<Graphic>
```

```
<ExternalGraphic>
```

```
<OnlineResource xlink:type="simple"
```

```
xlink:href="http://www.vendor.com/geosym/226  
7.svg"/>
```

```
<Format>image/svg+xml</Format>
```

```
</ExternalGraphic>
```

```
<ExternalGraphic>
```

```
<OnlineResource xlink:type="simple"
```

```
xlink:href="http://www.iconrepo.com/${filena  
meAttribute}.png"/>
```

```
<Format>image/png</Format>
```

```
</ExternalGraphic>
```

```
<ExternalGraphic>
```

```
<OnlineResource xlink:type="simple"
```

```
xlink:href="mil2525b://{symbol}"/>
```

```
<Format>image/png</Format>
```

```
</ExternalGraphic>
```

<Mark>

<WellKnownName>ttf://wingdigs#0x7B</WellKnownName>

<Fill>

<CssParameter
name="fill">#00DEFF</CssParameter>

</Fill>

</Mark>

```
<Size>15.0</Size>
</Graphic>
</PointSymbolizer>
```

In the above example above:

- the first is a normal external graphic reference
- the second one uses the Feature's `filenameAttribute` to build an URL to a file located on a remote server
- the third one passes the url over to a factory that builds the symbol whose specification is contained in the `symbol` attribute of the current feature.
- the

Proposal MarkFactory interfaces

API

Here are the interfaces for Mark and ExternalSymbolizer factories:

```
/**
 * Symbol handler for a Mark.
 */
public interface MarkFactory {
    /**
     * Turns the specified URL into an
     Shape, eventually using the Feature
     attributes
     * to evaluate CQL expressions embedded
     in the url, or returns null
     * if the factory cannot evaluate this
     symbolUrl.
     */
    public Shape getShape(Expression
symbolUrl, Feature feature) throws
Exception;
}
```

```
/**
 * Symbol handler for an external
symbolizers.
 */
public interface ExternalGraphicFactory {
    /**
     * Turns the specified URL into an Icon,
eventually using the Feature attributes
     * to evaluate CQL expressions embedded
in the url.<br>
     * The <code>size</code> parameter
defines the size of the image (so that
vector
     * based symbols can be drawn at the
specified size directly), or may be -1 if
the
     * size was not specified (in that case
the "natural" size of the image will be
used).<br>
     * <code>null</code> will be returned if
this factory cannot handle the provided url.
     */
    public Icon getIcon(Feature feature,
```

```
Expression url, String format, int size)
throws Exception;
}
```

The MarkFactory is meant to generate `java.awt.Shape` objects that can be sized, filled and stroked at will (even TTF symbols can be turned into a shape using).

The ExternalGraphicFactory on the other side has to generate finished images that will be eventually just rotated. This is why a size parameter is provided, to allow generating the best sized Icon should the original resource be a vector type one (e.g. SVG). If the returned Icon is really a `java.awt.Image`, then the `ImageIcon` class can be used. If the icon returned is based on an SVG, some vector drawing could be used in `paint(...)` to achieve top quality.

Each URL and Mark name is an OGC Expression, meaning that it may be dependent on the Feature attributes. Factories that do need to turn the expression into a name (Mark) or an URL (ExternalGraphic) just need to evaluate the expression providing the feature as a parameter: `expression.evaluate(Feature, String.class)` or `expression.evaluate(Feature, URL.class)`. Using Expression also provides the `SLDStyleFactory` enough information to decide whether the expression is static (that is, a Literal) or Feature dependent, and optimize the first case by means of symbol caching. At the same time, it leaves the door open for the factory to use complex attributes that might not be turned efficiently into a string (an alternative API design could have used `getIcon(String url, String format, int size)` but that would have been problematic for complex attributes).

Each factory would be registered in the SPI just like `DataStore` and coverage readers are, that is, special text file contained in META-INF would contain the name of all the factories for the current jar (each jar having its own registry file).

Original Suggestion

Original suggestion to handle MIL2525B was to use `WellKnownName` and allow the usage of an Expression in it:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<PointSymbolizer version="1.1.0"
```

```
xsi:schemaLocation="http://www.opengis.net/se/1.1.0/Symbolizer.xsd"
```

```
xmlns="http://www.opengis.net/se"
```

```
xmlns:ogc="http://www.opengis.net/ogc"
```

```
xmlns:xlink="http://www.w3.org/1999/xlink"
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
```

instance"

uom="http://www.opengeospatial.org/se/units/
metre">

<Name>MyPointSymbolizer</Name>

<Description>

<Title>Example Pointsymbolizer</Title>

<Abstract>This is just a simple example of
a point symbolizer.</Abstract>

</Description>

<Graphic>

<Mark>

<WellKnownName><PropertyName>symbol</code></
WellKnownName>

<Fill>

<SvgParameter

name="fill">#ff0000</SvgParameter>

</Fill>

</Mark>

<Size>8.0</Size>

```
</Graphic>
</PointSymbolizer>
```

In the above example the WellKnownName is defined by the attribute **symbol** - and we provide a WellKnownNameFactory to generate additional glyphs beyond those provided by the specification (ie "start", "circle", etc...). The Java api would not have to change, but:

- the SLD parsers would have to be modified
- the above document would not be compliant with any published SLD schema
- it is inconsistent with the Mark nature, which is a shape, not a colored icon (MIL2525B embeds color in the symbol name as well)

This approach is suitable for emergency response and mil2525B symbols; but is not as clean or as extensible as the CQL approach proposed above.

Set of factories being developed

Here is a rough idea of what factories could be developed:

- ExternalGraphicFactory
 - AbstractExternalGraphicFactory: an abstract handler designed to access graphics that are really external, either on a remote server or on the file system.
 - ImageIOExternalGraphicFactory: a subclass that can load image files as handled by ImageIO
 - SVGExternalGraphicFactory: a subclass that can load SVG files
 - Mil2525Factory: a subclass handling MIL2525 symbols
- MarkFactory
 - WellKnownNameMarkFactory: a factory handling the current well known name symbols (square, circle, ...)
 - TTFMarkFactory: one that handles fonts based symbols
 - MapServerMarkFactory: for MapServer symbol names
 - CADMarkFactory: for Autocad hatch pattern files

The SLDStyleFactory will be augmented with a method used to expand the URL/String retrieved from the symbolizer into a full blown OGC expression. In the specific case of Mark, the GeoTools code already allows the usage of an Expression. In this case, the expansion will take place only if the Mark expression is a literal string, whilst the expression will be left as is to handle the case where the Expression has been build by code directly (and thus does not need expansion of the embedded `{cqlExpression}` items).

```

/**
 * Evaluates as CQL expressions embedded
 in the url as
 * <code>${cqlExpression}</code> using
 the provided Feature to feed them
 */
Expression expand(String url, Feature
feature);
}

```

Given an URL like `mil2525b://${mil2525Att}` and a Feature whose `mil2525Att` value is `"SFGPUCAAA-*****"`, calling `ExpressionExpander.expand(...)` would result in a OGC expression like `strConcat("mil2525b://",property("mil2525Att"))` that the style factory will evaluate in order to get to `"mil2525b://SFGPUCAAA-*****"`.

Status

This proposal has been accepted; the work is done except for the documentation.

Voting has completed:

- [Andrea Aime](#) +1
- [Ian Turton](#) +1
- [Justin Deoliveira](#) +1
- [Jody Garnett](#) +1 (from email)
- [Martin Desruisseaux](#)
- [Simone Giannecchini](#) (pending - see email)

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
--	-------------	---	------	---	---------	---	--------------------------	---	------------------

1. ✓ Implement the proposed factory interfaces and develop a helper class to look for the right factory given a URL (and format) (aaime)
2. ✓ Implement `ImageIOExternalGraphicFactory`, `SVGExternalGraphicFactory` and `WellKnownNameMarkFactory` (aaime)
3. ✓ Change `SLDStyleFactory` to use the new factory interfaces and make sure everything still works as before

(aaime)

4. Add TTFMarkFactory (aaime)
5. Add a demo Mil2525ExternalGraphicFactory as a demo of the new ExternalGraphicFactory (in the demo section, since this would not be complete?) (theunsgis)
6. Update the user docs 
7. Implement other factories as time permits (out of the scope of this proposal)
8. write change proposal for OGC SLD Specification to pass changes to standard

Documentation Changes

list the pages effected by this proposal

- [Home](#): add a sample page showing how to build your won MarkFactory or ExternalSymbolizerFactory, and a sample user doc showing how to drive the factories developed so far