AIDA Styles

This is a collection of thoughts on AIDA Styles and on their use in the new plotter.

Style Store

This is an object that can save and retrieve styles from a store (xml file on disk, database table). The basic xml style definition is the AIDA one. Next to the style itself we also identified the need to define Style Rules that should also be saved/retrieved to/from the store:

```
<styleStore>
   <styles>
       <style>
          .......
          <rules>
             <rule>
                .......
             </rule>
          </rules>
       </style>
   </styles>
</styleStore>
```

Multiple stores might be loaded at once: System Store, Personal Store, Group Store, Experiment Store. Their information is grouped and managed by a Style Registry

Style Rules

It should be possible to define rules on how to attach a style to a particular plot object. Multiple rules might be applied to a given style. It should be possible to apply rules to:

- object type: hep.aida.IHistogram1D, org.myproject.MyData1D
- object path: with path expressions like "MC/**"
- plot order: a specific order position, 0, 7 or a cyclical recurrence (3) -> every third plot
- action: like printing (plots for printing might require less details)
- category: "experiment=GLAST, quality=preliminary"

Style Registry

A style registry combines the styles and rules from different style stores and provides the right set of styles (with the appropriate Style Order) for a given plot object.

Style Order

We have to define the order in which the styles are applied to an object:

- passed style (when plotting)
- explicit in annotation (like labels)
- implicit:
  - action
  - category
  - plot order
  - path
  - type

Style Editor

GUI front end for viewing and editing individual styles or combinations of styles.
We already have a first version of the style editor. We need to add the possibility to view the information contained in a style registry, i.e. the chain of styles contributing to a given object and the set of rules that have contributed to it.

Comments to AIDA Styles

- does isVisible() belong to IBaseStyle?
- can we plot an object by passing only an IDataStyle (rather than an IPlotterStyle)?

Ideas for Implementation

Let's leave Style Store out of this discussion just for the moment.

So Style Registry is created and is getting populated by (or just acts as a manager/façade for) the Style Stores, but it has access to Style/Rules information.

Just before object is plotted/printed and we know the object's path in a Tree, object's type, order of the object in particular IPlotterRegion, etc. All this information is used to query the Style Registry and create a cumulative IPlotterStyle for the implicit parameters.

Information that we need:

- Object type: from the object itself
- Object path: from AIDA tree, from initial user plotting request, or from the object
- Plot order: from the current IPlotterRegion - is it overlay or not, and if it is - how many objects are already plotted there
- Attribute: arbitrary attributes in a form of "key=value" pair can be added
- Action: program that is performing action should know what it is doing - plotting, printing, etc.
- Category: this is an external parameter and has to come from a separate service. We must be able to keep track of multiple categories and their current values, e.g. "experiment=BaBar" _and_ "quality=draft"
    - Can be part of Style Registry functionality, or independent service
    - Example: selected by user from the list of available Categories - PAW style plots, GLAST style plots
    - Action looks a lot like Category: "printing" -> "printing=true"

If any styles are passed explicitly to the plotter, this implicit cumulative style will be set as parent (or merged in as a lower priority Style).

I've set three interfaces that might do the job (below), have a look.

IStyleRegistry interface:

IPlotterStore in StyleRegistry is identified by a name (storeName). Store names have to be unique.

Also StyleRegistry can manage the Categories: list of all available keys and set of current values.

```
package hep.aida.ref.plotter.style.registry;

import hep.aida.IPlotterStyle;

public interface IStyleRegistry {

    // To work with Style Stores

    String[] getAvailableStoreNames();

    IStyleStore getStore(String storeName);


    // To work with categories, this can be a separate service
    // Available category keys are filled from Rules of all available Stores

    String[] getAvailableCategoryKeys();

    String[] getAvailableCategoryValues(String categoryKey);

    String getCategoryCurrentValue(String categoryKey);

    void setCategoryCurrentValue(String categoryKey, String categoryValue);

    // Following methods are used to obtain cumulative IPlotterStyle
    // for particular plotter, region, object, action, and (possibly) categories

    IPlotterStyle getStyleForState(IPlotterState state);
}
```

IStyleStore interface:

```
package hep.aida.ref.plotter.style.registry;

import hep.aida.IPlotterStyle;

/**
 * This interface can be implemented as "In-Memory" copy of persistent
 * facility, or as keeping live connections and committing any change
 * immediately.
 */

public interface IStyleStore {

    // Key for AIDA type of object that the Style is going to be used with
    public static String STYLE_PREVIEW_TYPE = "STYLE_PREVIEW_TYPE";

    // Key for Style name
    public static String STYLE_STORE_NAME = "STYLE_STORE_NAME";

    String getStoreName();

    String getStoreType();

    boolean isReadOnly();


    // Manage Styles

    boolean hasStyle(String styleName);

    void addStyle(String styleName, IPlotterStyle style);

    void addStyle(String styleName, IPlotterStyle style, IStyleRule rule);

    IPlotterStyle getStyle(String styleName);

    /**
     * Remove Style and Rule associated with it from the Store
     */
    IPlotterStyle removeStyle(String styleName);

    String[] getAllStyleNames();


    // Create new Rule for this Store - Store acts as a Rule Factory

    IStyleRule createRule();


    // Manage Rules - only one rule per style is allowed

    IStyleRule getRuleForStyle(String styleName);

    void setRuleForStyle(String styleName, IStyleRule rule);

    void removeRuleForStyle(String styleName);

    /**
     * Write all information from Store to the undelying persistent
     * facility: XML file, database, etc.
     */
    void commit();

    /**
     * Close all connections and free all resources.
     * Store is not usable after this method is executed.
     */
    void close();

}
```

IStyleRule interface:

Style Rule contains expression that is evaluated at run-time

```
package hep.aida.ref.plotter.style.registry;

public interface IStyleRule {

    public static String PATH = "Path";
    public static String OBJECT = "Object";
    public static String OBJECTTYPE = "ObjectType";
    public static String NULL = "Null";
    public static String ATTRIBUTE = "attribute(\"\")";
    public static String OVERLAYINDEX = "OverlayIndex";
    public static String OVERLAYTOTAL = "OverlayTotal";
    public static String REGIONINDEX = "RegionIndex";
    public static String REGIONTOTAL = "RegionTotal";

    String getDescription();

    // Evaluates the Rule
    boolean ruleApplies(IPlotterState state);
}
```

How to Evaluate Style Rules

- CLASS
  - Plotted object is exactly instance of specified class
  - Plotted object is derived from specified class
- PATH
  - Path contains specified sub-path
    - Case sensitive
    - NOT Case sensitive
  - Regular expression
- ORDER
  - Absolute number of overlaid plots, like 3-rd
  - Position in the recurring sequence, like 4-th out of 7
- ATTRIBUTE: match "key=value" pair
  - Case sensitive
  - NOT Case sensitive
- ACTION: match action name
  - Case sensitive
  - NOT Case sensitive
- CATEGORY: match "key=value" pair
  - Case sensitive
  - NOT Case sensitive

IPlotterState interface:

```
package hep.aida.ref.plotter.style.registry;

/**
 * This object encapsulates information about relevant
 * IPlotterRegion, object, and actions.
 * It is used for obtaining implicit IPlotterStyle
 */

import java.util.Map;

public interface IPlotterState {

    static String ATTRIBUTE_KEY_PREFIX = "IPlotterState";

    Object getObject();
    String getObjectPath();

    int getOverlayIndex();
    int getOverlayTotal();

    int getRegionIndex();
    int getRegionTotal();

    String getAttribute(String key);

    Map getAttributes();

    void clear();
}
```