Custom Formatter

Overview

top
The Yellowfin application has the ability to format different data types based on a user-defined java class. This allows for parsing the returned database value and transforming it into another format.

CustomFormatter Class

top
Developers can create their own formatter class that extends the class:

```
com.hof.mi.interfaces.CustomFormatter
```

The following methods must be implemented in your formatter class:

- `public abstract String getName();`
- `public abstract booleanacceptsNativeType(int type);`
- `public abstract String render(Object value, intrenderType) throws Exception;`

public abstract String getName();

This method returns a display name for your formatter. This is the name that is displayed to the user when choosing a formatter. For example:

```
public String getName() {
return "My Formatter";
}
```

public abstract booleanacceptsNativeType(int type);

This method determines which native data types can be handled by this formatter. The available native types are defined in the super class of `Custom Formatter`:

```
public static final int TYPE_NUMERIC          = 1;
public static final int TYPE_TEXT             = 2;
public static final int TYPE_DATE             = 3;
public static final int TYPE_TIME             = 4;
public static final int TYPE_TIMESTAMP        = 5;
public static final int TYPE_BOOLEAN          = 6;
public static final int TYPE_BINARY           = 7;
public static final int TYPE_GISPOINT         = 8;
public static final int TYPE_GISPOLYGON       = 9;
public static final int TYPE_GISLINESTRING    = 10;
public static final int TYPE_GISMULTILINE     = 11;
public static final int TYPE_GISMULTIPOLY     = 12;
public static final int TYPE_GISMULTIPOINT    = 13;
public static final int TYPE_GISMULTIGEOMETRY = 14;
public static final int TYPE_GISGEOMETRY      = 15;
```

Your class should return true for types that it supports, and false otherwise.

Example 1

```
publicbooleanacceptsNativeType(int type) {
   // we accept text and numeric data
if (type == TYPE_TEXT || type == TYPE_NUMERIC) {
return true;
   } else {
      // don't allow any other types
return false;
   }
}
```

Example 2

```
publicbooleanacceptsNativeType(int type) {
   // we can handle any type
return true;
}
```

public abstract String render(Object value, intrenderType) throws Exception;

This method renders a single value. The `renderType` argument determines which output type we are rendering for:

```
public static final int RENDER_HTML       = 1;
public static final int RENDER_TEXT       = 2;
public static final int RENDER_CSV        = 3;
public static final int RENDER_LINK       = 4;
```

Example

```
public string render(Object value, intrenderType) throws Exception {
if (value == null) return null;

if (renderType == RENDER_LINK) {
      // Render the value for a drill-through link.
      // In this case we almost always want to return a generic
      // representation of the value.
returnvalue.toString();
   }

   // Return the formatted value
return "Value: " + value.toString();

}
```

The `renderType` parameter gives you a hint as to how to display the value. When outputting to document formats such as PDF and XLS the `RENDER_TEXT` type is used.

Embedded HTML

In some cases you may want to return an HTML string from your formatter. This may be useful to customise the displayed text (eg. with embedded bold/italic styling), or present a link. This can be achieved by overriding the method:

```
publicbooleanreturnsHtml() {
return false;
}
```

By default, this method returns false, indicating that any HTML entities in the formatted values should be escaped when outputting to HTML. By overriding this to return true, Yellowfin will not do any HTML escaping. An example of a formatter that includes HTML output:

```
publicbooleanreturnsHtml() {
return true;
}
public String render(Object value, intrenderType) throws Exception {
if (value == null) return null;
if (renderType == RENDER_LINK) {
returnvalue.toString();
   } else if (renderType == RENDER_HTML) {
return "<b>" + value.toString() + "</b>";
   } else {
      // rendering to non-html output
returnvalue.toString();
   }
}
```

Compiling a Custom Formatter

To compile your custom formatter class, you will need to include two libraries in your classpath:

```
i4-core.jar
i4-mi.jar
```

These are located in the `WEB-INF\lib\` directory within the Yellowfin webapp, for example: `Yellowfin\appserver\webapps\ROOT\WEB-INF\lib\`.

top