Implementing a Custom Formatter

A custom formatter is an implementation of the class **com.hof.mi.interfaces.CustomFormatter**. The following functions must be implemented in this class:

- public abstract String getName();
- public abstract boolean acceptsNativeType(int type);
- public abstract String render(Object value, int renderType) throws Exception;

These methods can be overwritten, if required:

- public boolean returnsHtml() { return false; }
- public boolean compareConditionalAgainstFormattedValue() { return false; }

Function Definitions

public abstract String getName();

This method returns a custom formatter name as a String. This is the name that is displayed to the user when choosing a formatter. For example:

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

---

public abstract boolean acceptsNativeType(int type);

This method returns true if the formatter will support fields of the given data type. Otherwise it returns false. A list of supported types is provided in the appendix.

Example 1:

```
public boolean acceptsNativeType(int type) {

//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:

```
public boolean acceptsNativeType(int type) {

// we can handle any type
return true;
}
```

---

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

This method renders a single value. The renderType argument determines which output type you are rendering for. Yellowfin executes this function for every value of the formatted column. The renderType parameter gives you a hint as to how to display the value. When outputting to document formats such as PDF/XLS, the RENDER_TEXT type is used. For a list of possible renderType values, refer to the appendix.

```
public String render(Object value, int renderType) throws Exception {
      if (value == null) return null
      if (renderType == RENDER_LINK)
       // Render the value for a drill-through link.
       // In this case you almost always want to return a generic
       // representation of the value.

      return value.toString();
          }

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

---

public boolean returnsHtml();

This will return true if the values returned by the formatter include HTML, but false otherwise. By default this function returns false.

---

public boolean compareConditionalAgainstFormattedValue();

This function returns true if the fields using this formatter should compare conditional formats against the formatted values, otherwise it returns false. By default this function returns false.

Previous topic: Formatter prerequisites
Next topic: Embedded HTML inclusion