Object Formatter Component Deployment Guide

Requirements

Environment:

- Solaris 7
- RedHat Linux 7.1
- Windows 2000

TopCoder Software Components:

None.

Third Party Components:

None.

Detailed Configuration

None.

Usage Notes

- **How to format primitive types**

PrimitiveFormatterFactory provides 3 ways to format a primitive type:

1. The static *getPlainFormatter* method return a formatter which converts each primitive type into its plain string representation using its appropriate String.valueOf method.

2. The static *getPrettyFormatter* method return a formatter which converts each primitive type into a “pretty” string representation. Their formats are as follow:
   int: ##,###
   short: ###
   long: ##,###
   byte: #
   float: ##,###.00
   double: ##,###.00

3. Two static *getFormatter* methods allow users to define their own ways to format the primitive types. Users can provide the format string, which follows the pattern strings for java.text.DecimalFormat. Users can also provide their java.text.DecimalFormat, then its number-formatting information will be used to interpret the format string.

Below is an example to format a primitive:
```
PrimitiveFormatter formatter =
PrimitiveFormatterFactory.getFormatter("0,00.00");
double d = -1234567.890;
formatter.format(d); // the return value is "-1,23,45,67.89"
```

- **How to format Date type**

FormatMethodFactory provides ways to format a Date similar to *PrimitiveFormatterFactory.*

1. If using the *getDefaultDateFormatMethod* method with a *null* parameter, the default format “MM/dd/yyyy” for the Date will be used.

2. Users can define their own ways to format a Date by providing the format string, which follows the pattern strings for java.text.SimpleDateFormat Users can also provide their java.text.SimpleDateFormat, then its date-formatting information will be used to interpret the format string.

Below is an example to format a date:

```
DateFormatMethod formatter =
FormatMethodFactory.getDefaultDateFormatMethod("yyyy.MMMMM.dd hh:mm
aaa");
Calendar c = Calendar.getInstance();
c.set(2002, Calendar.DECEMBER, 25, 0, 0, 0); // December 25, 2002, 00:00:00
Date d = c.getTime();
formatter.format(d); // the return value is "2002.December.25 12:00 AM"
```

- **How to format Object types**

ObjectFormatterFactory provides 3 static methods to create a *ObjectFormatter.*

1. The *getEmptyFormatter* method returns an empty formatter, which will not be able to format any object without being supplied with the appropriate ObjectFormatMethod.

2. The *getPlainFormatter* method returns a formatter that associates Objects with the format method returned by *FormatMethodFactory.getDefaultObjectFormatMethod,* which simply format an Object with their *toString* method.

3. The *getPrettyFormatter* method returns a formatter that creates an association for each primitive type's wrapper class equivalent of formatter for the primitive types from *PrimitiveFormatterFactory.getPrettyFormatter()* and also creates an association for Dates, using the format method returned by *FormatMethodFactory.getDefaultDateFormatMethod(java.lang.String)*

A note for the usage of ObjectFormatter: If an Object obj could be formatted by more than one format method, then the following algorithm will be used to determine the format method to use, if any. The format method whose associated type is closest to obj's type will be used. This closest type is determined by first determining all format methods (a) whose associated type is a super-type of obj (that is, it is an interface that obj implements, or it is a superclass of obj), and (b) that actually could format obj (because either the type matches exactly, or because the type is a super-type and is allowed to format

sub-types). This set of types $type_1$, $type_2$, ... $type_n$ is then scanned to determine if there is any $type_i$ that is a sub-type of all the others. If so, then the format method associated with $type_i$ is used to format obj; if not, an exception is thrown.

Users can also decide whether subtypes of the supplied type should be formatted with this format method by setting the boolean value when they use *setFormatMethodForClass* method.

Below is an example to format an Object:

```
ObjectFormatter of = ObjectFormatterFactory.getPrettyFormatter();

ObjectFormatMethod ofmCollection = new ObjectFormatMethod() {
        public String format(Object obj)
        {
            if(!(obj instanceof Collection))
                throw new IllegalArgumentException("expected Collection");
            return "Collection! " + obj;
        }
};

 ObjectFormatMethod ofmList = new ObjectFormatMethod() {
        public String format(Object obj)
        {
            if(!(obj instanceof List))
                throw new IllegalArgumentException("expected List");
            return "List! " + obj;
        }
};

 ObjectFormatMethod ofmRandomAccess = new ObjectFormatMethod() {
        public String format(Object obj)
        {
            if(!(obj instanceof RandomAccess))
                throw new IllegalArgumentException("expected RandomAccess");
            return "RandomAccess! " + obj;
        }
};

 of.setFormatMethodForClass(Collection.class,   ofmCollection,   true);
 of.setFormatMethodForClass(List.class,         ofmList,         true);
 of.setFormatMethodForClass(RandomAccess.class, ofmRandomAccess, true);

 LinkedList ll = new LinkedList();
 ll.add(new Integer(1));
 ll.add(new Integer(2));
 ll.add(new Integer(3));

 // LinkedList should be formatted with the format method of List class,
 // which is the closest supertype of LinkedList
```

```
of.format(ll));

ArrayList al = new ArrayList(ll);

// ArrayList cannot be formatted, because List and RandomAccess are both
// super-types  of ArrayList, but neither is a sub-type of the other. It will throws
// an IllegalArgumentException
String bob = of.format(al);

// Set the fFormatSubtypes of the RandomAccess class to be false so
// that the subtypes of RandomAccess can't be formatted with its format method
of.setFormatMethodForClass(RandomAccess.class, ofmRandomAccess, false);

// Now ArrayList can be formatted with the format method of List class.
of.format(al));
```

Build Notes

No changes are needed in the build.xml file distributed with the component design.