Time Tracker Report 1.0 Component Specification

1. Design

The Time Tracker Report custom component is part of the Time Tracker application. It provides the ability to create time and expense reports for employees, projects, and clients. This component handles the necessary database queries that make up the various reports. This component is required to generate the actual reports as tables embedded in a web page.

This component will be largely flexible in terms of the following areas:

1. The Columns that are displayed on the report. (Any subset of the Default set of columns should be configurable for display).
2. Runtime decision of Filters for the report. (A report will have mandatory filters as well as optional filters). At runtime, it should be possible to switch off or on a particular optional filter. (Example: For the same JSP page, if one user logs in, he should be able to see the report containing his details and if other user accesses the same JSP page, the report should be able to show his details. In addition to this, the user can pass on certain further filtering information (say a particular Project or a particular Client) to the same JSP page and he should be able to get the Filtered details in display back.
3. Configurable CSS styles for the report.
4. Has support provided in the framework to allow the generation of the Report in different formats. (Though only the implementation for generating the report as a HTML Table is provided)

Analyzing these configurations and the flexibility expected, we arrive at 2 sets of configurations that are possible, one which is configurable at runtime(Filters for the report) and one which does not require configuration at runtime(CSS Styles and Columns to be displayed).

It is important to note that, the Filters for the reports need to be configured at runtime or else two users cannot get data specific to them by accessing the same JSP page.

To keep the development using the component simple and also provide the required flexibility, we define a single JSP tag which has attributes specifying the *Runtime Configuration* of the report to be displayed and contains information as to:

1. The *Category* of the Report.
2. The *Type* of the Report.
3. The *Filters* for the Report.

The attributes of this JSP tag have been defined and explained in section 3.2. (TLD configurations for tag).

The configuration of what Columns, Column Headers, CSS Styles needs to be used for a JSP Page displaying the report is not a part of Runtime Configuration, but can be Configured by just changing the Namespace attribute in the JSP tag.

The design has categorized the handling of specific section of the Report rendering based on the category of Report. This is because, based on the Category the summarization information changes for a report.

The following table provides a quick reference to the locations in the document, in the associated configuration files and the UML diagrams, where the design meets the requirements specified:

No	Requirement	Location
1	Requirement 1.2.1.1	The enumeration classes for ReportCategory, ReportType, defines the various combinations of the reports that can be generated by the component. **The SQL Queries for each of the nine Reports, is present in the Time_Tracker_Report.xml file. Please refer this file for the queries.**
2	Requirement 1.2.1.2 (Filters)	The Filter interface and its implementations and the FilterCategory and the FilterType enumerations work towards satisfying the requirement.
3	Requirement 1.2.1.3 (Configurable Columns)	This is present as a part of the configuration for the report in the Time_Tracker_Report.xml file. The classes that work towards satisfying this requirement are Column, ColumnDecorator interface and its implementations.
4	Requirement 1.2.1.4 and 1.2.1.5.	The enumeration classes for ReportCategory, ReportType has been defined for the same and reports generation for the different combination of these is provided.
5	Requirement 1.2.2	The report implementation classes TimeReport, ExpenseReport and TimeExpenseReport.
6	Requirement 1.2.4 (Pluggable persistence)	The interface and classes defined in the package “com.topcoder.timetracker.report.dbhandler” allow for pluggability of the Database.
7	Requirement 1.2.5 (CSS Styles)	The enumeration class StyleConstant and the style configuration in the Time_Tacker_Report.xml file for the report work towards satisfying this requirement.
8	Requirement 1.2.6 (Custom JSP Tag)	The taglib definition for the custom JSP tag for the Time Tracker Report application is defined in the “time_tracker_report.tld” file. The class handling this custom JSP Tag is ReportDisplayTag.

In addition to the basic set of requirements mentioned, the component also provides additional features which are detailed as follows:

No	Requirement	Location
1	Framework for allowing the report to be generated in different formats.	This is not a part of the core requirement, but extremely useful, if the report data is required to be rendered in some other format.
2	Column Decoration	This is again not a part of the core requirement, but a very useful functionality to actually decorate the column data before displaying it. This component provides a basic decoration feature wherein the column data can be prefixed by a

		constant prefix and/or suffixed with a constant suffix.

Note: The design assume that a ConnectionProducer for the Informix database has been registered with the DBConnetionFactory, by the "Time Tracker" application. The required properties for creating the ConnectionProducer for the Informix database should be specified in the configuration of the DBConnectionFactory component under the namespace "com.topcoder.timetracker.report.Informix".

Also the configuration file Time_Tracker_Report.xml file has to be configured to be loaded by the ConfigManager component.

1.1 Design Patterns

1. Factory pattern is used for ReportFactory and DBHandlerFactory for handling the instantiation of the Report and DBHandler implementations.
2. Strategy pattern is used for Filter, ColumnDecorator, DBHandler and Report.
3. AbstractReport, TimeReport, ExpenseReport and TimeExpenseReport together form a Decorator pattern.
4. Data Transfer Object (DTO) pattern is used for ReportConfiguration class. (Basically a sort of bean with getter and setter methods).

1.2 Industry Standards

- JSP
- JSTL
- JDBC
- CSS

1.3 Required Algorithms

Note: (Important) It would be best to go through the details in the configuration file "Time_Tracker_Report.xml" file, before proceeding to read the algorithm section. This will be very much useful to help in understanding the algorithms better.

1.3.1 Get the Report instance, creation of Filters and executing the report.

(Inside the ReportDisplayTag:doStartTag() method)

1. Check if any of the following instance members: type, category or namespace is null or return a zero length string on trimming. If yes, then throw a JSP exception wrapping an instance of ReportConfigurationException containing the appropriate message.(Missing required attribute values), else continue with next step. 2. Read into local string variables typeValue and categoryValue, the values returned by super.pageContext.findAttribute(type) and super.pageContext.findAttribute(category) respectively. Check if any of the typeValue or categoryValue is null or empty string. If yes then throw

an exception as mentioned in step1, else continue with step3.
(The specification of the attributes names by the type and the category variables, allow us not to tie these attributes to be expecting to have a specific name, but the name can be changed by changing the value of the attributes of the tag in the JSP page rather than requiring to change the Client sending the parameter value.)

3. Call the method createFilters() to create an List containing Filter objects based on the data available in the HTTPRequest object and the HTTPSession object.
If the method throws an exception, then wrap the exception to a JSPException and throw the exception, else continue.

4. Get a Report object instance corresponding to the category by invoking ReportFactory.getInstance().getReport(FORMAT,categoryValue).
If the method throws an exception, then wrap the exception to a JSPException and throw the exception, else continue.

5. Invoke the execute(namespace,typeValue,filters) method on the Report object and write out the String returned by the method to the pageContext.
e.g: pageContext.getOut().println(report.execute(namespace,typeValue,filters)).
If the method throws an exception, then wrap the exception to a JSPException and throw the exception, else continue.

6. Return SKIP_BODY.

1.3.2 Loading of the Report implementations by the ReportFactory: (loadConfiguration() method)

```
String namespace = "com.topcoder.timetracker.report.Reports";

ConfigManager manager = ConfigManager.getInstance();

String reports[] = manager.getStringArray(namespace,"ReportClasses");

if(reports!=null){

// registering of the Report Implementation from the configuration.

  for(int i=0;i<reports.length;i++){
        String className = reports[i];
        if(className!=null){
          try{    // loading through reflection.

              Object object =  Class.forName(className).newInstance();
               if(!(object instanceof Report){
                  throw new ReportConfigurationException("Class is not an instance of
```

```
                        Report: "+classname);
                }
                Report report = (Report)object;
                String key = report.getFormat() + "_" + report.getCategory().getCategory();
                reports.put(key, object);
            }
            catch(Exception e){
                throw new ReportConfigurationException(e.getMessage(),e);
            }
        }
}
```

1.3.3 Loading of the DBHandler implementations by the DBHandlerFactory: (loadHandlersFromConfiguration() method)

```
String namespace = "com.topcoder.timetracker.report.DBHandlers";

ConfigManager manager = ConfigManager.getInstance();

String handlers[] = manager.getStringArray(namespace,"DBHandlers");

if(handlers=null){

// registering of the DBHandler Implementation from the configuration.

    for(int i=0;i<handlers.length;i++){
        String className = manager.getStringArray(namespace,handlers[i]);
        if(className!=null){
            try{  // loading through reflection.

                Object object =  Class.forName(className).newInstance();
                if(!(object instanceof DBHandler){
                    throw new ReportConfigurationException("Class is not an instance of
                            DBHandler: "+classname);
                }

                dbhandlers.put(handlers[i], object);
            }
            catch(Exception e){
                throw new ReportConfigurationException(e.getMessage(),e);
            }
        }
    }
```

1.3.4 Creation of Report Configuration:

(Inside AbstractReport.createConfiguration() method)

1. Get the List containing the validFilters for the current type of Report from the "validFiltersForType" HashMap instance member.

2. Check if any of the filters in the List passed, is not a part of the validFilters List.
   If found any, then throw a ReportConfigurationException with appropriate message, else continue.

3. Check if the List passed has the defaultFilter for the type of report.
   If not then throw a ReportConfigurationException with appropriate message, else continue.

4. Get the instance of ConfigManager, using ConfigManager.getInstance() and have it in temporary variable configManager.

5. Create a temporary List for columns.

6. For each "column" constant returned by Column.getColumns() method do
   a. String propertyValue = configManager.getProperty(namespace,"COLUMN_"+column.getName());
   b. if propertyValue is null, continue,
      else create a ColumnDecoration and add the ColumnDecoration to the columns list.

      e.g: columns.add(new BasicColumnDecorator(column.getName(),propertyValue));

7. Check if columns List is empty after the loop.
   If yes, then the default set of columns for report is loaded into the columns List by creating BasicColumnDecorator for each of them and the DisplayLabel same as the columnname.

8. Check the columns List with the List of the default columns for the report type obtained from the "defaultColumnSetForType" HashMap. If some column present in the columns list is not present in the defaultList, then throw a ReportConfigurationExcpetion with appropriate message, else continue.
   This is for checking if some invalid column for the report was specified as a part of the configuration.

9. For each of the BasicColumnDecorator in the columns List,
   read the value of the property PREFIX_ and SUFFIX_ from the namespace and set prefix and suffix property on the BasicColumnDecorator.

10. Create a "styles" Map. A Local variable.

11. For each of the styles constants obtained from StyleConstant.getStyleConstants():
- try fetching the property value as in step 6 and if not null, then
insert it into styles HashMap, else insert an empty string in
the HashMap for the style.

12. Create the ReportConfiguration by using the columns List, the
Filters List, the namespace, type, category(obtained using the getCategory() method)
and the styles Map.

13. Return the reportConfiguration.

1.3.5 Create ReportData as HTML table from ReportConfiguration for Time category Reports (TimeReport.executeReport() method)

1. ConfigManager configManager = ConfigManager.getInstance();

2. String dbHandler = configManager.getProperty(config.getNamespace(), "DBHandler");

3. DBHandlerFactory factory = getDBHandlerFactory();

4. DBHandler handler = factory.getDBHandler(dbHandler);

5. ResultSet rs = handler.getReportData(config);

6. Initialize the count for totalHours to zero.

7. Initialize a String "buffer" to "";

8. buffer = buffer + "<TABLE";

9. Map styles = config.getStyles();

10. buffer = buffer + " STYLE=\""+styles.get(StyleConstant.TABLE_STYLE)+"\"><TR
STYLE=\""+styles.get(StyleConstant.TR_STYLE)+"\">";

11. For each columnDecorator in config.getColumnDecorators() do:
buffer = buffer + "<TH
STYLE=\""+styles.get(StyleConstant.TH_STYLE)+"\">"+columnDecorator.getColumnDisplayText(

12. buffer = buffer+"</TR>";

13. For each of the row found in the ResultSet do:
a: buffer = buffer + <TR STYLE=\""+styles.get(StyleConstant.TR_STYLE)+"\">";
b: For each columnDecorator in config.getColumnDecorators() do:
i. Read data for the column from the ResultSet in some "temp" variable.

ii. If the columnName was "HOURS" add the value to totalHours.
iii.temp = columnDecorator.decorateColumn(temp);
iv. buffer = buffer + "<TD STYLE=\""+styles.get(StyleConstant.TD_STYLE)+"\">"+temp+"</

c: If column "HOURS" was not a part of the config.getColumnDecorators(), then read the value fro
resultset for the "HOURS" column and add it to the totalHours. (This is to ensure that Total Hours ge
for summarization purpose even though it might not be a part of the Columns being displayed in the r

d: buffer = buffer+"</TR>";

14. buffer = buffer+"</TABLE>";

15. Append totalinformation to the buffer Data. < buffer = buffer + "

<CENTER>Total H
"+totalHours +"</CENTER>";>

16. handler.release(rs); // Releasing the ResultSet and the corresponding connection.

17. Return the buffer.

1.3.6 Create ReportData as HTML table from ReportConfiguration for Expense category Reports (ExpenseReport.executeReport() method)

1. ConfigManager configManager = ConfigManager.getInstance();

2. String dbHandler = configManager.getProperty(config.getNamespace(), "DBHandler");

3. DBHandlerFactory factory = getDBHandlerFactory();

4. DBHandler handler = factory.getDBHandler(dbHandler);

5. ResultSet rs = handler.getReportData(config);

6. Initialize the count for totalAmount to zero.

7. Initialize a String "buffer" to "";

8. buffer = buffer + "<TABLE";

9. Map styles = config.getStyles();

10. buffer = buffer + " STYLE=\""+styles.get(StyleConstant.TABLE_STYLE)+"\"><TR
STYLE=\""+styles.get(StyleConstant.TR_STYLE)+"\">";

11. For each columnDecorator in config.getColumnDecorators() do:
buffer = buffer + "<TH
STYLE=\""+styles.get(StyleConstant.TH_STYLE)+"\">"+columnDecorator.getColumnDisplayText(

12. buffer = buffer+"</TR>";

13. For each of the row found in the ResultSet do:
a: buffer = buffer + <TR STYLE=\""+styles.get(StyleConstant.TR_STYLE)+"\">";
b: For each columnDecorator in config.getColumnDecorators() do:
i. Read data for the column from the ResultSet in some "temp" variable.
ii. If the columnName was "AMOUNT" add the value to totalAmount.
iii. temp = columnDecorator.decorateColumn(temp);
iv. buffer = buffer + "<TD STYLE=\""+styles.get(StyleConstant.TD_STYLE)+"\">"+temp+"</
c: If column "AMOUNT" was not a part of the config.getColumnDecorators(), then read the value
resultset for the "AMOUNT" column and add it to the totalAmount. (This is to ensure that Total Amo
calculated for summarization purpose even though it might not be a part of the Columns being displa
report.
d: buffer = buffer+"</TR>";

14. buffer = buffer+"</TABLE>";

15. Append totalinformation to the buffer Data. < buffer = buffer + "

<CENTER>Total A
"+totalAmount +"</CENTER>";>

16. handler.release(rs); // Releasing the ResultSet and the corresponding connection.

17. Return the buffer.

1.3.7 Create ReportData as HTML table from ReportConfiguration for TimeExpense category Reports (TimeExpenseReport.executeReport() method)

1. ConfigManager configManager = ConfigManager.getInstance();

2. String dbHandler = configManager.getProperty(config.getNamespace(), "DBHandler");

3. DBHandlerFactory factory = getDBHandlerFactory();

4. DBHandler handler = factory.getDBHandler(dbHandler);

5. ResultSet rs = handler.getReportData(config);

6. Initialize the count for totalAmount and totalHours to zero.

7. Initialize a String "buffer" to "";

8. buffer = buffer + "<TABLE";

9. Map styles = config.getStyles();

10. buffer = buffer + " STYLE=\""+styles.get(StyleConstant.TABLE_STYLE)+"\"><TR
STYLE=\""+styles.get(StyleConstant.TR_STYLE)+"\">";

11. For each columnDecorator in config.getColumnDecorators() do:
buffer = buffer + "<TH
STYLE=\""+styles.get(StyleConstant.TH_STYLE)+"\">"+columnDecorator.getColumnDisplayText(

12. buffer = buffer+"</TR>";

13. For each of the row found in the ResultSet do:
a: buffer = buffer + <TR STYLE=\""+styles.get(StyleConstant.TR_STYLE)+"\">";
b: For each columnDecorator in config.getColumnDecorators() do:
i. Read data for the column from the ResultSet in some "temp" variable.
ii. If the columnName was "AMOUNT" add the value to totalAmount.
iii.If the columnName was "HOURS" add the value to totalHours.
iv. temp = columnDecoration.decorateColumn(temp);
iv. buffer = buffer + "<TD STYLE=\""+styles.get(StyleConstant.TD_STYLE)+"\">"+temp+"</
c: If column "AMOUNT" was not a part of the config.getColumnDecorators(), then read the value
resultset for the "AMOUNT" column and add it to the totalAmount. (This is to ensure that Total Amo
calculated for summarization purpose even though it might not be a part of the Columns being displa
report.
d: If column "HOURS" was not a part of the config.getColumnDecorators(), then read the value fr
resultset for the "HOURS" column and add it to the totalHours. (This is to ensure that Total Hours ge
for summarization purpose even though it might not be a part of the Columns being displayed in the r
e: buffer = buffer+"</TR>";

14. buffer = buffer+"</TABLE>";

15. Append totalinformation for Hours to the buffer Data. < buffer = buffer + "

<CENTE
Hours: "+totalHours +"</CENTER>";>

16. Append totalinformation for Amount to the buffer Data. < buffer = buffer + "

<CENT
Amount: "+totalAmount +"</CENTER>";>

17. handler.release(rs); // Releasing the ResultSet and the corresponding connection.

18. Return the buffer.

1.3.8 Fetching of the Report data from the Database based on the ReportConfiguration (InformixDBHandler.() method)

1. ConfigManager manager = ConfigManager.getInstance();

2. String connectionName = manager.getProperty(namespace,"Connection");
String dbConnectionFactoryNamespace =
manager.getProperty(namespace,"DBConnectionFactoryNamespace");

3. DBManagerFactory factory = new
DBManagerFactoryImpl(dbConnectionFactoryNamespace);

4. Connection con = factory.getConnection(connectionName);

5. String reportType = config.getType().getType();

6. String reportCategory = config.getCategory().getCategory();

7. if(config.getCategory() == ReportCategory.TIMEEXPENSE)
then follow steps 8 to 17.
/* Needs special handling for TIMEEXPENSE reports because the queries for TIMEEXPENSE reports involve a union in the query and hence the filterpart of the query and the setting up of parameters for the query from the filter values are handled differently and needs to be looked at with more care. */

else // if the reports are of the category of TIME or EXPENSE.
Follow the steps 18 to 26.

8. String baseQuery =
manager.getProperty("com.topcoder.timetracker.report.QueriesConfiguration",
"BASE_QUERY_"+reportType+ "_" + reportCategory);

9. String baseQueryParts[] = baseQuery.split("UNION"); /* since the base query involves a union, we split the base query at the union point and then append the filter conditions to both the parts of the query and then join these two parts back with a union to get the final query.*/

10. Initialize "totalParams" to zero.

11. For each filter in config.getFilters() List do:
a. String colName = filter.getColumn().getName();
b. String filterQueryPart =
manager.getProperty("com.topcoder.timetracker.report.QueriesConfiguration",
"FILTER_"+colName+"_"+reportType+ "_" + reportCategory);

c. String filterparts[] = filterQueryPart.split("UNION"); /* since the filter part of the query for the TIMEEXPENSE reports are configured with a UNION and each side part of the UNION of the filterQueryPart will be appended to the corresponding part of the BaseQuery. */

d. int count = (filter instanceof RangeFilter)?
((RangeFilter)filter).getLowerRangeValues().size() :
((EqualityFilter)filter).getFilterValues().size();
e. For int i=0 to count do
i. baseQueryParts[0] = baseQueryParts[0] + " AND " + filterparts[0];

ii. baseQueryParts[1] = baseQueryParts[1] + " AND " + filterparts[1];
f. count = (filter instanceof RangeFilter)?count*2:count;
g. totalParams += count;

11. baseQuery = baseQueryParts[0] + " UNION " + baseQueryParts[1];

12. PreparedStatement statement = con.prepareStatement(baseQuery);

13. int counter = 0;

14. For each filter in config.getFilters() List do:
a. String colName = filter.getColumn().getName();
b. String colType =
manager.getProperty("com.topcoder.timetracker.report.ColumnTypes",
colName+"_TYPE");

c. if (filter instanceof RangeFilter) do
i. List lowerValues = ((RangeFilter)filter).getLowerRangeValues();
ii. List upperValues = ((RangeFilter)filter).getUpperRangeValues();
iii. int size = lowerValues.size();
iv. For int i =0 to size do
a. counter+=1;
b. String lowerVal = (String)lowerValues.get(i);
c. String upperVal = (String)upperValues.get(i);
d. Set the value of the parameter at location "counter" and "counter+(totalParams/2)" in the PreparedStatement "statement" with the value of "lowerVal", based on the type of the column specified by "colType".
e. counter += 1;
f. Similarly, set the value of parameter at location "counter" and "counter+(totalParams/2)" in the PreparedStatement "statement" with the value of "upperVal", based on the type of the column specified by "colType".

d. else // filter is an EqualityFilter
i. List values = ((EqualityFilter)filter).getFilterValues();
iii. int size = values.size();
iv. For int i =0 to size do
a. counter++;
b. String value = (String)values.get(i);
c. Set the value of the parameter at location "counter" and the location "counter+(totalParams/2) in the PreparedStatement "statement" with the value of "value", based on the type of the column specified by "colType".

15. ResultSet resultSet = statement.executeQuery();

16. resultsetConnectionMap.put(resultSet,con);

17. Return resultSet.

// For reports belonging to the category of TIME or EXPENSE reports.

18. String baseQuery =
manager.getProperty("com.topcoder.timetracker.report.QueriesConfiguration",
"BASE_QUERY_"+reportType+ "_" + reportCategory);

19. Initialize "totalParams" to zero.

20. For each filter in config.getFilters() List do:
a. String colName = filter.getColumn().getName();
b. String filterQueryPart =
manager.getProperty("com.topcoder.timetracker.report.QueriesConfiguration",
"FILTER_"+colName+"_"+reportType+ "_" + reportCategory);

c. int count = (filter instanceof RangeFilter)?
((RangeFilter)filter).getLowerRangeValues().size() :
((EqualityFilter)filter).getFilterValues().size();
d. For int i=0 to count do
i. baseQuery = baseQuery + " AND " + filterQueryPart;
e count = (filter instanceof RangeFilter)? count*2 : count;
f. totalParams += count;

21. PreparedStatement statement = con.prepareStatement(baseQuery);

22. int counter = 0;

23. For each filter in config.getFilters() List do:
a. String colName = filter.getColumn().getName();
b. String colType =
manager.getProperty("com.topcoder.timetracker.report.ColumnTypes",
colName+"_TYPE");

c. if (filter instanceof RangeFilter) do
i. List lowerValues = ((RangeFilter)filter).getLowerRangeValues();
ii. List upperValues = ((RangeFilter)filter).getUpperRangeValues();
iii. int size = lowerValues.size();
iv. For int i =0 to size do
a. counter+=1;
b. String lowerVal = (String)lowerValues.get(i);

c. String upperVal = (String)upperValues.get(i);
d. Set the value of the parameter at location "counter" in the PreparedStatement "statement" with the value of "lowerVal", based on the type of the column specified by "colType".
e. counter += 1;
f. Similarly, set the value of parameter at location "counter" in the PreparedStatement "statement" with the value of "upperVal", based on the type of the column specified by "colType".

d. else // filter is an EqualityFilter
i. List values = ((EqualityFilter)filter).getFilterValues();
iii. int size = values.size();
iv. For int i =0 to size do
a. counter++;
b. String value = (String)values.get(i);
c. Set the value of the parameter at location "counter" in the PreparedStatement "statement" with the value of "value", based on the type of the column specified by "colType".

24. ResultSet resultSet = statement.executeQuery();

25. resultsetConnectionMap.put(resultSet,con);

26. Return resultSet.

1.4 Component Class Overview

ReportDisplayTag: [class]

The ReportDisplayTag is a custom JSP Tag that is responsible for outputting the ReportData as a HTML table along with the summarization information.
Note: This tag is not a container tag and hence cannot contain subtags within this tag. This tag will render itself when the JSPContainer will process the start of this tag by calling the doStartTag() method.

During the processing of this tag (doStartTag() method) the following sequence of operations occur:
1. Read the required values of parameters from the PageContext object available to the tag.
2. Obtain a Report object for the particular Category.
3. Create a List containing the Filter objects, constructed based on the values of the parameters from the PageContext
4. Invokes the execute method of the Report, by passing the required parameters.

5. The return value of the execute method contains the ReportData as a HTML table along with the summarization information which is output.

This class is as such not threadsafe, but this class is invoked by the JSP Container and the thread-safe access to the class depends on the handling by the JSPContainer.

Report: [interface]

The Report interface defines the basic behavior of the report.
Whenever the Report is executed by calling the execute() method, the data for the Report is gathered based on the parameters passed.
The gathered data is then formatted based on the Format of the Report based used.
The namespace parameter defines the namespace from which the properties for the Report are fetched from the configuration file.
The formatted Report Data in the then returned as a String on the completion of the execution of the execute() method.

The implementations of this interface are required to be thread-safe.

AbstractReport: [abstract class]

AbstractReport class is an abstract class and is thread safe, as there are no setter methods to modify the internal variables.
This is the class which provides the basic functionality of gathering the required information for the Report from the configuration file, and leaves the rendering of the data to the subclasses.
The constructor of this class has been made protected primarily to disallow creation of objects of this class outside the package

Note that the formation of the query and its subsequent execution and the report rendering in the specific format is done in the subclasses.
This is because each Category of report might require different summarization information and hence would be required to handle the fetched data for the Report in different ways.

TimeReport: [class]

TimeReport class is a subclass of AbstractReport and is responsible for generating the ReportData and organizing it in the HTML Table Format for the Time Category Reports.
This class is thread-safe as the instance members of this class are not modifiable.

ExpenseReport: [class]

ExpenseReport class is a subclass of AbstractReport and is responsible for

generating the ReportData and organizing it in the HTML Table Format for the Expense Category Reports.
This class is also threadsafe as its instance variables are not modifiable.

TimeExpenseReport: [class]

TimeExpenseReport class is a subclass of AbstractReport and is responsible for generating the ReportData and organizing it in the HTML Table Format for the TimeExpense Category Reports.
This class is also threadsafe as its instance members are not modifiable.

Filter: [interface]

This interface defines the behavior of a Filter. A Filter is for a particular Column and a Filter also has a specific type, which defines its nature (equality filter or range filter or greater than filter etc.)

The implementers of this interface provide the behavior required based on its type.

The implementers of this interface may not be thread-safe. The component creates new Filter objects and ensures that it uses the Filter objects in a thread-safe manner.

RangeFilter: [class]

This class represents a Filter which of the type range. This filter defines that the values within the range specified by the Lower and Upper values needs to be selected.

This filter is multi-valued and can define set of ranges to be used.

This class is not thread-safe, but since the application creates new objects of this class for each of the reports and handles it in a thread-safe way, from the view of the entire component, this class will be used by the component in a thread-safe manner.

EqualityFilter: [class]

This class represents a Filter which of the type "equality". This filter defines that the values which are equal to the Filter value specified needs to be selected.

This filter is multi-valued and can define set of equality values to be used.

This class is not thread-safe, but since the application creates new objects of this class for each of the reports and handles it in a thread-safe way, from the view of the entire component, this class will be used by the component in a thread-safe

manner.

ColumnDecorator: [interface]

This interface basically defines the behavior of a ColumnDecorator.
A ColumnDecorator will be associated with a Column and will assist in formatting the Columns Data. Additionally it also contains the information as to what should be the Columns display text and the name of the column it is decorating.

The implementers of this class may not be thread-safe. However, the ColumnDecorator objects are created and used by this component in a thread-safe manner. (The ColumnDecorator objects will not be reused and will not be passed across threads).

BasicColumnDecorator: [class]

This class implements the ColumnDecorator interface. It provides a basic column decoration feature, wherein the columns data can be prefixed with a specified constant prefix and/or be suffixed with a specified constant string.

This class is not as such thread-safe, but will be used the component in a thread-safe manner. (The objects created will not be used across threads and not reusing the objects.)

ReportFactory: [class]

This class is factory and contains all the Report implementation registered by the application. The registration of the Report implementation happens automatically during the loading of the class (This is when the singleton instance is created) from the configuration information specified in the configuration file (Time_Tracker_Report.xml). This class allows the fetching of a Report registered by Format and Category.

This class is thread-safe as the instance members are not modifiable.

ReportConfiguration: [class]

This class represents the ReportConfiguration that is interpreted for processing and the display of the report. This contains the full information required for the rendering of the report. This class is as such not thread-safe, but the component uses this class in a thread-safe manner (no reusing of the instance, no use of the same instance across threads)

This class is used by the implementation of the Report and the DBHandler interface to fetch the data for the Report and display it.

DBHandlerFactory: [class]

This class is factory and contains all the DBHandler implementation registered by the application. The registration of the DBHandler implementation happens automatically during the loading of the class (This is when the singleton instance is created) from the configuration information specified in the configuration file (Time_Tracker_Report.xml). This class allows the fetching of a DBHandler registered by name.

This class is thread-safe as the instance members are not modifiable.

DBHandler: [interface]

This interface defines the basic behavior of a DBHandler. The DBHandler handles the Database operations like fetching of Data from the DataBase, managing the Connection to the database, setting and doing the necessary conversion for the parameters values used in the queries, creating the queries dynamically for the reports, executing the queries against the target Database and presenting the retrieved data in the form of a java.sql.ResultSet object.

The implementation of this class are required to handle the above mentioned operation in a manner specific to the Database is abstracts the usage.

The implementers of this class are supposed to be thread safe..

InformixDBHandler: [class]

This class implements the DBHandler interface and handles the Database specific operation defined by the interface for the Informix Database.

This class is thread-safe, because its instance member "baseQueries" is immutable and the access to the only other instance member "resultsetConnectionMap" is synchronized. (By making the map a synchronized Map).

ColumnType: [enumeration class]

This class is an enumeration and defines the different Data types for the Database column supported by the component. This enumeration defines the supported DataTypes and the corresponding DBHandler implementation would require to map this information to the DataTypes that are supported by the corresponding Database. As an example, the DataType for "MONEY" is not as such available with most of the Database and the DBHandler implementation would require to map it to the corresponding data type and handle the necessary conversion required.

This is an enumeration class and is thread-safe.

ReportType: [enumeration class]

This is an Enumeration class and is thread-safe. This class defines the different types of Reports possible.

FilterCategory: [enumeration class]

This is an enumeration class and is hence Thread-Safe.
This class defines the different categories of Filter for the reports of this component. It is important to note that all the FilterCategory defined here may not be applicable for all the Reports.

FilterType: [enumeration class]

This class is an enumeration and hence is Thread-safe. This enumeration defines the different types of Filters possible as a part of this component.

ReportCategory: [enumeration class]

This class is an enumeration class and hence is thread-safe. This enumeration class defines the different category of Reports possible.

Column: [enumeration class]

This class is an enumeration and hence is thread-safe. This enumeration class defines all the Columns used in the Time Tracker Report component.

The Columns here are not the actual Database column but the column as identified by the application. It is possible that the name of the application column is same as the Database column, but not necessary. For simplicity of understanding it has been kept same. It will be different if 2 or more tables have the column with the same name but different column types. If 2 or more tables have column with the same name and also the same data type, then the application level column name can be the same for them, but if the data type differs, then the application level column name has to be different.

StyleConstant: [enumeration class]

This is an enumeration and hence the class is Thread-safe.
This class defines constants for the property names for the CSS Styles specified in the configuration file for the Report (Time_Tracker_Report.xml file).

1.5 Component Exception Definitions

ReportException[Custom]:

This is the Base exception of all the exceptions that could occur in generating the Custom Report. This exception is thrown by the execute() method of the Report interface. Basically this Exception is thrown if there are any problems during the report generation.

This exception is sub-classed to throw the exceptions during specific operations during the report generation.

Example: 1. ReportSQLException: if there are problems in DataBase access or fetch of the Report Data from the Database.

2. ReportConfigurationException: if there are errors in the configuration information for a report or if there are errors during the reading of information using the ConfigManager

ReportConfigurationException[Custom]:

This exception is thrown if any Invalid configuration is detected for a report. This includes:

1. Missing required parameters in the configuration.
2. Invalid Filters are specified for a report. (Filters not applicable for a particular report are specified in the Configuration)
3. Missing parameter values for mandatory filter for a report.
4. Used as a wrapper for wrapping an exception thrown from the ConfigurationManager component.

ReportSQLException[Custom]:

This exception is thrown, if there is an exception during access to the Database or during the fetch of the data from the Database. This exception is basically a wrapper over the java.sql.SQLException and indicates an exception during Database operation.

ReportNotFoundException[Custom]:

This exception indicates that the particular Report requested is not registered.

During the loading of the ReportFactory, the Report instances are loaded through reflection after reading the corresponding properties from the configuration file and registered with the ReportFactory.

When a request for a particular Report is made and there is no Report registered for the specified ReportCategory and Format in the ReportFactory, then this exception is thrown to indicate the same.

DBHandlerNotFoundException[Custom]:

This exception indicates that the particular DBHandler requested is not registered.

During the loading of the DBHandlerFactory, the DBHandler instances are loaded through reflection after reading the corresponding properties from the configuration file and registered with the name specified in the configuration.

When a request for a particular DBHandler is made and there is no DBHandler registered with the requested name in the DBHandlerFactory, then this exception is thrown to indicate the same.

JSPException

This exception will be thrown if any error occurs when starting or ending the JSP tags. It would wrap any other exceptions thrown in the doStartTag() method.

NullPointerException

This exception will be thrown when the argument is null where this is disallowed. Refer to the documentation tab for details.

IllegalArgumentException

This exception will be thrown when an invalid argument is used. Typically given String parameter is empty. Refer to the documentation for details.

1.6 Thread Safety

This component should be thread-safe. The user accesses just the JSP page which will create an instance of the ReportDisplayTag, set the attribute values using the setter methods and then call the doStartTag() method. This method delegates it to the Report instance the execution after obtaining the Report instance from the ReportFactory.
All the Report instances are thread-safe and also only new Objects for Filter, ReportConfiguraton and ColumnDecorator are created. Since the non thread-safe classes have been used in a thread-safe manner [using only new instances of these classes for requests and not using these objects across threads] the component is thread-safe. Also the ReportFactory, DBHandlerFactory and the DBHandler implementations are also thread-safe.

2. Environment Requirements

2.1 Environment

- Development language: Java1.4
- Compile target: Java1.3, Java1.4

2.2 TopCoder Software Components

- Configuration Manager: 2.1.3
- DB Connection Factory: 1.0.
- BaseException: 1.0.
- TypeSafe Enum: 1.0.

This component does NOT depend on the Report Tags component and the reasons for it are:

1. The Report Tags component does not support the usage of CSS styles.
2. The Report Tags component does not support the creation of dynamic queries, which is required in our case. Report tags require the queries to be a part of the static configuration.
3. The Report Tags component does not provide rendering of the entire tables with contents, but only table tag and the table header tags.

This component handles by itself the rendering of the Report Data as an HTML table, with support for CSS styles and dynamic query construction.

NOTE: The default location for TopCoder Software component jars is../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.

2.3 Third Party Components

- Informix Database is being used.

3. Installation and Configuration

3.1 Package Name

1. com.topcoder.timetracker.report [For the framework classes of the component]

2. com.topcoder.timetracker.report.htmlreport [For classes handling the Report display in HTML Table format.]

3. com.topcoder.timetracker.report.dbhandler [For classes handling the persistence logic]

3.2 Configuration Parameters

(The configuration file Time_Tracker_Report.xml and time_tracker_report.tld is also provided in /docs folder.)

The configuration file "Time_Tracker_Report.xml" contains all the configuration information required by this component. This configuration file is required to be loaded by the "Configuration Manager" component and the component needs to be configured for the same. The configuration file defines the following:

1. One DefaultConfiguration for report and any number of CustomConfiguation for report. (The "Time_Tracker_Report.xml" defines one custom configuration, but defines all the properties possible in the report configuration and could be used as an example to define further more Custom Configurations.)
2. The configuration for the Queries(Base Query configuration and the Filter extensions to the query).
   Note: The Filter extension for the query for the report defines the conditions (part of the query) that need to be added(appended) when optional filters for the report are present. As a result the final query for a report is dynamically generated from the Base query and Filter Extensions to the query for the report, based on the filters for the report being used.
3. The configuration for valid Filters for a report.
   (This is required for validating whether a Filter specified is actually valid for

report, since filters for report can be specified dynamically).

4. The configuration for default set of columns for a report.
   (This is required for 2 reasons:
   a. Validating whether the Columns requested for display (if defined) in Custom Configuration for report are valid for the report.
   b. Display the default set of Columns for the report if Default Configuration for report is used or Custom Configuration does not override the columns to be displayed for a report.)

Note: Please go through the "Time_Tracker_Report.xml" file, as it contains detailed explanation for the various configuration properties.

1. Report Configuration

Parameter	Description	Example Values
DBConnectionFactoryNamespace	The namespace to be used when initializing the DBConnectionFactoryImpl instance. (Required).	"com.topcoder.timetracker.report.Informix"
Connection	The name with which the connection producer for the Informix Database used by the TimeTracker application is registered with the DBConnectionFactory(Required).	"InformixConnection"
Column properties. This includes the property names for all the columns that are possible across the nine different reports. The property names are: COLUMN_DATE, COLUMN_CLIENT, COLUMN_PROJECT, COLUMN_TASK, COLUMN_HOURS, COLUMN_PAY_RATE, COLUMN_BILLABLE, COLUMN_EMPLOYEE, COLUMN_TYPE, COLUMN_DESCRIPTION and COLUMN_AMOUNT	These are optional properties and the value of the property defines the Label to be used when the corresponding column is rendered in the report.	Label for the particular column for the report.
Prefix properties. These properties define the column prefix to be used when rendering the column data. The names of these properties can be obtained by prefixing the literal "PREFIX_" to the column properties names. Eg:PREFIX_COLUMN_DATE	These are optional properties. These properties define the column prefix to be used when rendering the column data.	Prefix for particular column data. (eg. $ for Amount columns)
Suffix properties. These	These are optional properties. These	Suffix for particular

properties define the column suffix to be used when rendering the column data. The names of these properties can be obtained by prefixing the literal “SUFFIX_” to the column properties names. Eg:SUFFIX_COLUMN_DATE	properties define the column suffix to be used when rendering the column data.	column data. (eg. $ for Amount columns)
TABLE_STYLE	The CSS Style to be used in the Table HTML tag. (Optional)	background: blue; color: white;
TH_STYLE	The CSS Style to be used in the TH (Table Header) HTML tag. (Optional)	background: grey; color: white;
TR_STYLE	The CSS Style to be used in the TR (Table Row) HTML tag. (Optional)	color: white;
TD_STYLE	The CSS Style to be used in the TRD HTML tag. (Optional)	color: white;

Sample Report Configuration:

```
<!-- Default Configuration for the reports. All reports irrespective of the
type or the category they belong to, can use this Configuration. This
configuration just specifies the value for the Connection property, which is a
required parameter for the Configuration. All other properties are optional for
the configuration.

The namespace attribute in the "reportdisplay" JSP tag can specify this
Configuration name as its value.
-->

<Config name="com.topcoder.timetracker.report.DefaultConfiguration">
      <Property name="Connection">
          <Value>InformixConnection</Value>
      </Property>
</Config>


<!--  Customized Configuration for the report. This Configuration shows us an
example as to what parameters could be provided in a Configuration and whats
the intended use of the value of the parameter in Report generation.
The namespace attribute in the "reportdisplay" JSP tag can specify this
Configuration name as its value.
-->

<Config name="com.topcoder.timetracker.report.CustomConfiguration">

      <!-- Required Property For the Configuration. -->
      <Property name="Connection">
            <Value>InformixConnection</Value>
      </Property>

      <!-- Optional Properties for the Configuration -->
```

```
        <!-- Column Properties.
               These set of properties specify the Column Labels to be used for
               display during the rendering of the report. If no column property
               is specified then the default set of Columns for the report with
               the default Columns Labels will be used. If Column properties are
               specified, then only the columns specified by the column property
               and applicable to the report under consideration will be
               displayed. The names of the Column properties for the reports
               under consideration are:

                  COLUMN_DATE, COLUMN_CLIENT, COLUMN_PROJECT, COLUMN_TASK,
                  COLUMN_HOURS, COLUMN_PAY_RATE, COLUMN_BILLABLE,
                  COLUMN_EMPLOYEE, COLUMN_TYPE, COLUMN_DESCRIPTION and
                  COLUMN_AMOUNT.

               The following property defines that only the columns for Employee
               and Amount will be displayed for the report and the Column Labels
               in the report display will be "Employee Name" and "Amount Spent"
               respectively.
         -->

        <Property name="COLUMN_EMPLOYEE">
               <Value>Employee Name</Value>
        </Property>

        <Property name="COLUMN_AMOUNT">
               <Value>Amount Spent</Value>
        </Property>


        <!-- Prefix and Suffix Properties.
               These set of properties specify the prefix and the suffix for a
               particular Column. This is useful for supporting when certain
               prefix or suffix needs to be added to the Column data during
               display, an example being the display of "$" symbol as prefix for
               AMOUNT Column.

               Note that only the prefixes and the suffixes for the columns being
               display will be considered, others will be ignored.

               The names of the prefix and the suffix properties are obtained by
               concatenating the "PREFIX_" and the "SUFFIX_" literals
               respectively to the COLUMN properties.
                 Examples of prefix and suffix property names are:
                      PREFIX_COLUMN_DATE, PREFIX_COLUMN_CLIENT,
                      SUFFIX_COLUMN_TASK, SUFFIX_COLUMN_CLIENT etc.
         -->

        <Property name="PREFIX_COLUMN_AMOUNT">
               <Value>$</Value>
        </Property>

        <!-- Properties for CSS support. Note that not all properties required to
        be specified.  -->

        <!-- The CSS Style to be used in the Table HTML tag -->

        <Property name="TABLE_STYLE">
               <Value>background: blue; color: white;</Value>
         </Property>

         <!-- The CSS Style to be used in the TH (Table Header) HTML tag -->
         <Property name="TH_STYLE">
```

```
            <Value>background: grey; color: white;</Value>
        </Property>

        <!-- The CSS Style to be used in the TR (Table Row) HTML tag -->
        <Property name="TR_STYLE">
            <Value>color: white;</Value>
        </Property>

        <!-- The CSS Style to be used in the TRD HTML tag -->
        <Property name="TD_STYLE">
            <Value>color: white;</Value>
        </Property>

    </Config>
```

2. Queries Configuration

The Queries Configuration defines the BaseQuery and FilterExtensions for a report. The complete Queries Configuration is specified in the "Time_Tracker_Report.xml" configuration for all the 9 different reports used by the Time Tracker application. The configuration name that has been defined for these set of properties in the "Time_Tracker_Report.xml" configuration file is "com.topcoder.timetracker.report.QueriesConfiguration". The following table defines the names of the properties and description about what the property is meant for:

Parameter	Description	Example Values
BaseQuery properties. These properties define the "Base Query" for the different type and the category of the Report.	These are required properties. These properties define the "Base Queries" for the report. (Required) The property name for the "Base Query" for the report is in the format BASE_QUERY_<REPORT_TYPE>_<REPORT_CATEGORY>. **Note:** Base Query implies the query for the report without the filter conditions.	Example values can be looked in the sample configuration given below.
"FilterExtension for query" properties. These properties define the "Filter extensions" that should be appended to the BaseQuery of the report to form the final query. The filter extensions will be appended based on the values for the filters passed by the user. It means a Filter extension will not be appended to the base query if the filter (filter value) is not	These are required properties. These properties define the filter extensions to the "Base Queries" for the different type and category of the Report. (Required) The property name for the "Filter Extensions for Query" for the report is in the format <FILTER_NAME>_<REPORT_TYPE>_<REPORT_CATEGORY>. **Note:** These are the parts of the query which are appended to the Base	Example values can be looked in the sample configuration given below.

by the user.	Query of the report to form the final query which is executed to fetch the data for the report.	

The following sample configuration shows the configuration for “Employee Time Report”.

Sample Queries Configuration:

```
<!--
       This is a required configuration and specifies the "Base Query" and the
       FilterExtension of Query for each of the Report.
       The property name for the "Base Query" for the report is in the format
       BASE_QUERY_<REPORT_TYPE>_<REPORT_CATEGORY>. eg. BASE_QUERY_EMPLOYEE_TIME

       The property name for the "FilterExtension for query" for a report is in
       the format <FILTER_NAME>_<REPORT_TYPE>_<REPORT_CATEGORY>.
       eg. FILTER_DATE_EMPLOYEE_TIME
       Note that the <REPORT_TYPE>, <REPORT_CATEGORY> and <FILTER_NAME>
       corresponds to the constants values defined by the enumeration classes
       ReportyType, ReportCategory and FilterCategory class.
      -->

<Config name="com.topcoder.timetracker.report.QueriesConfiguration">


       <!-- Query Configuration and FilterExtensions for the "Employee Time
       Report" -->

             <Property name="BASE_QUERY_EMPLOYEE_TIME">
                <Value>
                   SELECT TE.ENTRYDATE DATE,
                          CL.NAME CLIENT,
                          PR.NAME PROJECT,
                          TT.DESCRIPTION TASK,
                          TE.HOURS HOURS,
                          PW.PAYRATE PAY_RATE,
                          TE.BILLABLE BILLABLE

                     FROM TIMEENTRIES TE,
                          TASKTYPES TT,
                          CLIENTS CL,
                          PROJECTTIMES PT,
                          PROJECTS PR,
                          CLIENTPROJECTS CP,
                          PROJECTWORKERS PW,
                          USERS US

                    WHERE TE.TASKTYPESID = TT.TASKTYPESID
                      AND PT.TIMEENTRIESID = TE.TIMEENTRIESID
                      AND PT.PROJECTSID = PR.PROJECTSID
                      AND PW.PROJECTSID = PR.PROJECTSID
                      AND CL.CLIENTSID = CP.CLIENTSID
                      AND CP.PROJECTSID = PR.PROJECTSID
                      AND US.USEDRSID = PW.USERSID
                      AND US.NAME = TE.CREATIONUSER
                </Value>
             </Property>

              <Property name="FILTER_EMPLOYEE_EMPLOYEE_TIME">
```

```
                <Value>TE.CREATIONUSER = ?</Value>
            </Property>

            <Property name="FILTER_DATE_EMPLOYEE_TIME">
                <Value>TE.ENTRYDATE BETWEEN ? AND ?</Value>
            </Property>

            <Property name="FILTER_CLIENT_EMPLOYEE_TIME">
                <Value>CL.NAME = ?</Value>
            </Property>

            <Property name="FILTER_PROJECT_EMPLOYEE_TIME">
                <Value>PR.NAME = ?</Value>
            </Property>

            <Property name="FILTER_BILLABLE_EMPLOYEE_TIME">
                <Value>TE.BILLABLE = ?</Value>
            </Property>

         <!—Similar configurations for other 8 reports can be found in the
      Time_Tracker_Report.xml configuration file →

</Config>
```

3. Filters Configuration

The Filters Configuration defines the mandatory and optional filters for a report. The complete Filters Configuration is specified in the "Time_Tracker_Report.xml" configuration for all the 9 different reports used by the Time Tracker application. The configuration name that has been defined for these set of properties in the "Time_Tracker_Report.xml" configuration file is "com.topcoder.timetracker.report.FiltersConfiguration".

The following table defines the names of the properties and description about what the property is meant for:

Parameter	Description	Example Values
Mandatory Filter property. This property defines the "Mandatory Filter" whose value should be provided for the report to be executed.	This is a required property and defines the value of the "Mandatory Filter" for a report.(Required) The property name for the "Mandatory filter" for the report is in the format <REPORT_TYPE>_<REPORT_CATEGORY>_MANDATORY_FILTER.	Example values can be looked in the sample configuration given below.
Optional Filter property. This property is multi-valued and each value defines a filter which is optional for the report.	This is a required property and defines the value of the "Optional Filters" for a report.(Required) The property name for the "Optional filters" for the report is in the format <REPORT_TYPE>_<REPORT_CATE	Example values can be looked in the sample configuration given below.

	GORY>_OPTIONAL_FILTER.	

The following sample configuration shows the configuration for “Employee Time Report”.

Sample Queries Configuration:

```
<!--
This is a required configuration and specifies the Filters valid for a
particular Report and also the one mandatory filter for the report.

The property name for the mandatory filter for a report is in the format
<REPORT_TYPE>_<REPORT_CATEGORY>_MANDATORY_FILTER.
The property name for the optional filters for a report is in the format
<REPORT_TYPE>_<REPORT_CATEGORY>_OPTIONAL_FILTER.

Note that a filter which is mandatory cannot be a part of the optional filter.
Note that the value for the property is from the Filter constants defined by
the enumeration constants in the enumeration
com.topcoder.timetracker.report.FilterCategory.

-->

<Config name="com.topcoder.timetracker.report.FiltersConfiguration">

      <!-- Filter Configuration for the "Employee Time Report" -->

      <Property name="EMPLOYEE_TIME_MANDATORY_FILTER">
            <Value>FILTER_EMPLOYEE</Value>
      </Property>

      <Property name="EMPLOYEE_TIME_OPTIONAL_FILTER">
            <Value>FILTER_DATE</Value>
            <Value>FILTER_CLIENT</Value>
            <Value>FILTER_PROJECT</Value>
            <Value>FILTER_BILLABLE</Value>
      </Property>

      <!— Similar configurations for other 8 reports can be found in the
      Time_Tracker_Report.xml configuration file →

</Config>
```

4. Columns Configuration

The Columns Configuration defines the default set of columns for a report. The complete Columns Configuration is specified in the “Time_Tracker_Report.xml” configuration for all the 9 different reports used by the Time Tracker application.

The configuration name that has been defined for these set of properties in the “Time_Tracker_Report.xml” configuration file is “com.topcoder.timetracker.report.ColumnsConfiguration”.

The following table defines the names of the properties and description about what the property is meant for:

Parameter	Description	Example Values
Columns property. This property defines the valid and the default set of columns for a particular type and category of Report.	This is a required property and is multi-valued and defines the valid and the default set of columns for a particular type and category of Report. (Required) The property name for the “Columns property” for the report is in the format <REPORT_TYPE>_<REPORT_CATEGORY>_COLUMNS.	Example values can be looked in the sample configuration given below.

The following sample configuration shows the configuration for “Employee Time Report”.

Sample Columns Configuration:

```
<!--
This is a required configuration and specifies the default and also the valid
set of columns for a report, since by default all the columns valid for a
report are available.
The property name for the default columns for a report is in the format
<REPORT_TYPE>_<REPORT_CATEGORY>_COLUMNS.
 Eg. EMPLOYEE_TIME_COLUMNS
Note that the values for these properties will have a value from the
enumeration constants defined in the enumeration
com.topcoder.timetracker.report.Column class.

Note that these properties are multivalued.
-->


<Config name="com.topcoder.timetracker.report.ColumnsConfiguration">

      <!-- Default Columns Configuration for the "Employee Time Report" -->

      <Property name="EMPLOYEE_TIME_COLUMNS">
            <Value>DATE</Value>
            <Value>CLIENT</Value>
            <Value>PROJECT</Value>
            <Value>TASK</Value>
            <Value>HOURS</Value>
            <Value>PAY_RATE</Value>
            <Value>BILLABLE</Value>
      </Property>

      <!— Similar configurations for other 8 reports can be found in the
      Time_Tracker_Report.xml configuration file →

</Config>
```

Note: The “Time_Tracker_Report.xml” contains the complete configuration for the component (includes all **Queries** and configurations required for all the 9 different reports used by the Time Tracker application).

TLD configurations for tag:

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

<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
 "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>cr</short-name>
  <uri>com.topcoder.timetracker.report</uri>
  <description>Tag Library for Displaying Custom Reports</description>

        <tag>
          <name>reportdisplay</name>
          <tag-class>com.topcoder.timetracker.report.ReportDisplayTag</tag-class>
          <body-content>empty</body-content>
          <description>Display Custom Report with CSS support</description>

          <attribute>
            <name>namespace</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
            <description>The namespace to load report configuration.</description>
          </attribute>

          <attribute>
             <name>type</name>
             <required>true</required>
             <rtexprvalue>true</rtexprvalue>
             <description>The parameter name in the HttpRequest object or the attribute
                in the HttpSession object, whose value specifies the type of the
                report(Employee,Project or Client Report).
             </description>
          </attribute>

          <attribute>
            <name>category</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
            <description>The parameter name in the HttpRequest object or the attribute
               in the HttpSession object, whose value specifies the category of the
               report(Time,Expense or TimeExpense Report).
            </description>
          </attribute>

          <attribute>
             <name>clientFilter</name>
             <required>false</required>
             <rtexprvalue>true</rtexprvalue>
             <description>The parameter name in the HttpRequest object or the
                 attribute in the HttpSession object, whose value specifies the Client
                 Name to be used as a filter for the Report.
             </description>
          </attribute>

          <attribute>
             <name>employeeFilter</name>
             <required>false</required>
             <rtexprvalue>true</rtexprvalue>
             <description>The parameter name in the HttpRequest object or the
```

```
                attribute in the HttpSession object, whose value specifies the
                Employee Name to be used as a filter for the Report.
            </description>
        </attribute>

        <attribute>
            <name>projectFilter</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
            <description>The parameter name in the HttpRequest object or the
                attribute in the HttpSession object, whose value specifies the Project
                Name to be used as a filter for the Report.
            </description>
        </attribute>

        <attribute>
            <name>billableFilter</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
            <description>The parameter name in the HttpRequest object or the
                attribute in the HttpSession object, whose value specifies if only
                Billable entries needs to be considered for the Report.
            </description>
        </attribute>

        <attribute>
            <name>startDateFilter</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
            <description>The parameter name in the HttpRequest object or the
                attribute in the HttpSession object, whose value specifies the Start
                range for the Date Entries that need be considered for the Report.
            </description>
        </attribute>

        <attribute>
            <name>endDateFilter</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
            <description>The parameter name in the HttpRequest object or the
                attribute in the HttpSession object, whose value specifies the End
                range for the Date Entries that need be considered for the Report.
            </description>
        </attribute>

    </tag>

</taglib>
```

3.3 Dependencies Configuration

- Please review the “DB Factory” Component and the “Configuration Manager” component to make it work.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Execute ‘ant test’ within the directory that the distribution was extracted to.

4.2 Required steps to use the component

1. The configuration file “Time_Tracker_Report.xml” is loaded by the “Configuration Manager” component. Please refer the “Configuration Manager” component for details to load the configuration file by the component.
2. The TLD definition mentioned in the file “time_tracker_report.tld” (also mentioned in section 3.2) is available to the JSP container at runtime.
3. The JSP page makes use of the “reportdisplay” tag defined in the TLD, by specifying the required attributes for the tag usage and also the optional attributes of the tag which might be required for the particular report.
4. The *Filter* values for a report (required filter values and also additional filter values (defined to be used in the tag) for a report) is required to be passed by the application calling the JSP page containing the tag as a part of the HttpRequest parameter or as an attribute in the HttpSession object.

4.3 Demo

1. Employee Time Report with just the Employee Filter and Default Report Configuartion:

   This will demonstrate the how Employee Time Report can be displayed on the JSP page. This example considers the display of the report with just the mandatory *Filter* for the report(Employee Filter).

Code for the Sample JSP Page for the Report: (Sample1.jsp)

```
<HTML>
<TITLE> Time Tracker Report </TITLE>
<BODY>

<!-- Render the Employee Time Report.
     The namespace specifies to use Default Configuration. -->
<report:reportdisplay
   namespace="com.topcoder.timetracker.report.DefaultConfiguration"
   type="Report_Type"
   category="Report_Category"
   employee_filter="Filter1">
</report:reportdisplay>

</BODY>
</HTML>
```

Note: Please refer the “Time_Tracker_Report.xml” configuration file or section 3.2 of the document for details of the configuration of

"com.topcoder.timetracker.report.DefaultConfiguration" namespace.

Code for the an HTML page invoking the JSP Page (Sample1.jsp)for the Report:

```
<FORM ACTION="Sample1.jsp">
Enter Employee Name:

<INPUT TYPE = "TEXT" NAME="Filter1" Value="EmployeeName"/>
<BR>

<INPUT TYPE="HIDDEN" NAME ="Report_Type" Value="EMPLOYEE"/>

<INPUT TYPE="HIDDEN" NAME ="Report_Category" Value="TIME"/>

<INPUT TYPE="SUBMIT" NAME ="Submit" Value="Submit"/>

</FORM>
```

Note: The colors have been specifically used to indicate that the value for the attributes (shown in color) in the "reportdisplay" tag should be passed by the application calling the JSP Page as parameters in the HttpRequest object or as attribute in the HttpSession object.
In the above example "type" of the report is the value of the "Report_Type" parameter which is "EMPLOYEE".
Similarly the "category" of the report is the value of the "Report_Category" parameter which is "TIME".
On similar lines the value for employee_filter for the report is the value of the parameter "Filter1" which is "EmployeeName".
So the report that will be displayed is the EMPLOYEE TIME report with the mandatory filter for the report (employee_filter) only with the value for the filter as "EmployeeName".

2. Employee Expense Report with Multiple Filters and Default Report Configuration:

This will demonstrate the how Employee Expense Report can be displayed on the JSP page. This example considers the display of the report with Multiple *Filters* for the report (Employee Filter, Date Filter and Billable Filter).

Code for the Sample JSP Page for the Report: (Sample2.jsp)

```
<HTML>
<TITLE> Time Tracker Report </TITLE>
<BODY>

<!-- Render the Employee Expense Report.
     The namespace specifies to use Default Configuration. -->
<report:reportdisplay
   namespace="com.topcoder.timetracker.report.DefaultConfiguration"
   type="Report_Type"
   category="Report_Category"
   employee_filter="Filter1"
   startdate_filter=”StartDate_Filter”
   enddate_filter=”EndDate_Filter”>
   billable_filter=”Filter21”>
</report:reportdisplay>

</BODY>
</HTML>
```

Code for the an HTML page invoking the JSP Page (Sample2.jsp)for the Report:

```
<FORM ACTION=”Sample2.jsp”>

Employee Name:
<INPUT TYPE = “TEXT” NAME=”Filter1” Value=”EmployeeName”/>
<BR>

Start Date:
<INPUT TYPE = “TEXT” NAME=” StartDate_Filter” Value=”09-21-2004”/>
<BR>



End Date:
<INPUT TYPE = “TEXT” NAME=” EndDate_Filter” Value=”01-01-2005”/>
<BR>

Billable:
<INPUT TYPE = “TEXT” NAME=” Filter21” Value=”1”/>
<BR>

<INPUT TYPE=”HIDDEN” NAME =”Report_Type” Value=”EMPLOYEE”/>

<INPUT TYPE=”HIDDEN” NAME =”Report_Category” Value=”EXPENSE”/>

<INPUT TYPE=”SUBMIT” NAME =”Submit” Value=”Submit”/>
```

```
</FORM>
```

3. Project Time Report with just the Project Filter and Custom Configuartion:

This will demonstrate the how Project Time Report can be displayed on the JSP page. This example considers the display of the report with just the mandatory *Filter* for the report (Project Filter).

Code for the Sample JSP Page for the Report: (Sample3.jsp)

```
<HTML>
<TITLE> Time Tracker Report </TITLE>
<BODY>

<!-- Render the Project Time Report.
     The namespace specifies to use Custom Configuration. -->
<report:reportdisplay
   namespace="com.topcoder.timetracker.report.CustomConfiguration"
   type="Report_Type1"
   category="Report_Category1"
   project_filter="Project_Filter">
</report:reportdisplay>

</BODY>
</HTML>
```

Note: Please refer the “Time_Tracker_Report.xml” configuration file or section 3.2 of the document for details of the configuration of “com.topcoder.timetracker.report.CustomConfiguration” namespace.

Code for the an HTML page invoking the JSP Page (Sample3.jsp)for the Report:

```
<FORM ACTION=”Sample3.jsp”>
Project Name:

<INPUT TYPE = “TEXT” NAME=” Project_Filter” Value=”MyProject”/>
<BR>

<INPUT TYPE=”HIDDEN” NAME =”Report_Type1” Value=”PROJECT”/>

<INPUT TYPE=”HIDDEN” NAME =”Report_Category1” Value=”TIME”/>
```

```
<INPUT TYPE=”SUBMIT” NAME =”Submit” Value=”Submit”/>

</FORM>
```

4. Client TimeExpense Report with Multiple Filters and Custom Configuration:

This will demonstrate the how Client TimeExpense Report can be displayed on the JSP page. This example considers the display of the report with Multiple *Filters* for the report (Client Filter and Billable Filter).

Code for the Sample JSP Page for the Report: (Sample4.jsp)

```
<HTML>
<TITLE> Time Tracker Report </TITLE>
<BODY>

<!-- Render the Client TimeExpense Report.
     The namespace specifies to use Custom Configuration. -->
<report:reportdisplay
   namespace="com.topcoder.timetracker.report.CustomConfiguration"
   type="Type"
   category="Category"
   client_filter="MyFilter1"
   billable_filter=”Billable”>
</report:reportdisplay>

</BODY>
</HTML>
```

Code for the an HTML page invoking the JSP Page (Sample4.jsp)for the Report:

```
<FORM ACTION=”Sample2.jsp”>

Client Name:
<INPUT TYPE = “TEXT” NAME=”MyFilter1” Value=”ClientName”/>
<BR>

Billable:
<INPUT TYPE = “TEXT” NAME=” Billable” Value=”0”/>
<BR>
```

```
<INPUT TYPE=”HIDDEN” NAME =”Type” Value=”CLIENT”/>

<INPUT TYPE=”HIDDEN” NAME =”Category” Value=”TIMEEXPENSE”/>

<INPUT TYPE=”SUBMIT” NAME =”Submit” Value=”Submit”/>

</FORM>
```

The Demos have shown how using the same JSP tag, the different types of reports can be generated and this flexibility is provided by the design.

5. Future Enhancements

- Tags can be added with additional attributes to support CSS configuration as a Runtime configuration.
- Additional formats for the reports can be supported.
- Additional filters and ColumnDecorators can be supported.