[TopCoder]

Project Payment Calculator 1.0 Component Specification

1. Design

The Project Payment Calculator component defines a framework for calculating default project payments and provides implementations of the payment calculators.

This component defines ProjectPaymentCalculator interface which provides the contract for getting the project payments for a given project and a list of resource roles IDs.

DefaultProjectPaymentCalculator defined in this component is a concrete implementation of the ProjectPaymentCalculator interface which calculates the project payments using a linear formula and by retrieving the needed parameters to be used for computing the project payments from the database, and additionally provides another method which gets the payment of a single resource role based on the provided input parameters.

1.1 Design Patterns

Strategy pattern – ProjectPaymentCalculator and its implementations can be used in some external context as pluggable strategies. ProjectPaymentCalculator and its implementation is used as a strategy in the ProjectPaymentAdjustmentCalculator context.
Decorator pattern – ProjectPaymentAdjustmentCalculator is a decorating implementation of ProjectPaymentCalculator interface.

1.2 Industry Standards

JDBC, SQL, XML (for configuration).

1.3 Required Algorithms

1.3.1 Logging

Logging should be performed in all public methods (and optionally private/protected methods) of BaseProjectPaymentCalculator, DefaultProjectPaymentCalculator and ProjectPaymentAdjustmentCalculator defined in this component.

BaseProjectPaymentCalculator aggregates the Logging Wrapper Log implementation instance that will be used for logging errors and activity. If it is null this means that logging is not required to be performed.

It will log errors at ERROR level, and method entry/exit, input/output information at DEBUG level.

Specifically, logging will be performed as follows.

- Method entrance and exit will be logged at DEBUG level.
    - Entrance format: Entering method [*className#methodName*]
    - Exit format: Exiting method [*className#methodName*]. Only do this if there is no exception
- Method input parameters and response parameters will be logged at DEBUG level
    - Format for method parameters: Input parameters *[paramName1:paramValue1, paramName2:paramValue2, …, paramNameN:paramValueN]*
    - Format for the response: Output parameter: *return value*. Only do this if there is no exception and method is not void
- All exceptions will be logged at ERROR level, and automatically log inner exceptions as well.
    - Format: Simply log the text of exception: Error in method [*className#methodName*], details: *error details*
    - The stack trace of the error and a meaningful message.

In general, the order of the logging in a method should be as follows:

1. Method entry
2. Log method entry

3. Log method input parameters if any
4. If error occurs, log it and skip to step 7
5. Log method exit
6. If not void, log method output value
7. Method exit

1.4 Component Class Overview

ProjectPaymentCalculator[Interface]

This interface is the main interface of this component, It is the contract of project payment calculation. It provides a single contract method which calculates for each resource role id the payments for a given project identified by its id.

BaseProjectPaymentCalculator[Abstract]

This is the base class of the payment calculators defined in this component. It can be extended by classes that need to gain access to the logger:Log instance to be used to perform logging, it provides createConnection() method to be used by subclasses to create a connection to the database. This class is configured using Configuration API ConfigurationObject. It holds a reference to Logging Wrapper Log instance that can be retrieved by subclasses via getLogger() getter and uses DBConnectionFactory from DB connection Factory component to create connections to the database.

DefaultProjectPaymentCalculator

This class is a concrete implementation of the ProjectPaymentCalculator interface. It provides a concrete implementation of the getDefaultPayments(projectId, resourceRoleIDs) method defined in the interface by getting parameters needed to compute the payments values from the database to which it connects using the connection created by the createConnection() method defined in the parent class. This class uses the following linear formula to compute the payment for each resource role :

Payment = Fixed_Amount + (Base_Coefficient + Incremental_Coefficient * Number_Of_Submissions) * 1st_place_prize;

The parameters values needed to compute the payment are retrieved from the database. This class additionally defines the getPayment(projectId, resourceRoleId, prize, submissionsCount) which computes the payment for the specified resource role ID, prize and number of submissions. It uses the Log instance defined in the base class to perform logging. this class supports only the following resource roles : Primary screener, reviewer, accuracy reviewer, failure reviewer, stress reviewer, aggregator, final reviewer, specification reviewer, milestone screener, milestone reviewer and copilot.

ProjectPaymentAdjustmentCalculator

This class acts as a project payment adjuster which wraps/decorates the ProjectPaymentCalculator implementation instance. It extends BaseProjectPaymentCalculator to gain access to the logger to be used to perform logging, and to create the connection to be used to connect to the database using createConnection(). It is a decorating implementation of the ProjectPaymentCalculator interface. It holds a reference to the ProjectPaymentCalculator implementation instance to which it delegates the calls. After delegating the call to the underlying implementation it will optionally scale the resulting payment values by a multiplier or will replace them with a fixed amount. The scaling multiplier and fixed amount values depend on the project id and resource role and will be read from the database.

1.5 Component Exception Definitions

ProjectPaymentCalculatorException

This exception extends BaseCriticalException. It is thrown by implementations of ProjectPaymentCalculator when some error occurred during processing.

ProjectPaymentCalculatorConfigurationException

This exception extends BaseRuntimeException. It is thrown by BaseProjectPaymentCalculator and its subclasses when there is some error during configuration (configuration file can not be opened, required configuration parameter not found or has invalid value).

1.6 Thread Safety

This component is thread safe.

Implementations of ProjectPaymentCalculator interface are required to be thread safe.

BaseProjectPaymentCalculator is thread safe since all its fields are initialized in the constructor and never changed after that. implementations of the aggregated DBConnectionFactory and Log are expected to thread safe.

DefaultProjectPaymentCalculator is thread safe since its base class is thread safe and it adds no additional mutable state which can affect the thread safety.

ProjectPaymentAdjustmentCalculator is thread safe since its base class is thread safe and all its fields are initialized in the constructor and never changed after that and the aggregated ProjectPaymentCalculator implementation instance is expected to be thread safe.

This makes this component thread safe.

2. Environment Requirements

2.1 Environment

Development language: Java 1.8
Compile target : Java 1.8
QA Environment : Java 1.8, RedHat Linux 4, Windows 2000, Windows 2003.

2.2 TopCoder Software Components

Base Exception 2.0.0 – defines BaseCriticalException, BaseRuntimeException and ExceptionData used in this component.

Configuration API 1.1.0 – defines ConfigurationObject used in this component for intializing classes that need configuration.

Configuration Persistence 1.0.2 – defines ConfigurationFileManager used by this component for reading configuration data from file.

DB Connection Factory 1.1.1 – defines DBConnectionFactory used by this component to create connections to the database..

Logging Wrapper 2.0.0 – defines Log, LogManager and Level classes used by this component to perform logging.

Object Factory 2.2.0 - is used for creating pluggable object instances.

Object Factory Configuration API Plugin 1.1.0 – allows to use configuration API for creating Object Factory.

TopCoder Commons Utility 1.0.0 – is used for checking parameters and logging.

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

None.

NOTE: The default location for 3rd party packages is ../lib relative to this component installation. Setting the ext_libdir property in topcoder_global.properties will overwrite this default location.

3. Installation and Configuration

3.1 Package Name

com.topcoder.management.payment.calculator
com.topcoder.management.payment.calculator.impl

3.2 Configuration Parameters

3.2.1 BaseProjectPaymentCalculator configuration parameters description

The following table describes the structure of the ConfigurationObject passed to the BaseProjectPaymentCalculator() constructor (angle brackets are used for identifying child configuration objects).

Parameter	Description	Values
logger_name	The name of the logger to be used to perform logging. When not provided, logging is not performed.	String. Optional. When empty, it is treated as it is not provided.
connection_name	The database connection name to be used by this class. When not specified the default connection is used.	String. Optional. When empty, it is treated as it is not specified.
< db_connection_factory_config>	The configuration for DBConnectionFactoryImpl instance to be used by this class.	ConfigurationObject. Not null. Required.

3.2.2 DefaultProjectPaymentCalculator configuration parameters description

DefaultProjectPaymentCalculator can be configured using XML files or properties files (see the demo for a sample XML configuration file and refer to Configuration Persistence component for more information) or directly via a ConfigurationObject. Since this class does not any extra configuration parameters, then the configuration file or the ConfigurationObject to be used to configure them should have the same structure as shown in section 3.2.1.

3.2.3 ProjectPaymentAdjustmentCalculator configuration parameters description

ProjectPaymentAdjustmentCalculator can be configured using XML files or properties files (please refer to the demo for a sample configuration file).
In addition of the configuration parameters described in the section 3.2.1, the ProjectPaymentAdjustmentCalculator needs the configuration parameters defined in the following table :

Parameter	Description	Values
<object_factory_config>	This section contains configuration of Object Factory used by this class for creating pluggable object instances. It is required to provide this configuration parameter when the caller will use any one of the constructors that do not accept a ProjectPaymentCalculator input parameter.	ConfigurationObject. Required.
project_payment_calculator_key	The Object Factory key that is used for creating an instance of ProjectPaymentCalculator underlying implementation to be used by this class for payments calculation. It is required to provide this configuration parameter when the caller will use any one of the constructors that do not accept a ProjectPaymentCalculator input parameter.	String, Not Empty. Required.

3.3 Dependencies Configuration

Please refer to the dependencies documentation to configure them properly.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Edit file build-dependencies.xml to configure the dependencies
- Execute 'test_files/create.sql' to create the necessary tables in database.
- Edit the database configuration (url, username, password) in 'test_files/SampleConfig.xml' to match to your environment
- Execute 'ant test' within the directory that the distribution was extracted to.
- Optionally, 'test_files/drop.sql' can be used to remove the created tables

4.2 Required steps to use the component

Please see the demo.

4.3 Demo

4.3.1 Sample configuration

This following XML files show sample configuration. (Please note that the configuration can done using XML files or properties, for more details about the configuration files format please refer to the Configuration Persistence component documentation).

File **SampleConfig.xml**

```
<?xml version="1.0"?>
<CMConfig>
    <!-- Default project payment calculator -->
       <Config

       name="com.topcoder.management.payment.calculator.impl.DefaultProjectPayme
ntCalculator">
              <Property name="logger_name">
                     <Value>my_logger</Value>
              </Property>
              <Property name="connection_name">
                     <Value>my_connection</Value>
              </Property>
```

```
            <!-- configuration for DBConnectionFactoryImpl -->
            <Property name="db_connection_factory_config">
                  <Property
name="com.topcoder.db.connectionfactory.DBConnectionFactoryImpl">
                        <Property name="connections">
                              <Property name="default">
                                    <Value>my_connection</Value>
                              </Property>

                              <Property name="my_connection">
                                    <Property name="producer">

      <Value>com.topcoder.db.connectionfactory.producers.JDBCConnectionProducer
</Value>
                                    </Property>
                                    <Property name="parameters">
                                          <Property name="jdbc_driver">

      <Value>com.informix.jdbc.IfxDriver</Value>
                                          </Property>
                                          <Property name="jdbc_url">
                                                <Value>jdbc:informix-
sqli://localhost:2021/tcs_catalog:informixserver=informixoltp_tcp</Value>
                                          </Property>
                                          <Property name="user">
                                                <Value>informix</Value>
                                          </Property>
                                          <Property name="password">
                                                <Value>1nf0rm1x</Value>
                                          </Property>
                                    </Property>
                              </Property>
                        </Property>
                  </Property>
            </Property>
      </Config>

      <!-- Project payment adjustment calculator -->
      <Config

      name="com.topcoder.management.payment.calculator.impl.ProjectPaymentAdjus
tmentCalculator">
            <!-- configuration for ObjectFactory -->
            <Property name="object_factory_config">
                  <Property name="DefaultProjectPaymentCalculator">
                        <Property name="type">

      <Value>com.topcoder.management.payment.calculator.impl.DefaultProjectPaym
entCalculator</Value>
                        </Property>
                  </Property>
            </Property>
            <Property name="project_payment_calculator_key">
                  <Value>DefaultProjectPaymentCalculator</Value>
            </Property>
            <Property name="logger_name">
                  <Value>adjustment_calculator_logger</Value>
            </Property>
            <Property name="connection_name">
                  <Value>my_connection</Value>
            </Property>
            <!-- configuration for DBConnectionFactoryImpl -->
            <Property name="db_connection_factory_config">
```

```
                        <Property
name="com.topcoder.db.connectionfactory.DBConnectionFactoryImpl">
                              <Property name="connections">
                                    <Property name="default">
                                          <Value>my_connection</Value>
                                    </Property>

                                    <Property name="my_connection">
                                          <Property name="producer">

      <Value>com.topcoder.db.connectionfactory.producers.JDBCConnectionProducer
</Value>
                                          </Property>
                                          <Property name="parameters">
                                                <Property name="jdbc_driver">

      <Value>com.informix.jdbc.IfxDriver</Value>
                                                </Property>
                                                <Property name="jdbc_url">
                                                      <Value>jdbc:informix-
sqli://localhost:2021/tcs_catalog:informixserver=informixoltp_tcp </Value>
                                                </Property>
                                                <Property name="user">
                                                      <Value>informix</Value>
                                                </Property>
                                                <Property name="password">
                                                      <Value>1nf0rm1x</Value>
                                                </Property>
                                          </Property>
                                    </Property>
                              </Property>
                        </Property>
                  </Property>
      </Config>
</CMConfig>
```

The configuration file above is loaded into <code>ConfigurationObject</code> instance via Configuration Persistence component by providing a configuration properties file as following:

File **config.properties**

```
com.topcoder.management.payment.calculator.impl.DefaultProjectPaymentCalculator=
SampleConfig.xml
com.topcoder.management.payment.calculator.impl.ProjectPaymentAdjustmentCalculat
or=SampleConfig.xml
```

It contains the mappings between namespace and the XML configuration file.

The configuration then can be loaded using code like this

```
BaseProjectPaymentCalculator.getConfigurationObject("test_files/config.propertie
s",
"com.topcoder.management.payment.calculator.impl.DefaultProjectPaymentCalculator
");
```

4.3.2 *Data setup*

The following tables describe the data setup to be used for demo purpose. (only relevant columns are shown for brevity)

project_category_id	resource_role_id	fixed_amount	base_coefficient	incremental_coefficient
1	2	00.0	0.00	0.01
1	4	00.0	0.12	0.05
1	8	10.0	0.00	0.00
2	9	00.0	0.05	0.00

default_project_payment

Project_id	Project_category_id
230	1
231	2
232	3
233	4

Project

Project_id	Prize_type_id	place	Prize_amount
230	15	1	500
232	15	1	650
233	15	1	1000

prize

Submission_type_id	Upload_id	Submission_status_id
1	500	1
1	623	2
3	46	7

submission

Project_id	Upload_type_id	Upload_id
230	1	500
230	1	623
233	1	46

upload

Project_id	Resource_role_id	Fixed_amount	multiplier
230	2	14	NULL
230	4	22	NULL
230	5	NULL	3.0
230	8	NULL	2.0
230	9	NULL	NULL

project_payment_adjustment

4.3.3 API demo

```
// create an instance of DefaultProjectPaymentCalculator using the default
configuration file
// found in
com/topcoder/management/payment/calculator/impl/DefaultProjectPaymentCalculator.
properties
ProjectPaymentCalculator calculator = new DefaultProjectPaymentCalculator();

// create an instance of DefaultProjectPaymentCalculator using the provided
configuration file
String configFile = "test_files" + File.separator + "config.properties";
String namespace =
"com.topcoder.management.payment.calculator.impl.DefaultProjectPaymentCalculator
";

calculator = new DefaultProjectPaymentCalculator(configFile, namespace);

// Get the default payments for the primary screener and reviewer for the
project identified
// by projectId = 230
List<Long> resourceRoleIDs = new ArrayList<>();

resourceRoleIDs.add(DefaultProjectPaymentCalculator.PRIMARY_SCREENER_RESOURCE_RO
LE_ID);

resourceRoleIDs.add(DefaultProjectPaymentCalculator.REVIEWER_RESOURCE_ROLE_ID);

Map<Long, BigDecimal> result = calculator.getDefaultPayments(230,
resourceRoleIDs);
System.out.println("Default payment: ");
result.forEach((roleId, payment)->System.out.println("Payment for role " +
roleId + " = " + payment));


// The returned Map will contain two entries :
// The first element : key = 2, BigDecimal value = 10 = 0 + (0.0 + 0.01*2)*500
// This project has two submissions :
// (submission.upload_id = 500 , and submission.upload_id = 623)

// The second element : key = 4, BigDecimal value = 85 = 0 + (0.12 + 0.05*1)*500
// This project has a single submission that passed screening.
// (submission.submission_status_id != 2)

// Get the default payment for a Final Reviewer for a project with prize =
$ 1000.00 having 3
// submissions and project category id = 2
BigDecimal finalReviewerPayment =
    ((DefaultProjectPaymentCalculator) calculator).getDefaultPayment(2,
                DefaultProjectPaymentCalculator.FINAL_REVIEWER_RESOURCE_ROLE_ID,
new BigDecimal(1000), 3);

System.out.println("Payment for Final Reviewer role = " + finalReviewerPayment);

// The value of the finalReviewerPayment will be 50 = 0 + (0.05 + 0.00*0)*1000
// in the above calculation the number of submissions is equal to zero, because
it has no meaning
// for a final reviewer.

// create an instance of DefaultProjectPaymentCalculator using the default
configuration file
// found in
com/topcoder/management/payment/calculator/impl/ProjectPaymentAdjustmentCalculat
or.properties
```

```
ProjectPaymentCalculator adjuster = new ProjectPaymentAdjustmentCalculator();

// create a new ProjectPaymentAdjustmentCalculator with to wrap the calculator
created above using the same
// configuration file.
String namespace1 =
"com.topcoder.management.payment.calculator.impl.ProjectPaymentAdjustmentCalcula
tor";

adjuster = new ProjectPaymentAdjustmentCalculator(calculator, configFile,
namespace1);

// Get the adjusted default payments for primary screener, reviewer and
aggregator for the
// project with project_id = 230.

resourceRoleIDs.add(DefaultProjectPaymentCalculator.AGGREGATOR_RESOURCE_ROLE_ID);

// the other two role ids were already in the list (see above).
Map<Long, BigDecimal> adjustedPayments = adjuster.getDefaultPayments(230,
resourceRoleIDs);

// The adjustedPayments map will contain the following 3 elements
// first element : key = 2, value = 14 (adjusted to the value of fixed_amount)
// second element : key = 4, value = 22 (adjusted to the value of fixed_amount).
// third element : key = 8, value = (10 + (0.00 + 0.00*0)*500 ) * 2 = 20
(multiplier == 2)
adjustedPayments.forEach((roleId, adjustedPayment)->System.out.println("Adjusted
Payment for role " + roleId + " = " + adjustedPayment));
```

5. Future Enhancements

None at this time.