Software Documentation : Java Custom Project Payment Calculator

This page last changed on Jan 26, 2013 by volodymyrk.

1. Scope

1.1 Overview

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

1.2 Logic Requirements

1.2.1 Project Payment Calculator interface

The component will provide an interface for calculating default payments for resources within a project. The interface will contain one method:

```
Map<Long, BigDecimal> getDefaultPayments(long projectId, List<Long> resourceRoleIDs)
```

The method takes a project ID and a list of resource role IDs and returns a map from resource role IDs to default payment values. The returned map doesn't necessarily have to include all requested resource roles, in which case the default payment for the missed roles is assumed to be unknown. Use BigDecimal (with scale 2 and rounding mode HALF_UP) to represent monetary values.

1.2.2 Linear Project Payment Calculator

The component will provide an implementation of the Project Payment Calculator interface which will calculate all payment values using the linear function:

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

Fixed_Amount, Base_Coefficient and Incremental_Coefficient are constant values which depend on the project category and resource role. These values will be read from the tcs_catalog.default_project_payment table. If the table doesn't contain values for some project category and resource role, the payment is assumed to be unknown and should not be included in the resulting map. You will need to join the tcs_catalog.project table to get the project category ID by the project ID.

Number_Of_Submissions has different meaning depending on the resource role:

1. For Primary Screener role: the total number of active contests submissions. If the number is zero, assume one submission.
2. For Reviewer, Accuracy Reviewer, Failure Reviewer and Stress Reviewer roles: the total number of active contests submissions that passed screening. If the number is zero, assume one submission.
3. For Milestone Screener role: the total number of active milestone submissions. If the number is zero, assume one submission.
4. For Milestone Reviewer role: the total number of active milestone submissions that passed milestone screening. If the number is zero, assume one submission.
5. For other resource roles this value doesn't have any meaning so should be assumed 0.

1st_place_prize is the prize for the first place winner of the contest.

The calculator implementation will support the following resource roles:

1. Primary Screener (resource role ID 2).
2. Reviewer, Accuracy Reviewer, Failure Reviewer, Stress Reviewer (resource role IDs are 4,5,6,7).
3. Aggregator (resource role 8).
4. Final Reviewer (resource role 9).
5. Specification Reviewer (resource role ID 18).
6. Milestone Screener (resource role ID 19).
7. Milestone Reviewer (resource role ID 20).
8. Copilot (resource role ID 14).

You can use this SQL to get all necessary data out of the DB (remember to set the project_id SQL parameter). If the result set doesn't contain any of the requested resource roles then it should not be included in the resulting map either.

```
SELECT dpp.resource_role_id, dpp.fixed_amount, dpp.base_coefficient, dpp.incremental_coefficient,
max(pr.prize_amount) as prize,
sum(case when s.submission_type_id = 1 then 1 else 0 end) as total_contest_submissions,
sum(case when s.submission_type_id = 1 and s.submission_status_id != 2 then 1 else 0 end) as
passed_contest_submissions,
sum(case when s.submission_type_id = 3 then 1 else 0 end) as total_milestone_submissions,
sum(case when s.submission_type_id = 3 and s.submission_status_id != 6 then 1 else 0 end) as
passed_milestone_submissions

FROM default_project_payment dpp
INNER JOIN project p ON dpp.project_category_id = p.project_category_id and p.project_id=?
LEFT OUTER JOIN prize pr ON pr.project_id=p.project_id and prize_type_id=15 and place=1
LEFT OUTER JOIN upload u ON u.project_id = p.project_id and u.upload_type_id = 1
LEFT OUTER JOIN submission s ON s.submission_type_id in (1,3) and s.upload_id = u.upload_id and
s.submission_status_id in (1,2,3,4,6,7)
WHERE dpp.resource_role_id in (2,4,5,6,7,8,9,14,18,19,20)
GROUP BY dpp.resource_role_id, dpp.fixed_amount, dpp.base_coefficient, dpp.incremental_coefficient
```

This implementation will also provide another getDefaultPayments method (which is not from the interface) that will let application compute expected payments even if the project ID id now known yet:

```
BigDecimal getDefaultPayments(long projectCategoryId, long resourceRoleId, BigDecimal prize, int
submissionsCount)
```

The method will basically perform the same as above except that the prize and submissions count are not fetched from the DB but are explicitly specified. It also computes payment for a single resource role and not for a list of resource roles.
You can use this SQL to get all necessary data out of the DB (remember to set the project_category_id and resource_role_id SQL parameters). If the result set doesn't contain the requested resource role, return null.

```
SELECT fixed_amount, base_coefficient, incremental_coefficient
FROM default_project_payment
WHERE project_category_id = ? and resource_role_id = ?
```

1.2.3 Adjusting Project Payment Calculator

The component will provide a decorating (see the Decorator pattern) implementation of the Project Payment Calculator interface which will keep a reference to another Project Payment Calculator implementation which the calls will be redirected to. The calculator implementation 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 DB.

The calculator implementation will perform as follows:

1. Get the multiplier and fixed amount values for the given project ID and all resource roles from the DB by running this SQL (remember to set the project_id SQL parameter). If the result set doesn't contain any of the requested resource roles then the fixed_amount and multiplier values are assumed to be null (i.e. unknown) for that resource role.

   ```
   SELECT resource_role_id, fixed_amount, multiplier
   FROM project_payment_adjustment
   WHERE project_id=?
   ```
2. Call the underlying calculator implementation.
3. For each requested resource role, if the corresponding fixed_amount for that role is not null then return the fixed_amount. If the fixed_amount is null and multiplier is not null, multiply the payment value by the multiplier. Otherwise just return the payment value directly.

1.2.4 Thread-Safety

The component is required to be thread-safe.

1.3 Required Algorithms

None.

1.4 Example of the Software Usage

The component will be used by the Online Review application to manage the project payment and payment adjustment entries.

1.5 Future Component Direction

Any enhancement needs to be approved either in forum or in email with managers to eliminate over-complicating the component with useless functions. All performance optimizations are highly encouraged and do not require explicit approval.

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None, only API interface will be provided.

2.1.2 External Interfaces

None.

2.1.3 Environment Requirements

- Development language: Java 1.5
- Compile target: Java 1.5, Java 1.6

2.1.4 Package Structure

com.topcoder.management.payment.calculator

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

None.

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

None.

3.2.2 TopCoder Software Component Dependencies:

- Base Exception 2.0.0
- Configuration API 1.1.0

- Configuration Persistence 1.0.2
- DB Connection Factory 1.1.1
- Logging Wrapper 2.0.0

**Please review the TopCoder Software component catalog for existing components that can be used in the design.

3.2.3 Third Party Component, Library, or Product Dependencies:

Any third party library needs to be approved.

3.2.4 QA Environment:

- Java 1.5
- RedHat Linux 4
- Windows 2000
- Windows 2003

3.3 Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines. Modifications to these guidelines for this component should be detailed below.

3.4 Required Documentation

3.4.1 Design Documentation

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification

3.4.2 Help / User Documentation

- Design documents must clearly define intended component usage in the 'Documentation' tab of TC UML Tool.