



Rollbase

Platform as a Service (PaaS)

Rollbase Formula Guide

Last Update: October 15th 2008

| | |
|--|----------|
| INTRODUCTION | 6 |
| CREATING FORMULA EXPRESSIONS | 6 |
| FORMULA FIELDS | 6 |
| FORMULAS IN WORKFLOW EVENTS | 6 |
| JAVASCRIPT..... | 7 |
| <i>Limitations</i> | 7 |
| ERROR PROCESSING..... | 7 |
| FORMULA RETURN TYPE | 8 |
| WORKING WITH DATES IN FORMULAS | 8 |
| WORKING WITH IMAGE AND SHARED IMAGE FIELDS IN FORMULAS..... | 9 |
| DEBUGGING FORMULAS | 9 |
| INTEGRATION NAMES & MERGE FIELDS | 9 |
| USING MERGE FIELDS..... | 10 |
| NESTED FORMULAS..... | 10 |
| KEEP FORMULAS SIMPLE | 10 |
| TEMPLATES VERSUS FORMULAS | 10 |
| GROUP FUNCTIONS IN FORMULAS..... | 11 |
| SUM..... | 11 |
| COUNT..... | 11 |
| MAX..... | 12 |
| AND | 12 |
| OR..... | 12 |
| LOOPING THROUGH RELATED RECORDS..... | 12 |
| <i>Limiting the Number of Iterations</i> | 13 |
| <i>Looping through a Particular View</i> | 13 |
| FORMULA EXAMPLES, PART I: ACCOUNT MANAGEMENT | 14 |
| Rating..... | 14 |
| <i>Contract Aging This formula calculates the age of a contract with an account as the number of days since the contract has been activated. The return type for this formula is an Integer.....</i> | 14 |
| <i>Contract Approval Process Aging</i> | 14 |
| <i>Month of Last Account Activity</i> | 15 |
| <i>Month of Service-Level Agreement Expiration</i> | 15 |
| FORMULA EXAMPLES, PART II: CASE MANAGEMENT | 15 |
| Case Aging (Open Cases)..... | 15 |
| Case Aging (Open and Closed Cases)..... | 16 |
| Case Aging (Assignments) as Text | 16 |
| Case Categorization..... | 17 |
| Case Data Completeness Tracking..... | 17 |
| Case Due Date Calculation | 17 |
| Flags for Case Priority..... | 18 |
| Color Squares for Case Age..... | 18 |
| FORMULA EXAMPLES, PART III: COMMISSION CALCULATIONS..... | 18 |
| Commission Amounts for Opportunities..... | 18 |
| Commission Deal Size..... | 19 |
| Commission Greater Than or Equal To | 19 |
| Commission Maximum..... | 19 |
| FORMULA EXAMPLES, PART IV: CONTACT MANAGEMENT..... | 19 |
| Contact's Age..... | 19 |
| Contact's Birthday in Current Month?..... | 20 |
| Contact Identification Numbering..... | 20 |

| | |
|--|----|
| <i>Contact Preferred Phone</i> | 20 |
| <i>Dynamic Address Formatting</i> | 20 |
| <i>Telephone Country Code</i> | 21 |
| <i>Unformatted Phone Numbers</i> | 21 |
| FORMULA EXAMPLES, PART V: DATA CATEGORIZATION..... | 21 |
| <i>Deal Size Large and Small</i> | 22 |
| <i>Product Categorization</i> | 22 |
| FORMULA EXAMPLES, PART VI: DATE FORMULAS..... | 22 |
| <i>Today's Day of Week(Number)</i> | 22 |
| <i>Today's Day of Week (Text)</i> | 22 |
| <i>Day of Week</i> | 22 |
| <i>Current Time</i> | 23 |
| <i>Current Time (formatted for 12-hour display)</i> | 23 |
| <i>Days until a target date</i> | 23 |
| <i>Days until end of Month</i> | 23 |
| <i>Fiscal Year Calculation:</i> | 24 |
| <i>Quarter Calculation This formula is used to identify which Quarter the campaign begins in using the campaign start date. Lets say the start date of the campaign is given by the Date field "Start Date" whose merge token is given "{Istart_date}" and the campaign quarters are broken out as follows: Q1: July - September Q2: October - December Q3: January - March Q4: April – June The formula is now calculated as,</i> | 24 |
| FORMULA EXAMPLES, PART VII: DISCOUNTING | 24 |
| <i>Maintenance and Services Discount</i> | 24 |
| <i>Opportunity Discount Amount</i> | 25 |
| FORMULA EXAMPLES, PART VIII: EMPLOYEE MANAGEMENT | 25 |
| <i>Total Hours Worked Per Week</i> | 25 |
| <i>Employee Weekly Pay</i> | 25 |
| <i>Bonus Calculations</i> | 25 |
| FORMULA EXAMPLES, PART IX: EXPENSE TRACKING | 26 |
| <i>Mileage Calculation</i> | 26 |
| FORMULA EXAMPLES, PART X: FINANCIAL CALCULATIONS..... | 26 |
| <i>Compound Interest</i> | 26 |
| <i>Compound Interest Continuous</i> | 26 |
| <i>Gross Margin</i> | 26 |
| <i>Gross Margin Based on Margin Percent</i> | 27 |
| <i>Payment Due Indicator</i> | 27 |
| <i>Payment Status</i> | 27 |
| <i>Shipment Tracking Integration Link</i> | 27 |
| FORMULA EXAMPLES, PART XI: LEAD MANAGEMENT | 28 |
| <i>Lead Aging (for open leads)</i> | 28 |
| <i>Lead Data Completeness</i> | 28 |
| <i>Lead Numbering</i> | 29 |
| <i>Round Robin Assignment of Cases or Leads</i> | 29 |
| FORMULA EXAMPLES, PART XII: OPPORTUNITY MANAGEMENT | 29 |
| <i>Days Left to Close</i> | 29 |
| <i>Display Close Month for Reporting Purposes</i> | 29 |
| <i>Expected Product Revenue</i> | 30 |
| <i>Maintenance Fee</i> | 30 |
| <i>Monthly Subscription-Based Calculated Amounts</i> | 30 |
| <i>Monthly Value</i> | 30 |
| <i>Opportunity Additional Costs</i> | 30 |
| <i>Opportunity Categorization</i> | 31 |
| <i>Opportunity Data Completeness</i> | 31 |
| <i>Opportunity Expected License Revenue</i> | 31 |
| <i>Opportunity Reminder Date</i> | 32 |

| | |
|--|----|
| <i>Opportunity Split Credit for Sales Representatives</i> | 32 |
| <i>Opportunity Total Deal Size</i> | 32 |
| <i>Professional Services Calculation</i> | 32 |
| <i>Shipping Cost by Weight</i> | 32 |
| <i>Shipping Cost Percentage</i> | 33 |
| <i>Commission</i> | 33 |
| <i>Total Contract Value from Recurring and Non-Recurring Revenue</i> | 33 |
| FORMULA EXAMPLES, PART XIII: PRICING | 33 |
| <i>Total Amount</i> | 33 |
| <i>User Pricing</i> | 34 |
| FORMULA EXAMPLES, PART XIV: PROJECT MANAGEMENT..... | 34 |
| <i>Task Due Date</i> | 34 |
| <i>Days Overdue</i> | 34 |
| <i>Number of Project Tasks</i> | 34 |
| <i>Project Completion Status as a Percentage</i> | 35 |
| <i>Project Progress Bar Formula</i> | 35 |
| <i>Traffic Lights to Represent Project Status</i> | 36 |
| FORMULA EXAMPLES, PART XV: SCORING & RATING | 37 |
| <i>Lead Scoring</i> | 37 |
| <i>Star Ratings</i> | 37 |
| <i>Horizontal Bars to Indicate Scoring</i> | 37 |
| FORMULA EXAMPLES, PART XVI: MISCELLANEOUS..... | 38 |
| <i>Celsius to Fahrenheit</i> | 38 |
| <i>Miles to Kilometers</i> | 38 |
| <i>Kilograms to Pounds</i> | 38 |

Introduction

Rollbase allows the use of formula expressions to perform calculations on the fly using field values a record in scope as well as any of its related records. Formula expressions can be used to define Formula fields associated with object definitions, as well as conditions on Workflow Events to determine whether a given event should be executed. They can also be used to define the return result of a specific type of workflow event which is designed to update the value of a field.

Creating Formula Expressions

Formula expressions are defined using JavaScript and are executed on the server side. Fields from a record in scope as well as related records can be accessed in formula expressions using merge fields where the value of the referenced field should be used in the expression. The only JavaScript functionality that cannot be used in formulas is looping using `for` and `while`, or defining new functions and arrays. These particular features are disabled to ensure adequate server-side performance. As we will discuss later in this document, formula expressions provide an alternative way to perform looping.

When creating a formula expression, merge fields can be used to extract data from record and related records. Merge fields are also used in templates (see Rollbase Templates Guide) and have the following format: `{!field_name}`. Merge fields can be placed anywhere in formula code and are replaced by the actual field values when the formula is executed. The following illustrates a `{!city}` merge field in a simple formula which returns a different value depending on whether a record's value for the city field is equal to "New York" or not:

```
var rate = 6.5;
if (" {!city}" == "New York")
    rate = 8.5;
return rate;
```

If this formula is executed for a particular object record whose city field has a value of "San Francisco", at runtime the formula will be translated to the following code before Rollbase executes it on the server-side:

```
var rate = 6.5;
if ("San Francisco" == "New York")
    rate = 8.5;
return rate;
```

In this case the result of the formula expression will be the number 6.5

Formula Fields

Formula fields are a powerful way to create dynamically generated fields associated with any of your objects. For example, formula fields can present the average or sum of a number of other fields. Formula field values are not stored in the database, rather they are calculated on the fly each time the field is displayed in the Rollbase user interface.

Formulas in Workflow Events

For objects with the "Workflow" attribute, formula expressions can be used to define workflow event conditions which return true or false to determine whether the associated event should be executed when the trigger is activated and all conditions are met. Formula expressions are also used to change the value of a specific field when an event of type "Change Field Value" is used. In this case the formula expression returns the new value to use in the target field.

JavaScript

Because formula expressions are defined using JavaScript, you have a tremendous amount of flexibility. In order to fully take advantage of this capability a familiarity with JavaScript is recommended but not entirely necessary. This Formula guide includes many example formulas for you to reference, most of which can be understood by non-JavaScript developers.

Limitations

Given the broad capabilities of JavaScript at your disposal, it is possible to perform complex calculations and create powerful formula expressions. However, to ensure proper performance for all customers on the Rollbase platform, a few limitations have been imposed:

- The length of formula expression cannot exceed 1000 characters
- Formula expressions cannot include loops using JavaScript keywords `for` and `while` (looping is allowed using other techniques; see *Looping Through Related Records* below)
- Formulas cannot include declarations of functions or arrays

Despite these imposed limitations, formulas in Rollbase are quite powerful. For example, you can include:

- Logical operations such as `if-then-else` and `switch`
- Standard JavaScript objects and their methods such as `String`, `Date`, `Math`, etc
- Standard functions available in any JavaScript capable environment
- Merge fields from both base record and all related records
- Special group functions designed for easy looping through related objects (see *Group Functions in Formulas* below)
- Special loop merge fields to iterate through related objects

Error Processing

When defining a formula expression in Rollbase, you will see a "Check for errors" link below the formula text area. Use this link to check whether the formula is valid before saving your changes. NOTE: This check will only work if at least one record of the given object type already exists. Valid formulas are also no guarantee that errors will not occur at runtime, for example division by zero cannot always be predicted. In the event of a runtime error a formula will be evaluated to "ERROR".

Rating: Edit Field Save Cancel

Field Properties Red = Required Information


Field properties are global settings that apply to this field wherever it is presented for input or display.

| | | | |
|----------------------------|------------------|---|--|
| Field Type | Formula (String) | | |
| Field Label | Starred Image | | |
| Formula Return Type | String | \$###,###.## | |
| Decimal Places | 0 | Determines number of digits to display to the right of the decimal point. | |

Available Merge Fields

| | | | |
|-------------------|--------------|------------------------|--------------------|
| Select Field Type | Select Field | Copy Merge Field Token | Select Field Value |
| Rating | | | |

Enter your formula and [check it for errors](#).

To debug your formula select a Rating: 

```
return "{!star1}";
else if ({!rating}==106470)
return "{!star2}";
else if ({!rating}==106471)
return "{!star3}";
else if ({!rating}==106472)
return "{!star4}";
else if ({!rating}==106473)
return "{!star5}";
else
return '';
```

Formula is valid

Formula Return Type

Formula fields support the following return types:

- Decimal
- Currency
- Integer
- String
- Boolean
- Date

If a formula field returns a value which does not match the specified return type this value will be ignored.

Example of a simple numeric formula expression:

```
"{!city}".length
```

Example of simple String formula expression:

```
"{!city}".toUpperCase()
```

Example of a simple Boolean formula expression:

```
"{!city}" == "Houston"
```

Example of a simple Date formula which adds 7 days to the date represented by the `createdAt` field:

```
(new Date("{!createdAt}")).getTime()+7*24*60*60*1000
```

Working with Dates in Formulas

Due to limitations imposed by server-side JavaScript please be aware of the following when working with Dates in

formula expressions:

- The only way to create a new `Date` instance in a formula is to use the `Date(String)` constructor as shown in the example above.
- When returning a `Date` value from a formula, the expression must return the results of a `Date` instance's `getTime()` method.

Working with Image and Shared Image Fields in Formulas

Each image or shared image field has two merge fields associated with it:


- `{!fieldname#html}` Use the image html merge field to include the full html code for an image in your formula. This merge field will be replaced with a full html IMG tag.

IMPORTANT NOTE: When working with image html merge fields in formulas you must wrap them in quotes to avoid errors. For example: `return "{!greenImage#html}";` is the correct way to return an image to be displayed as a formula result.

- `{!fieldname#url}` Use the image url merge field to only include the url for an image in your formula. This merge field will be replaced with a full path to the image. This is particularly useful when you want to build custom IMG tags that include more information such as a unique identifier, JavaScript event handlers, etc.

Debugging Formulas

Once you have defined and validated your formula expression you can execute the formula against a particular object record for debugging purposes by clicking the lookup icon to select a record as shown here:

To debug your formula select a Time Sheet: 

Once selected, a dialog will display showing the original formula, the parsed version of the formula which is executed on the server, and the final result:

| |
|--------------------------------|
| Original Formula |
| <code>{!hours_spent}*60</code> |
| Parsed Formula |
| <code>1.0*60</code> |
| Result |
| <code>60.0</code> |

Integration Names & Merge Fields

One common pattern you notice in Rollbase is a property called integration name for every object, field, and relationship. This integration name is used as a unique reference the component for various purposes. In particular, integration names are used in merge fields which can be used in formulas and templates.

For example when a field called "Account Name" is created, "Account Name" will be the display name for this field and by default the integration name will be set to "account_name". When we want to use this field in a formula expression, we can reference it using the merge field format as follows: `{!account_name}`.

Throughout this guide we will use merge fields in formula expressions to perform various calculations based on

the values of fields associated with an record or related records in scope.

Using Merge Fields

When defining formula expressions and templates, Rollbase provides a series of picklists to assist in the selection of appropriate merge fields:

Available Merge Fields
Select Field Type Select Field Copy Merge Field Token Select Field Value

- **Select Field Type** allows you to select the object or one of its related object to select merge fields from. Alternatively you can select from standard helper merge fields.
- **Select Field** allows you to select the desired field to insert into your formula as a merge field. Alternatively you can select from a number of predefined functions for working with related records, or you can select from looping placeholders allowing you to loop through a specific set of related records within your formula expression.
- **Copy Merge Field Field** displays the selected merge field, function or placeholder so you can easily copy and paste it into your formula expression as needed.
- **Select Field Value** is used to select a particular picklist value or related record ID for use in a formula expression.

The following example returns true if a User record has the Administrator role, where 90 is the ID of the Administrator role and `{!role}` is the role of the User in scope:

```
{!role}==90
```

Nested Formulas

It is possible to use formula fields within other formula expressions, creating nested formulas. However, in this scenario the order of execution cannot be guaranteed. In addition, the maximum level of recursion cannot exceed 10.

Keep Formulas Simple

Keeping your formulas as simple as possible will make your applications more maintainable while improving application performance. Some tips for simplifying formulas:

- Avoid using the `return` clause for one-line formulas without internal variables.
- Avoid unnecessary concatenation using `+`
- Use conditional operators `(X?A:B)` instead of `if()`

Templates Versus Formulas

Formulas and Templates both use merge fields to extract data from the record and related records in scope. The difference is that Templates only embed data into pre-formatted text, while Formulas invoke and underlying JavaScript engine to compute a results.

Formulas treat merge fields the same way Templates do. For instance, in order to concatenate two merge fields in a formula expression you can simply write:

```
"{!lastName}, {firstName}"
```

There is no need to use JavaScript concatenation such as:

```
"{!lastName}"+", "+ "{firstName}"
```

However, if you want to make this formula more intelligent and avoid an unnecessary comma if the firstName field is empty, you can use conditional operators as in the following example:

```
("{firstName}"==" " ? "{!lastName}" : "{!lastName}, {firstName}")
```

Group Functions in Formulas

Rollbase allows relationships to be defined between any number of objects and provides ways to perform simple and complex calculations among related objects in formula expressions and templates. The following Group functions are available for this purpose:

- SUM
- COUNT
- MAX
- MIN
- AND
- OR

SUM, MAX and MIN require a numerical parameter while AND and OR require a boolean parameter and COUNT does not take any parameters. Each of these functions iterates through a particular relationship between the record in scope and all of its related records.

SUM

Consider a Product object that is related to a Product Component object through a relationship defined by the integration name RB188912. The Product Component object has a price field with integration name price. Therefore, the sum of all product component prices associated with a given product record is given by the group function:

```
SUM.RB188912 ( {!RB188912.line_amount} )
```

Similarly we can calculate the number of related product component records for this product using the group function:

```
SUM.RB188912 (1)
```

We can combine multiple SUM calls to calculate the average amount as follows:

```
SUM.RB188912 ( {!RB188912.line_amount} ) / SUM.RB188912 (1)
```

COUNT

Count the number of related product component records for a product:

```
#FUNC_COUNT.RB188912 ( )
```

MAX

Calculate the maximum value of the price field across all related product component records:

```
#FUNC_MAX.RB188912 ( {!RB188912.price} )
```

AND

Return true if the value of the status field across all related product component records is 123456.

```
#FUNC_AND.RB188912 ( {!RB188912.status#id}==123456 )
```

This type of formula can be useful in modeling approval processes for example.

OR

Return true if a specific record is found among a group of related records:

```
#FUNC_OR.RB188912 ( {!RB188912.line_id}==1234567 )
```

Looping Through Related Records

An alternative to using group formulas to do computations on groups of related records is looping. Rollbase formulas offer a mechanism to loop through related records using specific placeholders. `LOOP_BEGIN` and `LOOP_END` placeholders are used to mark the beginning and end of a related records loop. For example:

```
var total = 0;
{!#LOOP_BEGIN. RB188912}
    if ("!city"=="New York")
        total += {!RB188912.state}
{!#LOOP_END. RB188912}
return total;
```

All code between the `LOOP_BEGIN` and `LOOP_END` placeholders will be executed for each related record. For example, the resulting formula when executed by Rollbase may look like this (assuming there are only two related records):

```
var total = 0;
if ("New York"=="New York")
    total += 10;
if ("Houston"== "New York")
    total += 9;
return total;
```

Syntax for loops.

The general syntax for using loops is,

```
{!#LOOP_BEGIN.rel_name#view_id(max_items)}
```

Where

- `rel_name` - Integration name of the relationship or the token "all". The `rel_name` is mandatory in the syntax. If the token "all" is used, the system will loop through all records defined by view that is specified next to the `rel_name`.

- view_id is original id of view used for looping. This is optional if relationship integration name is specified and mandatory otherwise. Make sure to use the original ID of the list views. Since the ID will be modified when the application is published or a published application is modified and updated. The original ID of the list Views can be found by clicking on the View name in object Definition. The viewView page that opens contains all information regarding the views and also includes the view's original ID.
- max_items (optional): max number of items in the loop, for example (10)

Explanation of each of these parameters are as follows.

Limiting the Number of Iterations

When using this technique to iterate through a group of related records, the number of iterations that the loop should make can also be controlled. For example:

```
var content = "<strong>5 Recent Product Prices</strong><br>";  
{!#LOOP_BEGIN.RB188912(5)}  
    "{!RB188912.price} <br>"  
{!#LOOP_END.RB188912}
```

In this example, all code between the LOOP_BEGIN and LOOP_END placeholders will be executed only 5 times. The code will loop through all of the related records that are defined by the relationship integration name RB188912.

Looping through a Particular View

When looping through related records, the related object you are looping through may have more than one View component. In this case you may want to select which View component to loop through. For example, if you want to display or compute information on the 10 most recent Leads, you can specify the appropriate View to loop through in your LOOP_BEGIN merge field as follows:

```
var content = "<strong>Recently Updated Leads</strong><br>";  
{!#LOOP_BEGIN.RB103771#80815(10)}  
    "{!RB103771.name} <br>"  
{!#LOOP_END.RB103771}
```

In this example it is assumed that 80815 is the ID of a View component for the Related Lead object defined by the relationship name RB103771 which is sorted by "Updated At".

Looping through the same object

In looping, you may want to loop through the records of the same object and not a related object. In this case, the LOOP_BEGIN Merge field should use the token "all" in place of the relationship integration name and should specify which view of this object you want to loop through. For example, if you want to display or compute information on the most 10 recent Leads, you can specify the appropriate View to loop through the LOOP_BEGIN merge field as follows:

```
var content = "<strong>Recently Updated Leads</strong><br>";  
{!#LOOP_BEGIN.all#80815(10)}  
    "{!RB103771.name} <br>"  
{!#LOOP_END.RB103771}
```

In this example it is assumed that 80815 is the ID of a View component for the Lead object which is sorted by "Updated At".

Formula Examples, Part I: Account Management

The following are typical account management formulas used in applications that manage account information.

Rating

This formula calculates the rating for an account as a String value of either “hot”, “warm” or “cold” depending on a number field values. The return type for this formula is a Sting.

In our sample account object, assume the “Annual Revenue” field given by the merge field `{!annual_revenue}` stores the annual revenue of an account and the account object has a picklist called ‘Billing Country’. The selected country for a particular account is given by the merge field `{!billing_country#value}` and the type of the account is tracked in a picklist called “Account Type”. Also assume the selected account type for a particular account is given by the merge field `"{!account_type#value}"`. Given this we can calculate the annual rating for that account as:

```
if({!annual_revenue} > 1000000 && ("{!country#value}" == "United States"))
{
    if("{!type#value}" == "Customer (Direct)")
        return "hot";
    else
        return ((("{!type#value}" == "Installation Partner") || ("{!type#value}"
== "Channel Partner")) ? "warm" : "cold");
}
else {
    return "cold";
}
```

Note: For picklist fields, three types of merge fields are available, `picklist#value`, `picklist#code`, `picklist#id`. Any of these can be used in checking the selected picklist value.

Contract Aging

This formula calculates the age of a contract with an account as the number of days since the contract has been activated. The return type for this formula is an Integer.

In our sample account object, assume the activation of a contract for a particular account is given by a date field called "Activation date" and the merge field for this field is `"{!activation_date}"`. The status of activation is tracked by a picklist field called "Activation Status" and the selected status value for an account is given by the merge field `"{!activation_status#value}"`. Given this the age of the contract can be calculated as follows:

```
var today = new Date();
var activation_date = new Date("{!activation_date}");
if("{!activation_status#value}" == "Activated")
    return (today - activation_date)/(24*60*60*1000);
```

Contract Approval Process Aging

This formula calculates the number of days a contract for an account is in an approval process. The return type for this formula is an Integer.

In our sample account object, assume the date a contract was submitted for approval is given by the field 'Approval date' and the merge field for this field is `"{!approval_date}"`. The approval status for the account is given by the picklist “Approval Status” and the selected status value for an account is given by the merge field `"{!account_status#value}"`. We can then calculate the number of days the contract has been in the approval

process as follows:

```
var today = new Date();
var approval_date = new Date("{!approval_date}");

if("{!approval_status#value}" == "Submitted for Approval")
    return parseInt((today.getTime() - approval_date.getTime()) / (24*60*60*1000));
```

Month of Last Account Activity

This formula displays the month of the last account activity. The return type for this formula is a String.

In our sample account object, the last date that a particular account record was updated is tracked as a System Information field called "UpdatedAt" and the merge field for this field is given by "{!updatedAt}". We can now display the month of last activity on this account as follows:

```
var last_activity_date = new Date("{!updatedAt}");
var months = ["January", "February", "March", "April", "May", "June", "July", "August",
"September", "October", "November", "December"];

return months[last_activity_date.getMonth()] + " " +
last_activity_date.getFullYear();
```

Month of Service-Level Agreement Expiration

This formula displays the month of a service level agreement expiration date. The return type for this formula is String.

In our sample account object, assume a service level agreement expiration date is tracked in a Date field called "SLA Expiration Date" and the merge field for this field is given as "{!sla_expiration_date}". We can display the month of expiration using the following formula:

```
var sla_expiration_date = new Date("{!sla_expiration_date}");
var months = ["January", "February", "March", "April", "May", "June", "July", "August",
"September", "October", "November", "December"];

return months[sla_expiration_date.getMonth()] + " " +
sla_expiration_date.getFullYear();
```

Formula Examples, Part II: Case Management

The following are formulas that might be used in case management applications to track support requests or help tickets. We assume here that all information regarding a case is tracked in an object called 'case'.

Case Aging (Open Cases)

This formula calculates the number of days a case is open. It returns 0 if the case is not open (i.e. if it has been closed). The return type for this formula is an Integer.

In our sample case object, assume the case status is tracked in a picklist field called "Case Status" and the selected status value of a case is given by the merge field "{!case_status#value}". The date the case was created is given by a System Information field called "CreatedAt" and the merge field for this field is "{!createdAt}". The case age formula can then be calculated as:

```
if( "{!case_status#value}" != "Closed" ) {
    var today = new Date();
```

```
var created_date = new Date("{!createdAt}");
return parseInt((today.getTime() - created_date.getTime()) / (24*60*60*1000));
}
else {
    return 0;
}
```

Case Aging (Open and Closed Cases)

This formula calculates the number of days a case is open or the number of days a case has been closed. The return type for this formula is an Integer.

In our sample case object, assume the case status is tracked in a picklist field called “Case Status” and the selected status value of a case is given by the merge field “{!case_status#value}”. The date the case was created is given by a System Information field called “CreatedAt” and the merge field for this field is “{!createdAt}”. Also assume the date a case was closed is given by a date field “Closed Date” whose merge field is “{!closed_date}”. We can then express the case age the formula as follows

```
var today = new Date();
var created_date = new Date("{!createdAt}");
var closed_date = new Date("{!closed_date}");
if( {!case_status#value} != "Closed" )
    return parseInt((today.getTime() - created_date.getTime()) / (24*60*60*1000));
else
    return parseInt((closed_date.getTime() -
created_date.getTime()) / (24*60*60*1000));
```

Case Aging (Assignments) as Text

This formula displays different text depending on the number of days a case has been open.

- "Reassign" for any case open three days
- "Assign Task" for any case open two days
- "Maintain" for all other cases.

The return type for this formula is a String.

In our sample case object, assume the case status is tracked in a picklist field called “Case Status” and the selected status of a case is given by the merge field “{!case_status#value}” and the number of days the case was open was calculated in a formula field called “Case Age” whose merge field is “{!case_age}”, the case assignment formula is calculated as

```
if( "{!case_status#value}" == "Open" ) {
    if({!case_age} == 3 )
        return "Reassign";
    else
        return ({!case_age} == 2) ? "AssignTask" : "Maintain";
}
```

Note: This formula can also be used as a workflow condition. For instance, assume that “Case Status” is a picklist field and “Assignment status” is also a picklist field. We can configure an “Update Field Value” type of workflow event using the “On Create” trigger, and set the field to be changed as “Assignment Status”. Then in the workflow condition we can use the following formula to change the Assignment Status as appropriate.

```
if( "{!case_status#value}" == "Open" ) {
    if({!case_age} == 3 )
        return 653412;
    else
        return ({!case_age} == 2) ? 653413 : 653414;
}
```

```
}
```

...where 653412, 653413, 653414 are the ID's for the picklist values "Reassign", "AssignTask" and "Maintain" respectively.

Case Categorization

This formula displays the text value of "RED," "YELLOW," or "GREEN," depending on the value of a case age field. The return type for this formula is a String.

In our sample case object, the number of days the case has been open is calculated in a formula field called "Case Age" whose merge field is "{!case_age}". Given this a case categorization formula can be expressed as follows:

```
if({!case_age} > 20)
    return "RED";
if({!case_age} >10)
    return "YELLOW";
else
    return "GREEN";
```

Case Data Completeness Tracking

This formula calculates the percentage of case data completeness by looking at the values of two fields: "Problem Number" and "Severity Number". If the fields are empty, the formula returns the value "0." The formula returns a value of "1" for each field that contains a value and multiplies this total by fifty to give you the percentage of fields that contain data. The return type for this formula is a String.

In our sample case object, assume the field "Problem Number" whose merge field "{!problem_num}" stores the problem number and the field "Severity Number" whose merge field given by "{!severity_num}" stores the severity number for the case. Given this we can express data completeness percentage as:

```
var result =0;
if("{!problem_num}"== null || "{!problem_num}" == "")
    result =1;
if("{!severity_num}" == null || "{!severity_num}" == "")
    result +=1;
result *= 50;
return result+ "%";
```

Case Due Date Calculation

This formula returns the due date of a case based on its' priority. If it is `high`, the due date is two days after the case was created. If it is `medium`, the due date is five days after the case was created. Otherwise, the due date is seven days after the case was created. This formula returns a Date

In our sample case object, assume the date the case was created is given by System Information field "Created At" whose merge field is "{!createdAt}" and the case priority is given by the picklist field called "Case Priority". The selected value for a particular case will then be given by the merge field "{!case_priority#value}". Given this the the following formula can be used to return a due date:

```
var created_date = new Date ("{!createdAt}");
switch ("{!case_priority#value}") {
    case "High":
        return created_date.getTime() + (2 * (24*60*60*1000));
    case "Medium":
```

```
        return created_date.getTime() + (5 * (24*60*60*1000));
default:
        return created_date.getTime() + (7 * (24*60*60*1000));
}
```

Flags for Case Priority

This formula displays a green, yellow, or red flag image icon to indicate case priority.

In our sample case object, we add shared image fields for each icon and name them “Green Image” whose display will be given by the merge field “{!green_image#html}”, “Yellow Image” whose display will be given by the merge field “{!yellow_image#html}”, and “Red Image” whose display will be given by the merge field “{!red_image#html}”. Assume the priority of the case is tracked in a picklist field called “Case Priority” and the selected status value for this field is given by the merge field “{!case_priority#value}”. Given this we can return the appropriate image html to be displayed as follows:

```
if("{!case_priority#value}" == "Low")
    return "{!greenImage#html}";
else if("{!case_priority#value}" == "Medium")
    return "{!case_yellowImage#html}";
else if("{!case_priority#value}" == "High")
    return "{!redImage#html}";
```

IMPORTANT NOTE: When working with Image and Shared Image fields in formula expressions you must wrap them in quotes to avoid errors. In the above example “{!greenImage#html}” represents the green flag image.

Color Squares for Case Age

This formula displays a red, yellow, or green image icon depending on the value of a field called “Case Age” whose merge field is given by “{!case_age}”:

```
if({!case_age} > 20)
    return "{!redbox#html}";
else if({!case_age} > 20)
    return "{!yellowbox#html}";
else if({!case_age} > 20)
    return "{!greenbox#html}";
```

Formula Examples, Part III: Commission Calculations

The following are formulas that might be used in calculating sales commissions. In these examples we assume the existence of an object called “opportunity” which stores information about any opportunity from a lead.

Commission Amounts for Opportunities

This formula calculates the commission based on a flat 2% of the opportunity amount. This example calculates the commission amount for any opportunity that has a “Closed Won” stage. The value of this field will be the amount times 0.02 for any closed/won opportunity. Open or lost opportunities will have a zero commission value. The return type for this formula is Decimal rounded to 2 decimal places.

In our opportunity object assume the stage of the opportunity is given by the picklist field called “Opportunity Stage” and the selected stage value is given by the merge field “{!opportunity_stage#value}”. Also assume that the opportunity amount is stored in a currency field called “Amount” whose merge field is “{!amount}”. The commission can then be expressed as:

```
("(!opportunity_stage#value)" == "Closed-Won") ? Math.round(!amount) * 0.02 : 0;
```

Commission Deal Size

This formula calculates commission based on deal size returning a 9% commission for deals with amount over \$100,000 and an 8% commission for smaller deals. The return type for this formula is Decimal rounded to 2 decimal places.

In our opportunity assume the currency field “Deal Size” stores the deal amount and the merge field for this field is given by `{!deal_amount}`. The commission can then be calculated as:

```
{!deal_amount} > 100000) ? 0.09 : 0.08;
```

Commission Greater Than or Equal To

This formula returns “Large” for opportunities with a commission greater than or equal to \$1,000,000, and “Normal” for anything less.

In our opportunity object assume the commission amount is a formula field called “Commission Amount” and the merge field for this field is given by `{!commission_amount}`. We can then write the expression as:

```
{!commission_amount} > 1000000) ? "Large" : "Normal";
```

Commission Maximum

This formula determines the commission to log for an asset based on which of the following three things is larger: the user's commission percentage of the price, the price times the discount percent stored for the account, or \$100. This formula returns a Decimal rounded to 2 decimal places.

Assume we have an object called “asset” with a percent field “User Commission Percent” that stores the user's commission percentage. The merge field for this field is given by `{!user_commission_percent}`. Also assume the asset price is stored in a currency field called “Price” whose merge field is `{!price}` and the discount percent for the account is stored in a field called “Account Discount” whose merge field is `{!account_discount}`. Given this we can calculate the maximum commission as follows:

```
var user_comm = {!user_commission_percent} * {!price};  
var account_discount = {!price} * {!account_discount};  
  
return (user_comm > accountdiscount) ? ((user_comm > 100) ? user_comm : 100) : ((100 > account_discount) ? 100 : account_discount);
```

Formula Examples, Part IV: Contact Management

The following are formulas that might be used in contact management scenarios. In these examples we assume the existence of a “Contact” object.

Contact's Age

This formula is used to calculate a person's age based on a Birthdate. The person's Birthdate is subtracted from today's date, and the result is divided by the number of days in a year and rounded down to the nearest integer. This formula returns an Integer value.

In our contact object assume the birthdate of a contact is stored in a Date field called “Birth Date” whose merge field is given as `{!birth_date}`. The age of this contact can then be expressed as:

```
var today = new Date();
var birthdate = new Date("{!birth_date}");
return parseInt((today.getTime() - birthdate.getTime()) / (24*60*60*1000)) / 365;
```

Contact's Birthday in Current Month?

This formula displays the value "YES" if the contact's birthday falls in the current calendar month otherwise it displays "NO".

In our contact object assume the birthdate of a contact is stored in a Date field called "Birth Date" whose merge field is given as "{!birth_date}". We can then create the expression as follows:

```
var today = new Date();
var birthdate = new Date("{!birth_date}");
return (today.getMonth() == birthdate.getMonth()) ? "YES" : "NO";
```

Contact Identification Numbering

This formula displays the first five characters of the contact's last name and the last four characters of the contact's social security number separated by a dash.

In our contact object assume the last name of a contact is stored in a text field called "LastName" whose merge field is given as "{!lastName}" and the social security number of a contact is stored in a field called "SSN" whose merge field is "{!ssn}". We can then create the expression as follows:

```
"{!lastName}".substr(-5, 5) + "-" + "{!ssn}".substr(-4, 4);
```

Contact Preferred Phone

This formula displays the contact's preferred contact method — work phone, home phone, or mobile phone — based on a selected option in a "Preferred Phone" picklist.

In our contact object assume the preferred phone picklist is called "Preferred Phone" and the selected preferred phone value is given by the merge field "{!preferred_phone#value}". The actual phone numbers for each of the phone types are given by the fields "Work Phone" with merge field {!phone}, "Home Phone" with merge field "{!home_phone}", and "Mobile Phone" with merge field "{!mobile_phone}". We can then create the expression as follows:

```
switch("{!preferred_phone#value}" ) {
  case "Work":
    return "Work: {!phone}";
  case "Home":
    return "Home: {!home_phone}";
  case "Mobile":
    return "Mobile: {!mobile_phone}";
  default:
    return "No Preferred Phone";
}
```

Dynamic Address Formatting

This formula field displays a formatted mailing address for a contact including spaces and line breaks where appropriate depending on the country for the account.

In our contact object assume the shipping country is tracked in a picklist called “Shipping Country” and the selected country value for a particular contact is given by the merge field “{!shipping_country#value}”. The shipping address is stored in fields such as “Shipping Street” whose merge field is “{!shipping_street}”, “Shipping City” whose merge field is “{!shipping_city}”, “Shipping State” whose merge field is “{!shipping_state}”, and “Zip Code” whose merge field is “{!zip_code}”. We can then express this as:

```
if({!shipping_country#value} == "USA")
return "{!shipping_street} <br> {!shipping_city}, {!shipping_state} {!zip_code}
<br> {!shippingCountry}";

if({!shipping_country#value} == "France")
return "{!shipping_street} <br> {!zip_code} {!shipping_city} <br>
{!shipping_country}";
```

Telephone Country Code

This formula determines the telephone country code of a contact based on the Mailing Country of the mailing address. This formula returns an Integer.

In our contact object assume the mailing country of a contact is stored in a picklist field called “Mailing Country” and the selected mailing country value for a contact is given by the merge field as “{!mailing_country#value}”. We can then express the telephone country code as follows:

```
switch("{!mailing_country#value}") {
  case "USA": return 1;
  case "Canada": return 1;
  case "France": return 33;
  case "Australia": return 61;
  case "UK": return 44;
  case "Japan": return 81;
  case "India": return 91;
  default: return 0;
}
```

Unformatted Phone Numbers

This formula removes the parentheses and dash characters from North American phone numbers. This is necessary for some auto-dialer software. The formula returns a String.

In our contact object assume the phone number of a contact is stored in a text field called “Phone” whose merge field is “{!phone}” and the country code of that phone is calculated in a field called “Country Code” whose merge field is “{!country_code}”. We can then express North American phone numbers without dashes and parentheses using the following expression:

```
var phone = "{!phone}";
if({!country_code} == "1") {
  phone = phone.replace(/-/,"");
  phone = phone.replace(/(/,"");
  phone = phone.replace(/)/,"");
}
return phone;
```

Formula Examples, Part V: Data Categorization

The following are formulas that might be used in the categorization of various data records.

Deal Size Large and Small

This formula displays "Large Deal" for deals over one million dollars or "Small Deal" for deals under one million dollars. The return value is a String.

Consider a Deals object and assume it has a currency field called "Sales Price" whose merge field is given by "{!sales_price}". We can then express the deal size using the following expression:

```
{!sales_price} > 1000000) ? "Large Deal" : "Small Deal";
```

Product Categorization

This formula categorizes a product by checking the content of a picklist field named Product Type and returning "Parts" for any product with the word "part" in it. Otherwise it returns "Service." Note that the values are case sensitive, so if a Product_Type field contains the text "Part" or "PART," this formula returns "Services." (We leave it as an enhancement for the reader to extend this formula to cover other cases.)

Consider a product object and assume it has a picklist field called "Product Type" and the selected type is given by the merge field "{!product_type#value}". We can express the category as:

```
("{!product_type#value}".match("Parts") == "Parts") ? "Parts" : "Services";
```

Formula Examples, Part VI: Date Formulas

The following are formulas that might be used when dealing with Dates in formula expressions.

Today's Day of Week(Number)

This formula calculates today's day of week as number. Day of the week returned is a number as in 0=Sunday, 1=Monday, 2=Tuesday, 3=Wednesday, 4=Thursday, 5=Friday, 6=Saturday. The return type for this formula is an Integer.

```
var today = new Date();  
return today.getDay();
```

Today's Day of Week (Text)

This formula calculates today's day as text. The return type for this formula is a String.

```
var myDays  
=["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"];  
var today = new Date();  
return myDays[today.getDay()];
```

Day of Week

This formula is used to calculate day of the week for a given date. The return type here is String.

If {!startdate} is the merge field representing a date field present in an object, the day of the week for this date can be calculated as follows:

```
var myDays  
=["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"];
```

```
var inputDate = new Date("{!startdate}");  
return myDays[inputDate.getDay()];
```

Current Time

This formula is used to calculate the current time and returns the current time as a String in the format hr:min:sec.

```
var d = new Date();  
var curr_hour = d.getHours(); // Returns hours  
var curr_min = d.getMinutes(); // Returns minutes  
var curr_sec = d.getSeconds(); // Returns seconds  
return curr_hour + " : " + curr_min + " : " + curr_sec;
```

Note: To create a JavaScript date instance for a given date, if `{!inputDate}` is the merge field for a given date or date/time field, use `var d = new Date("{!inputDate}")`;

Current Time (formatted for 12-hour display)

This formula returns the current time as a String in the format hr:min:sec AM/PM in a 12 hour, rather than 24 hour, format.

```
var today = new Date();  
var hours = today.getHours();  
var minutes = today.getMinutes();  
var seconds = today.getSeconds();  
var timenow = "" + ((hours > 12) ? hours - 12 : hours);  
if (timenow == "0")  
    timenow = "12";  
timenow += ((minutes < 10) ? ":0" : ":") + minutes;  
timenow += ((seconds < 10) ? ":0" : ":") + seconds;  
timenow += (hours >= 12) ? " PM" : " AM";  
return timenow;
```

Days until a target date

This formula calculates the number of days between two dates and returns a String. If the target date is given by the merge field `{!targetdate}`, the number of days can be calculated as follows:

```
var today = new Date();  
var targetdate = new Date("{!targetdate}");  
remaining = ((targetdate.getTime() - today.getTime()) / (24 * 60 * 60 * 1000));  
remaining = Math.round(remaining);  
return remaining + " days left";
```

Days until end of Month

This formula calculates the number of days remaining from the given date until the end of the month the date is in. The return type for this formula is an Integer.

```
var inputDate = new Date("{!input_date}");  
var m = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31];  
var totaldays = 0;  
  
if (inputDate.getMonth() != 2)
```

Rollbase Formula Guide

```
totaldays = m[ inputDate.getMonth(); - 1];  
//handle leap year case  
if ( inputDate.getYear()%4 != 0)  
    totaldays = m[1];  
if ( inputDate.getMonth()%100 == 0 && inputDate.getMonth();%400 != 0)  
    totaldays = m[1];  
else  
    totaldays = m[1] + 1;  
return (totaldays - inputDate.getDate());
```

Fiscal Year Calculation:

This formula calculates the number of fiscal year any object record falls under given the start date for that object record.

Let us assume, we want to calculate the fiscal year a particular campaign falls under. The start date of the campaign is given by the Date field "Start Date" whose merge token is given "{!start_date}" . In this case we assume that a fiscal year spans from July 1 to June 31. The formula is thus calculated as

```
st_date = new Date("{!start_date}");  
st_month = st_date.getMonth();  
st_year = st_date.getFullYear();  
result = "FY";  
  
yr = "";  
yr = (st_month <=5 && st_month >=0) ? yr+st_year : yr+(st_year+1);  
yr = yr.substr(-2,2);  
return result+yr;
```

Quarter Calculation

This formula is used to identify which Quarter the campaign begins in using the campaign start date. Lets say the start date of the campaign is given by the Date field "Start Date" whose merge token is given "{!start_date}" and the campaign quarters are broken out as follows:

- Q1: July - September
- Q2: October - December
- Q3: January - March
- Q4: April – June

The formula is now calculated as,

```
s_date = new Date("{!start_date}");  
s_month= s_date.getMonth();  
if(s_month <=8 && s_month >=6)  
    return "Q1";  
if(s_month <=11 && s_month >=9)  
    return "Q2";  
if(s_month <=2 && s_month >=0)  
    return "Q3";  
if(s_month <=5 && s_month >=3)  
    return "Q4";
```

Formula Examples, Part VII: Discounting

The following are formulas that might be used when calculating discounts.

Maintenance and Services Discount

This formula displays whether an opportunity was discounted or not. It displays "Discounted" on an opportunity if its maintenance amount and services amount do not equal the opportunity Amount standard field value.

Otherwise, it displays "Full Price." The return type for this field is String.

In our opportunity object assume we have two currency fields called "Service Amount" whose merge field is given as "{!service_amount}" and "Maintenance Amount" whose merge field is given by "{!maintenance_amount}". The discount can be expressed as follows:

```
(((!maintenance_amount) + {!service_amount}) != {!amount}) ? "Discounted" : "Full Price";
```

Opportunity Discount Amount

This formula calculates the opportunity Amount less the Discount Amount. This formula returns a Decimal rounded to 2 decimal places.

In our opportunity object assume the opportunity amount is given by the currency field "Amount" whose merge field is given as "{!amount}", and the discount amount on the opportunity is stored in a field called "Discount Amount" whose merge field is given as "{!discount_amount}":

```
{!amount} - {discount_amount}
```

Formula Examples, Part VIII: Employee Management

The following are formulas that might be used when working with employee data. In these examples assume employee related information is stored in an object called "employee".

Total Hours Worked Per Week

This formula calculates the total hours worked by an employee in a week. The return type here is an Integer.

In our employee object assume that the number of hours worked each day is tracked in fields called "Monday", "Tuesday", "Wednesday", "Thursday", "Friday" and their respective merge fields are "{!hrsMonday}", "{!hrsTuesday}", "{!hrsWednesday}", "{!hrsThursday}", "{!hrsFriday}". This becomes a simple calculation as follows:

```
{!hrsMonday}+{!hrsTuesday}+{!hrsWednesday}+{!hrsThursday}+{!hrsFriday};
```

Employee Weekly Pay

This formula calculates the total weekly pay for an employee. The return type is a Currency.

In our employee object assume that the regular pay rate for the employee is stored in a field called "Regular Pay" whose merge field is given as "{!regular_pay}", and the overtime rate is stored in a field called "OverTime Pay" whose merge field is given as "{!overtime_pay}". The total hours worked by an employee is stored in a field called "Total Work Hours" whose merge field is "{!total_work_hours}". Given this the weekly pay can be calculated as:

```
( {!totalHours} <= 40 ) ? {!totalHours} * {!regPay} : ((40 * {!regPay} ) + (!totalHours - 40) * {!overtimePay});
```

Bonus Calculations

This formula calculates the bonus amount for an employee. The return type for this formula is a Currency.

In our employee object assume that the gross income of the employee is stored in a currency field called "Gross

Income” whose merge field is given by `{!gross_income}` and the bonus percentage is stored in a percent field called “Bonus Percent” whose merge field is given as `{!bonus_percent}`. The company’s performance amount is stored in a field called “Company Performance” whose merge field is given as `{!company_performance}` and the number of employees is stored in a field called “No of Employees” whose merge field is `{!No_of_employees}`. Given this we can calculate the bonus as:

```
var performance_divided = {!company_performance} / {!No_of_employees}
var bonus = {!gross_income} * {!bonus_percent}
return (performance_divided < bonus) ? performance_divided : bonus;
```

Formula Examples, Part IX: Expense Tracking

The following are formulas that might be used when working with expenses. In these examples assume expense related information is stored in an object called “expense”.

Mileage Calculation

This formula calculates mileage expenses for visiting a customer site at 35 cents a mile. The formula returns a decimal rounded to 2 decimal places. Assume the expense object has a field called “Miles Driven” whose merge field is given as `{!miles_driven}`. Mileage expenses can be calculated as follows:

```
{!miles_driven} * 0.35
```

Formula Examples, Part X: Financial Calculations

The following are formulas that might be used when working with financial data.

Compound Interest

This formula calculates the interest you will have after T years, compounded M times per year, returning a Currency value.

Consider an object called “Bank Account” that has fields “Principal” whose merge field is given as `{!principal}`, “Interest Rate” whose merge field is given as `{!interest_rate}`, “No of Years” whose merge field is given as `{!no_of_years}` and the frequency at which the interest should be compounded given by the field “Frequency” whose merge field is `{!frequency}`. Given this we can calculate compound interest as follows:

```
{!principal} * (1 + {!interest_rate} / {!frequency}) ^ ( {!no_of_years} *
{!frequency})
```

Compound Interest Continuous

This formula calculates the interest that will have accumulated after T years, if continuously compounded. This can be accomplished by modifying our previous example to:

```
{!principal} * Math.exp({!interest_rate} * {!no_of_years})
```

Gross Margin

This formula provides a simple calculation of gross margin and returns a Currency field.

Consider a product object that has fields “Total Sales” whose merge field is `{!total_sales}` and “Goods Cost”

whose merge field is “{!goods_cost}”. Gross margin can be calculated as:

```
{!total_sales} - {!goods_cost}
```

Gross Margin Based on Margin Percent

This formula calculates the gross margin based on a margin percent. Here we assume a product object has another field called “Margin Percent” whose merge field is “{!merge_field}” in addition to “Total Sales” and “Goods Cost” fields:

```
{!margin_percent} * {!total_sales} * {!goods_cost}
```

Payment Due Indicator

This formula returns the date five days after the contract start date as the Payment Due Date.

In an account object assume there is a field called “Start Date” whose merge field is “{!start_date}” that stores the contract start date for this object:

```
if("{!pay_due_date}" != "")
{
    var startdate = new Date ("{!start_date}");
    return startdate.getTime() + (5 * (24*60*60*1000));
}
```

Payment Status

This formula determines if the payment due date is past and the payment status is "UNPAID". If so, it returns the text "PAYMENT OVERDUE" and if not, it leaves the field blank. The return type of this formula is String.

In an account object assume there are fields “Payment DueDate” whose merge field is given as “{!payment_duedate}” and payment status tracked in a picklist called “Payment Status”. Given this our expression can be built as:

```
var today = new Date();
var paydate = new Date("{!payment_duedate}");
if (paydate.getTime() > today.getTime() && "{!payment_status#value}" == "UNPAID")
    return "PAYMENT OVERDUE";
else
    return "";
```

Shipment Tracking Integration Link

This formula creates a link to FedEx, UPS, or DHL shipment tracking websites, depending on the selected shipping method.

In a product object assume we have a picklist field called “Shipping Method” and the selected shipping method value is given by the merge field “{!shipping_method#value}”. Also assume the tracking ID for the package is stored in a field called “Track ID” whose merge field is “{!track_id}”. Given this we can create a tracking link as follows:

```
switch("{!shipping_method#value}") {
    case "FEDEX":
        return "<a
```

```
href='http://www.fedex.com/Tracking?ascend_header=1&clienttype=dotcom&cntry_code=us
&language=english&tracknumbers= {!track_id}' > Track </a>";

    case "UPS":
        return "<a
href='http://wwapps.ups.com/WebTracking/processInputRequest?HTMLVersion=5.0&sort_b
y=status&loc=en_US&InquiryNumber1={!track_id}&track.x=32&track.y=7'> Track </a> ";

    case "DHL":
        return "<a href='http://track.dhl-
usa.com/TrackByNbr.asp?ShipmentNumber={!track_id}' > Track </a>";

    default:
        return "";
}
```

NOTE: The parameters shown in this example for FedEx, UPS, and DHL websites are for demonstration purposes only and do not represent the correct parameters for all situations.

Formula Examples, Part XI: Lead Management

The following are formulas that might be used when working with leads. In these examples assume lead related information is stored in an object called “lead”.

Lead Aging (for open leads)

This formula checks to see if a lead is open and if so, calculates the number of days it has been open by subtracting the date and time created from the current date and time. The return type is an Integer.

In our Lead object assume the lead status is tracked in a picklist field called “Lead Status” and the selected status for is given by the merge field “{!lead_status#value}”. Also assume the date the lead was created is tracked in Rollbase System Information field called “Created At” whose merge field is “{!createdAt}”. Given this the number of days the lead is open can be expressed as follows:

```
var today= new Date();
var created_date = new Date("{!createdAt}");
return ("{"!lead_status#value}" == "Open") ? parseInt((today.getTime() -
created_date.getTime()) / (24*60*60*1000)) : 0;
```

Lead Data Completeness

This formula calculates the percentage of lead data completed by your sales personnel. The return type for this formula is a String.

In our Lead object this formula checks the values of two fields: “Phone” whose merge field is “{!phone}” and “Email” whose merge field is “{!email}”. If the fields are empty the formula returns the value “0%”. The total percentage is computed by checking if each field has data and if so adding an appropriate percentage to the result variable:

```
var result =0;
if("{!phone}"== null || "{!phone}" == "")
    result =1;
if("{!email}" == null || "{!email}" == "")
    result +=1;
result *= 50;
return result+ "%";
```

Lead Numbering

This formula returns a numeric value for the text value in an auto-number field called Lead Number. This technique can be useful if you want to use an auto-number field in a calculation, such as round-robin or other lead routing purposes.

In our Lead object assume the field called “Lead Number” stores the lead number which is an Auto-Number field and the merge field for this field is given as “{!lead_number}”. Given this the lead number formula used to convert the auto-number text to a numeric value is:

```
parseInt("{!lead_number}", 10);
```

Round Robin Assignment of Cases or Leads

The following formula is an example showing how automatic lead assignment might occur. The formula returns an Integer.

In our Lead object assume you have three lead queues and you want to assign an equal number of incoming leads to each queue. The lead number for a lead is given by the auto number field called “Lead Number” whose merge field is given as “{!lead_number}”. You can use the result of this formula to determine which queue to place the lead in:

```
var leadNo = parseInt("{!lead_number}", 10);  
return leadNo % 3;
```

Formula Examples, Part XII: Opportunity Management

The following are formulas that might be used when working with opportunities. In these examples assume opportunity related information is stored in an object called “opportunity”.

Days Left to Close

This formula returns the expected number of days left to the close date of an opportunity. This formula returns an Integer.

In our opportunity object assume the expected close date is stored in a Date field called “Expected Close Date”, whose merge field is given as “{!expected_close_date}”. We can calculate the formula as:

```
var today = new Date();  
var expected_close_date = new Date (“{!expected_close_date}”);  
return parseInt(( expected_close_date.getTime() - today.getTime() ) / (24*60*60*1000))
```

Display Close Month for Reporting Purposes

This formula returns the month for the close date of an opportunity as a String. This formula can be useful, for example, when building a report that groups opportunities by the month of the Close Date.

In our opportunity object assume the close date is stored in a Date field called “Close Date” and the merge field for this field is given as “{!close_date}”. The month for this date can be calculated as:

```
var close_date = new Date (“{!close_date}”);  
var months = ["January", "February", "March", "April", "May", "June", "July", "August",  
"September", "October", "November", "December"];
```

```
return months[close_date.getMonth()];
```

Expected Product Revenue

This formula calculates total revenue from multiple products, each with a different probability of closing, and returns a Currency value.

In our product object assume there is a relationship with an order object defined by the relationship name R850347. The probability of an order is tracked in a field called “probability” and the related merge field for this field is given by “{!R850347.probability}”. Also assume the revenue for an order is tracked in a field called “Revenue” and the related merge field for this field is given by “{!R850347.revenue}”. Given this the expected product revenue can be calculated as:

```
var total =0;
{!#LOOP_BEGIN.R850347}
    total += {!R850347.probability} * {!R850347.revenue}
{!#LOOP_END.R850347 }
return total;
```

Maintenance Fee

This formula calculates maintenance fees as 20% of license fees per year, and returns the result as a Currency value.

In our opportunity object assume the opportunity amount is stored in a currency field called “Amount” whose merge field is “{!amount}” and the maintenance years is tracked in a field called “Maintenance Years” whose merge field is “{!maintenance_years}”. The maintenance fee can then be expressed as:

```
{!amount} * {!maintenance_years} * 0.2
```

Monthly Subscription-Based Calculated Amounts

This formula calculates an opportunity amount based on a monthly subscription rate multiplied by the subscription period, returning a Currency value.

In our opportunity object assume a monthly opportunity amount is stored in a currency field called “Monthly Amount” whose merge field is “{!monthly_amount}” and a subscription period is tracked in a field called “Subscription Period” whose merge field is “{!subscription_period}”. The monthly fee can then be calculated as:

```
{!monthly_amount} * {!subscription_period}
```

Monthly Value

This formula divides the total annual amount of an opportunity by 12 months to calculate the monthly value.

In our opportunity object assume the total annual opportunity amount is stored in a currency field called “annual_amount” whose merge field is “{!annual_amount}”. The monthly value can be computed as follows:

```
{!annual_amount}/12
```

Opportunity Additional Costs

This formula calculates the sum of the opportunity Amount, maintenance amount, and services fees, returning a

Currency.

In our opportunity object assume the opportunity amount is stored in a currency field called “Amount” whose merge field is “{!amount}”, the maintenance amount is tracked in a field called “Maintenance Amount” whose merge field is “{!maintenance_amount}”, and the service fee is stored in a field called “Service Fee” whose merge field is “{!service_fee}”. Given this the opportunity additional costs can be simply calculated as:

```
{!amount} + {!maintenance_amount} + {!service_fee}
```

Opportunity Categorization

This formula uses conditional logic to determine an Opportunity category based on the value of the Opportunity Amount. Opportunities with amounts less than \$1500 are "Category 1", those between \$1500 and \$10000 are "Category 2", and the rest are "Category 3".

In our opportunity object assume the opportunity amount is stored in a currency field called “Amount” whose merge field is given as “{!amount}”. The category can be determined as follows:

```
if({!amount} < 1500)
    return "Category 1";
else
    return ({!amount} > 10000) ? "Category 3" : "Category 2";
```

Opportunity Data Completeness

This formula takes a group of five opportunity fields and calculates the percentage of them that have been completed.

In our opportunity object assume the opportunity amount is stored in a currency field called “Amount” whose merge field is “{!amount}”, the maintenance amount is tracked in a field called “Maintenance Amount” whose merge field is “{!maintenance_amount}”, the service amount is stored in a field called “Service Amount” whose merge field is “{!service_amount}”, the discount percentage is stored in a field called “Discount Percent” whose merge field is “{!discount_percent}”, and the timeline for the opportunity is stored in a field called “Timeline” whose merge field is “{!timeline}”. Given this we can calculate opportunity data completeness as:

```
var result =0;
if("{!maintenance_amount}"== null || "{!maintenance_amount}" == "")
    result =1;
else if("{!service_amount}" == null || "{!service_amount}" == "")
    result +=1;
else if("{!discount_percent}" == null || "{!discount_percent}" == "")
    result +=1;
else if("{!amount}" == null || "{!amount}" == "")
    result +=1;
else if("{!timeline}" == null || "{!timeline}" == "")
    result +=1;
return result / 5;
```

Opportunity Expected License Revenue

This formula calculates expected revenue for licenses based on probability of closing and returns a Currency value.

In our opportunity object assume the expected license revenue is stored in a field called “Expected Revenue Licenses” whose merge field is “{!expected_revenue_licenses}” and the probability of closing is tracked in a field called “Probability” whose merge field is “{!probability}”. Expected license revenue is then computed as:

```
{!expected_revenue_licenses} * {!probability}
```

Opportunity Reminder Date

This formula creates a reminder date for seven days before the close date of an opportunity, returning a Date value. This is a good example of how a formula field that can be used in a workflow event to create a new event record for the appropriate user to take action.

In our opportunity object assume the close date for an opportunity is stored in a field called “Close Date” whose merge field is “{!close_date}”. An opportunity reminder can be calculated as:

```
var close_date = new Date("{!close_date}");  
return closeDate.getTime() + (7 * (24*60*60*1000));
```

Opportunity Split Credit for Sales Representatives

This formula divides opportunity amount between multiple sales representatives and returns a Currency value.

In our opportunity object assume the total opportunity amount is stored in a currency field called “Amount” whose merge field is “{!amount}” and the total number of representatives associated with the opportunity are tracked in a field called “Total Reps” whose merge field is “{!total_reps}”. Opportunity split credit can be calculated as follows:

```
{!amount} / {!total_reps}
```

Opportunity Total Deal Size

This formula calculates the sum of maintenance and services amounts and returns a Currency value.

In our opportunity object assume the opportunity amount is stored in a currency field called “Amount” whose merge field is “{!amount}” and the maintenance amount is tracked in a field called “Maintenance Amount” whose merge field is “{!maintenance_amount}”. Also assume the service amount is stored in a field called “Service Amount” whose merge field is “{!service_amount}”. Given this the total opportunity size is:

```
{!amount} + {!maintenance_amount} + {!service_amount}
```

Professional Services Calculation

This formula estimates professional service fees at an average loaded rate of \$1200 per day, returning a Currency value.

In our opportunity object assume the number of consulting days are stored in a field called “Consulting Days” whose merge field is “{!consulting_days}”. We can then calculate services fees as follows:

```
{!consulting_days} * 1200
```

Shipping Cost by Weight

This formula calculates postal charges based on weight, returning a Currency.

In our opportunity object assume package weight is stored in a field called “Package Weight” whose merge field is “{!package_weight}”, and the cost per lb is stored in a field called “Cost per LB” whose merge field is “{!cost_per_lb}”. Shipping costs can then be calculated as:

```
{!package_weight} * {!cost_per_lb}
```

Shipping Cost Percentage

This formula calculates shipping costs as a fraction of the total amount, returning a String value representing the percentage.

In our opportunity object assume the shipping cost is stored in a field called “Shipping Cost” whose merge field is “{!shipping_cost}” and the total amount is stored in a field called “Total Amount” whose merge field is “{!total_amount}” the shipping cost is calculated as

```
((!shipping_cost) / {!total_amount}) * 100 + "%"
```

Commission

This formula calculates the 2% commission amount of an opportunity that has a probability of 100%. All other opportunities will have a commission value of zero. This formula returns a Currency value.

In our opportunity object assume the probability for an opportunity is stored in a field called “Opportunity” whose merge field is “{!opportunity}” and the opportunity amount is stored in a field called “Amount” whose merge field is “{!amount}”. 2% commission on a 100% probable opportunity can be calculated as follows:

```
((!probability) == 100) ? Math.round(!amount) * 0.02 : 0;
```

Total Contract Value from Recurring and Non-Recurring Revenue

This formula calculates both recurring and non-recurring revenue streams over the lifetime of a contract.

In our account object assume the non-recurring revenue is stored in a field called “Non Recurring Revenue” whose merge field is “{!non_recurring_revenue}”, the contract length for the account is stored in a field called “Contract Length” whose merge field is “{!contract_length}”, and the recurring revenue amount is stored in a field called “Recurring Amount” whose merge field is “{!recurring_revenue}”. Total contract value can then be computed as:

```
{!non_recurring_revenue} + {!contract_length} * {!recurring_revenue}
```

Formula Examples, Part XIII: Pricing

The following are formulas that might be used when working with pricing scenarios.

Total Amount

This formula calculates a total amount based on unit pricing and total units, returning a Currency value.

In a product object assume unit price is stored in a field called “Unit Price” whose merge field is given as “{!unit_price}” and the total number of units is stored in a field called “Total Units” whose merge field is “{!total_units}”. The total amount is calculated as:

```
{!unit_price} * {!total_units}
```

User Pricing

This formula calculates a price per user license, returning a Currency value.

In our account object assume total license revenue is stored in a field called “Total License Revenue” whose merge field is given as “{!total_license_rev}” and the number of user licenses for the account is tracked in a field called “No of User Licenses” whose merge field is “{!no_of_user_licenses}”. Given this we can compute the price per user as:

```
{!total_license_rev} / {!no_of_user_licenses}
```

Formula Examples, Part XIV: Project Management

The following are some common formulas used in project management applications. Consider a project object and a project task object that are related in a “one project to many project tasks” relationship.

Task Due Date

This formula is used to calculate the due date for a project task when a start date and duration for the task is given. The return type of this formula is Date.

In our project object assume the start date of a project task is given as “Start Date” and the merge field for this field is given by “{!start_date}”, and the duration for this project task is tracked in a field called “Duration” whose merge field is “{!duration}”. The due date for this task would be calculated as follows:

```
var start_date = new Date("{!start_date}");  
return start_date.getTime() + ({!duration} * (24*60*60*1000));
```

NOTE: 24*60*60*1000 is the number of milliseconds in a single day

Days Overdue

This formula calculates the number of days a project task is overdue when a due date for a task is provided. Here the return type is a String.

In our Project Task object assume the due date of the task is stored in a Date field called “Due Date” and the merge field for this field is given by “{!duedate}”. The number of days overdue is calculated as:

```
var duedate = new Date("{!duedate}");  
var today = new Date();  
if(duedate < today)  
    days = parseInt((today.getTime() - duedate.getTime()) / (24*60*60*1000));  
return (days <= 0)? "On Schedule" : days + " overdue";
```

Number of Project Tasks

This formula is used to calculate the number of project tasks that a project is related to, returning an Integer.

In our project object assume each project has a number of tasks and the task information is stored in the project task object. The relationship defined between these two objects is represented by the integration name “R850347”. Therefore we can calculate the number of related tasks using the group function SUM or COUNT.

```
SUM.R850347 ( 1 )
```

Project Completion Status as a Percentage

This formula calculates the percentage project completion based on the status of related tasks, returning a String representing a percent value.

| % Complete |
|------------|
| 67% |
| 59% |
| 55% |

In our project object assume each project has a number of tasks and the task information is stored in the project task object. The relationship defined between these two objects is represented by the integration name "R850347". The progress of a project task is tracked in a field called "Percent Complete" whose merge field is given by "{!R850347.percent_complete}". The weight of each task is given by a picklist field called "Task Weight" and the selected weight value of the each is given by the merge field "{!R850347.task_weight#value}". Using this information we can now calculate the total % of project completion using the group function LOOP:

```
var perc = 0;
var weight = 0;
var totalweight = 0;
var total = 0;

{!#LOOP_BEGIN.R850347}

perc = {!R850347.percent_complete};
if("{!R850347.task_weight#value}" == "Very Significant")
    weight = 4;
else if("{!R850347.task_weight#value}" == "Significant")
    weight = 3;
else if("{!R850347.task_weight#value}" == "Important")
    weight = 2;
else if("{!R850347.task_weight#value}" == "Not Important")
    weight = 1;

totalweight = totalweight + weight;
total = total + (perc * weight);

{!#LOOP_END.R850347}

total = Math.ceil(total / totalweight);
return total+"%";
```

Project Progress Bar Formula

This formula renders the percentage of project completion as a progress bar returning a String of HTML.


```
return "{!redImage#html}";
```

Formula Examples, Part XV: Scoring & Rating

The following are formulas that might be used when working with scoring, rating or ranking scenarios.

Lead Scoring


This formula is used to scores leads. Here a higher score is provided for phone calls than website requests in the lead object.

In our Lead object assume the lead source is tracked in a picklist called "Lead Source" and the merge field for a particular lead's source value is given by "{!lead_source#value}". To return a different score based on this value:

```
switch("{!lead_source#value}") {  
  case "Phone":  
    return 2;  
  case "Web":  
    return 1;  
  default:  
    return 0;  
}
```

Star Ratings

This formula displays a set of one to five stars to indicate a rating or score using Shared Images that are stored as part of the object definition. The formula return type is a String.

| Star Rating | Starred Image |
|-------------|---|
| 5 |  |

In our Product object assume the product rating is tracked in a picklist field called "Rating" and the selected rating value is given by the merge field "{!rating#value}". We use Shared Image fields to store all of the rating images that we want to display with images called "Star1", "Star2", "Star3", "Star4", "Star5". The merge fields to display these images as HTML is given by "{!star1#html}", "{!star2#html}", "{!star3#html}", "{!star4#html}", "{!star5#html}". To display the correct image based on the rating value we can write the following:

```
if ("!rating#value"=="1")  
  return "{!star1#html}";  
else if ("!rating#value"=="2")  
  return "{!star2#html}";  
else if ("!rating#value"=="3")  
  return "{!star3#html}";  
else if ("!rating#value"=="4")  
  return "{!star4#html}";  
else if ("!rating#value"=="5")  
  return "{!star5#html}";  
else  
  return "{!star0#html}";
```

Horizontal Bars to Indicate Scoring

This formula displays a horizontal bar (red on a white background) of a length that is proportional to a numeric

score, assuming the score is on a scale of 0 to 100.



In our Lead object assume an industry score is given by a field called “Industry Score” whose merge field is “{!industry_score}” and {!industry.redblock#url} is the merge field for the url of the shared image redblock.gif (a small red block image used here to represent 10 points). Given this horizontal bar can be displayed as follows:

```
var num = Math.round({!industryScore});
return "<table align='left' style='border: black 1px solid' width='100px'> <tr><td
height='10px'> <img src='{!industry.redblock#url}' height='10px' width='"+num+"px'
/> </td> </tr> </table>" +num;
```

Formula Examples, Part XVI: Miscellaneous

The following are formulas that might be used in other types of calculations.

Celsius to Fahrenheit

This formula converts Celsius to Fahrenheit and returns a Decimal value. Assume the Celsius value is stored in a field called “Degrees Celsius” and the merge field is given as “{!degrees_celsius}”. The conversion to Fahrenheit can be expressed as follows:

```
1.8 * {!degrees_celsius} + 32
```

Miles to Kilometers

This formula converts miles to kilometers returns a Decimal value. Assume the number of miles is stored in a field called “Miles” and the associated merge field as “{!miles}”. Converting miles to kilometers can then be expressed as:

```
{!miles} * 1.609344
```

Kilograms to Pounds

This formula converts kilograms to pounds and returns a Decimal value. Assume the number of kilograms is stored in a field called “Kgs” and the merge field is given as “{!kgs}”. Converting kilograms to pounds can then be expressed as:

```
{!kgs} * 2.20462262
```