BUSINESS OBJECTS

# Crystal Enterprise Report Application Server (CE RAS) 9

## Using Formulas with the Java SDK

---

## Overview

This document discusses formulas, record selection formulas, and group selection formulas in conjunction with the Crystal Enterprise Report Application Server (CE RAS) 9 Java SDK.

| NOTE | Crystal Enterprise Report Application Server (CE RAS) 9 is shipped on a separate CD with Crystal Reports 9 Professional, Developer, and Advanced editions. |
|---|---|

## Contents

# Introduction

Crystal Reports includes an expressive formula language that allows you to select, aggregate, summarize, and manipulate many types of fields and field data in a report. Formula fields are used to display data, conditionally format fields, sections, and objects, as well as for record and group selection.

When you place a field on a report, values for this field from every record in the active database table(s) are displayed by default. However, in some cases, you may want to display only a subset of those values. The CE RAS SDK provides model and controller classes to add, remove, and modify record selection formulas in a report.

When you group or summarize data, all the groups in the report are included by default. If you do not wish to include all the groups, use a group selection formula. The CE RAS SDK also provides model and controller classes to add, remove, and modify group selection formulas in a report.

This document discusses formulas, record selection formulas, and group selection formulas in conjunction with the Crystal Enterprise Report Application Server (CE RAS) 9 Java SDK.

# Formula Fields

Usually, the data needed for a report already exists in a database (for example, the fields that describe the details of sales orders). However, you may wish to include data on a report that does not exist in a database, in which case you need to create a formula. For example, to calculate the number of days it takes to process each order, use a formula that calculates the number of days between the order date and the ship date. The CE RAS SDK provides model and controller classes to add, remove, and modify formula fields.

## Creating a new formula field

Formula fields are added to a report using the **FormulaFieldController** object. A formula field is represented by the **FormulaField** class. It extends the **Field** class that is the base class for database and parameter fields.

A valid formula field has at least four properties:

- name
- syntax
- type
- formula text

## Creating a simple formula field

The following sample code creates a formula field called "formula1" with the formula text 'Howdy':

```
// Create a new FormulaField object

FormulaField formulaField = new FormulaField();

// The string `Howdy' is handled by the Crystal formula
// parser. Since this is setting a string literal for the
// formula text, the string must be enclosed in single
quotes.

formulaField.setText("'Howdy'");

// The string sent to the Name property must be unique not
// only for all formulas in the report, but also for all
// objects in the report (for example, database fields,
// group name fields, and text fields).

formulaField.setName("formula1");

// The value for the formula syntax is a field of the
// FormulaSyntax class, which offers a choice between SQL,
// Crystal, and Basic syntaxes. For formula fields, choose
// either Crystal or Basic syntax. Choosing SQL syntax
// changes it into an SQL Expression field.

formulaField.setSyntax(FormulaSyntax.crystal);

// The type must correspond to the value type of the
// evaluated formula text. 'Howdy' uses
// FieldValueType.stringField, since the single quotes
// denote that this is a string literal, whereas "3 * 4" is
// a FieldValueType.numberField

formulaField.setType(FieldValueType.stringField);
```

## Adding the formula field to a report

Once the formula field object is initialized, it can be added to the report using the **FormulaFieldController**. This controller is used to add, remove, and modify formula fields. You retrieve this controller from a **ReportClientDocument** object using the **DataDefController** as follows:

```
report.getDataDefController().getFormulaFieldController().a
dd(formulaField);
```

## Displaying the formula field

Even though the formula field has been added to the report, it does not appear when you preview the report. The formula is, however, available to be used in a selection formula, text object, or another formula. Similarly, when you create a formula field in the Crystal Reports Designer, the formula field is visible in the Field Explorer, but it does not have a green checkmark next to it indicating that it is in use. However, once you drag and drop the formula field on to the report, the formula field is then in use.

To add the formula field to a report and display it as a field, you may use either the **ResultFieldController** or the **ReportObjectController** object.

### Displaying a formula field using the ResultFieldController

The following sample code demonstrates how to use the **ResultFieldController** object to display a formula field in the Details Section of a report:

```
report.getDataDefController().getResultFieldController.add(
-1, formulaField);
```

### Displaying a formula field using the ReportObjectController

What if you want to place the formula field somewhere else other than the Details section? Using the **ReportObjectController** object, you can choose the section to which to add the field, as well as format the field object. However, with this greater flexibility, you also then need to set additional properties in your application.

The following sample code demonstrates how to use the **ReportObjectController** object to display a formula field:

```
// Create new FieldObject
FieldObject field = new FieldObject();

// The FieldObject takes the formula field's FormulaForm
// property as its DataSource. The FormulaForm property
```

```
// returns a string representing the field data to be used
// (e.g. {Customer.Customer Name} for the 'Customer Name'
// field of the Customer table).  Since the formula field
// is already added to the report, the FieldObject can find
// the formula field and its data when the report is
processed.


field.setDataSource(formulaField.getFormulaForm());


// The FieldObject is sized and positioned using values
// specified in twips (1440 : 1 inch).


field.setTop(100);
field.setLeft(100);
field.setWidth(2880);


// The FieldObject borrows the type information from the
formula field


field.setFieldValueType(formulaField.getType());
ReportDefinition rptDef =
report.getReportDefController().getReportDefinition();
Section headerSection;
headerSection=
rptDef.getReportHeaderArea().getSections().getSection(0);


// The ReportObject controller adds the formula field to
// the specified section. The -1 value indicates the order
// of the object in the section. However, since the size
// and position of the field are specified, the order is
// not relevant.


report.getReportDefController().getReportObjectController()
.add(field, headerSection, -1);
```

## Accessing existing formula fields

To access a formula in a report, use the **DataDefController** object as follows:

```
Field fieldToModify;
fieldToModify = (Field)

// The clone method makes a new copy of an object, in this
// case, the first formula field
document.DataDefinition().getFormulaFields().getField(0).cl
one(true);
```

# Record Selection Formulas

Record selection formulas are added to a report using the **RecordFilterController** object. When a report is created, the record selection formula may change the SQL statement that is used to query the database for records.

The record selection formula is modeled using the **Filter** object. The filter contains a collection called **FilterItems**, consisting of one or more objects derived from the **FilterItem** class. The **FilterItem** class is the base class for two subclasses, **FieldRangeFilter** and **OperatorFilterItem**.

In the example selection formula `({Customer.Region} = 'CA') AND ({Customer.Country} = 'USA'),` The 'AND' is an **OperatorFilterItem** while the two joined expressions are **FieldRangeFilterItems**. (see Figure 1) **FieldRangeFilterItems** are so called because they specify a range of field values.

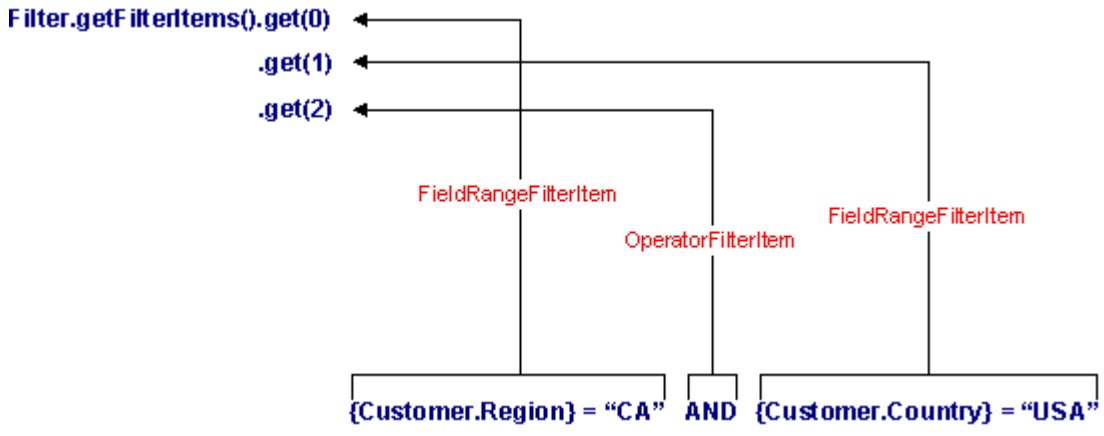


*Figure 1 – Filter Object Components*

# Creating a new record selection formula

Record selection formulas may be specified using either **FreeEditingText** property or **FilterItems**.

## Record selection using FreeEditingText

The following sample code demonstrates how to use the **FreeEditingText** property:

```
// When the selection formula is evaluated, the string is
// parsed into an array of FiterItems. Since this formula
// consists of two expressions and an operator, the
// FilterItems collection contains two
// FieldRangeFilterItems and one OperatorFilterItem.

com.crystaldecisions.sdk.occa.report.data.Filter filter =
new com.crystaldecisions.sdk.occa.report.data.Filter();
filter.setName("selformula");
filter.setFreeEditingText("({Customer.Region} = 'CA') AND
({Customer.Country} = 'USA')");
```

## Record selection using FilterItems

The other way of specifying a selection formula is to add each **FilterItem** individually. This method requires additional coding, but it is useful when your user interface allows you to choose one or more fields and operators instead of entering a selection formula. The following sample code demonstrates how to use **FilterItems**:

```
com.crystaldecisions.sdk.occa.report.data.Filter filter =
new com.crystaldecisions.sdk.occa.report.data.Filter();
filter.setName("selformula");

//get the collection of fields in the Customer table
Fields tableFields =
document.getDatabase().getTables().getTable(0).getDataField
s();

// create a FieldRangeFilterItem for the expression
// {Customer.Region} = 'CA'
FieldRangeFilterItem item1 = new FieldRangeFilterItem();

// add the value to the filter item
```

```
ConstantValue constant1 = new ConstantValue();
constant1.setValue("CA");
item1.getValues().add(constant1);

// add the field to the filter item
int index1 = tableFields.find("{Customer.Region}",
FieldDisplayNameType.formulaName, Locale.CANADA);
Field field1 = (Field) tableFields.getField(index1);
item1.setRangeField(field1);

// add the operation to the filter item
item1.setOperation(SelectionOperation.equal);
filter.getFilterItems().add(item1);

// create a OperatorFilterItem for the 'AND' operator
OperatorFilterItem newOperator = new OperatorFilterItem();
newOperator.setOperator("AND");
filter.getFilterItems().add(newOperator);

// create a filter item for the Customer.Country field
FieldRangeFilterItem item2 = new FieldRangeFilterItem();
ConstantValue constant2 = new ConstantValue();
constant2.setValue("USA");
item2.getValues().add(constant2);
int index2 = tableFields.find("{Customer.Country}",
FieldDisplayNameType.formulaName, Locale.CANADA);
Field field2 = (Field) tableFields.getField(index2);
item2.setRangeField(field2);
item2.setOperation(SelectionOperation.equal);
filter.getFilterItems().add(item2);

// Add the Record Selection Formula to the report
document.getDataDefController().getRecordFilterController()
.modify(filter);
```

As you can see, this method of using record selection formulas requires significantly more code than using the **FreeEditingText** property. It is recommended to use the **FreeEditingText** property to create new selection formulas, and **FilterItems** to modify or add parts to individual record selection formulas.

### Accessing a record selection formula

The following sample code demonstrates how to access a record selection formula from the report's DataDefinition:

```
com.crystaldecisions.sdk.occa.report.data.Filter filter =
document.getDataDefinition().getRecordFilter();
```

## Group Selection Formulas

When you group or summarize data, all groups are included by default in the resulting report. Sometimes, however, you want to restrict the number of groups. With a group selection formula you may:

- restrict groups by name.
- restrict groups by summarized values.

Group selection formulas are nearly identical to record selection formulas except that you use the **GroupFilterController** instead of the **RecordFilterController** to add, remove, and modify them. Another important difference is the types of formulas you specify, since group selection formulas are intended to operate on groups rather than records. If the report data has been grouped but not summarized, only group selection by name is possible.

### Group selection by name

To restrict a group by name, use a formula of the general form:

```
GroupName ({Table.Field}) = 'value'
```

The following sample code demonstrates how to filter groups in a report that is grouped by country so that only German customers are displayed:

```
com.crystaldecisions.sdk.occa.report.data.Filter grpFilter
= new

com.crystaldecisions.sdk.occa.report.data.Filter();

grpFilter.setName("groupSelectionFormula");

grpFilter.setFreeEditingText("GroupName
({Customer.Country}) = 'Germany'");

document.getDataDefController().getGroupFilterController().
modify(grpFilter);
```

## Group selection by summary

To restrict a group by summary value, use a formula of the general form:

```
sum ({Table.Summarised Field}, {Table.GroupedField}) >
value
```

Therefore, to filter groups in a report that is grouped by region so that only those customers with sales greater than $10,000 are displayed, use the following sample code:

```
com.crystaldecisions.sdk.occa.report.data.Filter sumFilter
= new

com.crystaldecisions.sdk.occa.report.data.Filter();

sumFilter.setName("groupSelectionFormula");

sumFilter.setFreeEditingText("Sum ({Customer.Region},
{Customer.Last Year's Sales}) > 10000");

document.getDataDefController().getGroupFilterController().
modify(sumFilter);
```

| NOTE | To highlight text in a PDF document for copying and pasting code, click the **Text Select Tool** toolbar button in Adobe Acrobat.<br>  <br>This procedure applies to Adobe Acrobat 4.0 and 5.0. |
|---|---|

| DISCLAIMER | The third party products discussed in this white paper were not fully tested in conjunction with Crystal Enterprise prior to its release. Officially supported Crystal Enterprise platforms are listed in the text file Platforms.txt found in the root of the Crystal Enterprise CD as well as in the Knowledge Base article c2009003 (search for this article number at http://support.crystaldecisions.com/kbase), which is more current.<br>The information in this document is provided as a courtesy to assist our customers with the configuration of our product in conjunction with these third party platforms.<br>In the event issues arise with an unsupported configuration, there is no escalation support; however, they will be considered during the development of the next generation of our product. |
|---|---|

# Finding More Information

For sample Crystal Enterprise Report Application Server (CE RAS) 9 Java applications, search for Dev_ras_9_jsp_samples.exe and Adv_ras_9_jsp_samples.exe on our support site at:

http://support.crystaldecisions.com/search

For more information on the objects, methods, and properties discussed in this document, consult the CE RAS 9 Javadocs.

For more information and resources, refer to the product documentation and visit the support area of the web site at: www.businessobjects.com

www.businessobjects.com