helpsystems

User Guide
Viewpoint
Client Report Option

Copyright Terms and Conditions

202107131250

Table of Contents

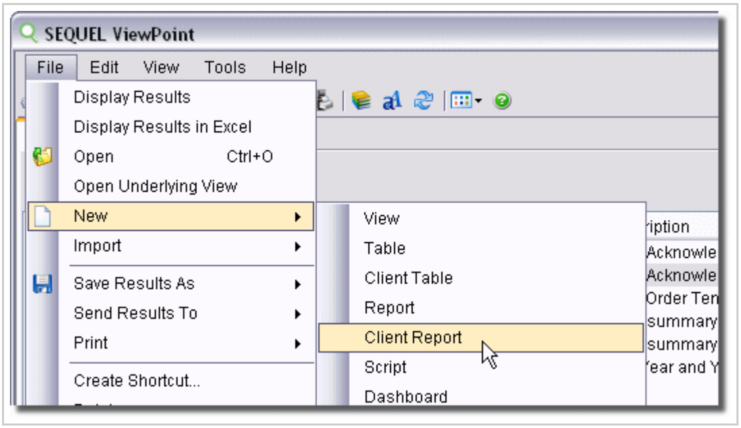
Client Report Option

New Client Report Wizard

The New Client Report Wizard will step through four screens to assist in the creation of most reports. You can set print attributes, select fields for display, create summary fields, and choose the level of summarization.

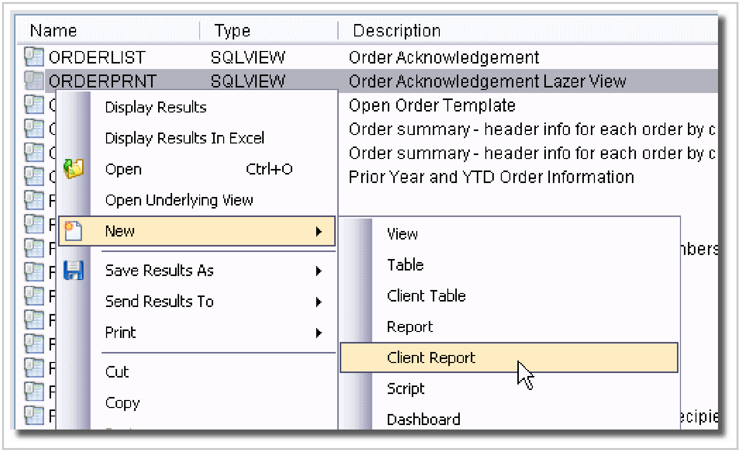
The Report Wizard is invoked any time a new report is started using any of the following methods:

From the ViewPoint Explorer, first highlight a view or table in the object list, and on the menu bar, select **File\New\Client Report.**



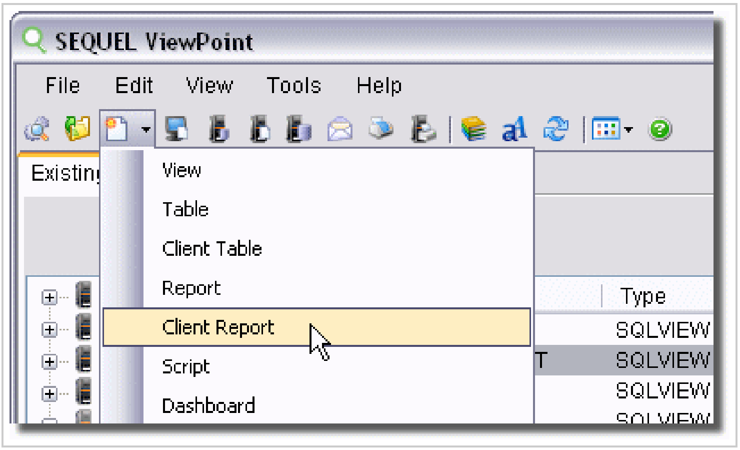
-or-

Right-click the desired view or table and select **New\Client Report** from the drop-down menu.


Name
Type
Description
ORDERLIST SQLVIEW Order Acknowledgement
ORDERPRNT SQLVIEW Order Acknowledgement Lazer View
Open Order Template
Order summary - header info for each order by c
Order summary - header info for each order by c
Prior Year and YTD Order Information
Display Results
Display Results In Excel
Open Ctrl+O
Open Underlying View
New
Save Results As
Send Results To
Print
Cut
Copy
View
Table
Client Table
Report
Client Report
Script
Dashboard

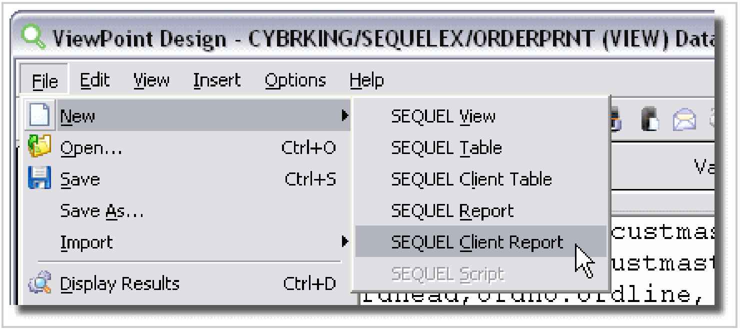
-or-

Highlight a view or table in the object list, press the **New** button on the toolbar, and select **Client Report.**



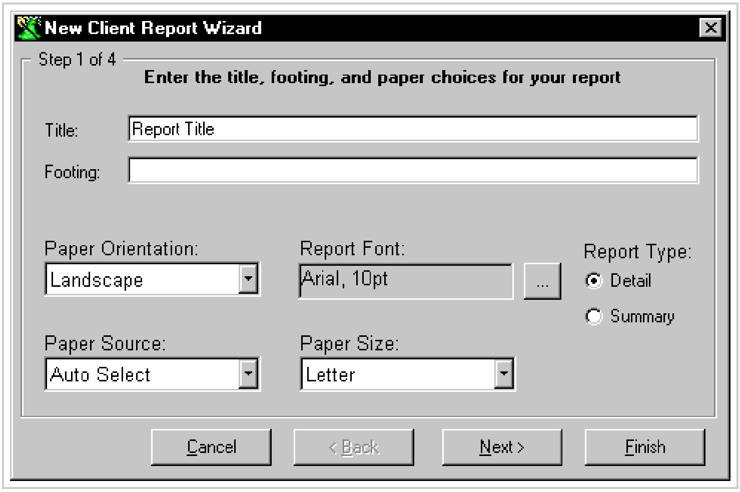
-or-

From the menu bar on the Design View display or the Table Designer, select **File\New\Client Report.**



Step 1 - CRO Attributes

The first step of the New Client Report Wizard is to define basic report attributes. Specify title and footer, default format, and forms attributes.


New Client Report Wizard

Step 1 of 4

Enter the title, footing, and paper choices for your report

Title: Report Title

Footing:

Paper Orientation: Landscape

Report Font: Arial, 10pt

Report Type: [✓ Detail] Summary

Paper Source: Auto Select

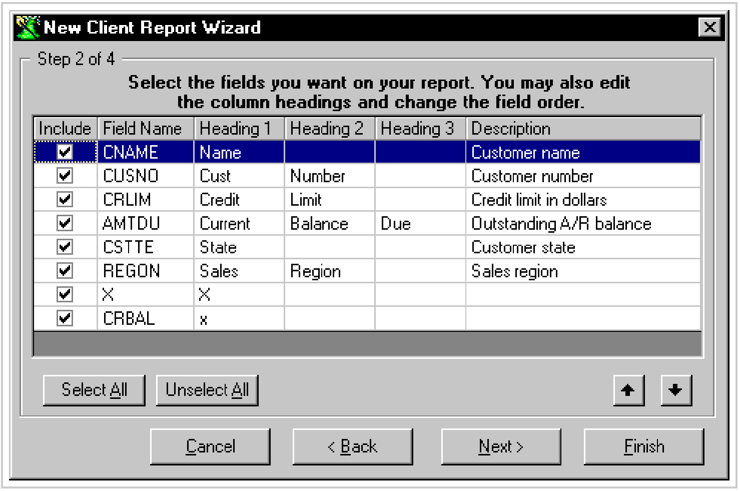
Paper Size: Letter

Press the **Cancel** button to exit the Wizard. Press the **Next** button to go to Step 2. Press the **Finish** button to complete the report and go directly to the Report Layout screen. (defaults will be supplied for any skipped steps)

Step 2 - Field Selection

The second step of the New Client Report Wizard is to select fields from the view to display on the report.

Usually, you will want to leave all the boxes ‘checked’. Bear in mind that a single view can drive more than one report. For instance, some of the fields from a view can be used in one report, and a different set of fields, *from the same view*, can be used in a second report. This screen will assist in the selection of the fields. If you are unhappy with the results of the wizard, you may easily restart and make diferent choices.



Select the fields you want on your report. You may also edit the column headings and change the field order.

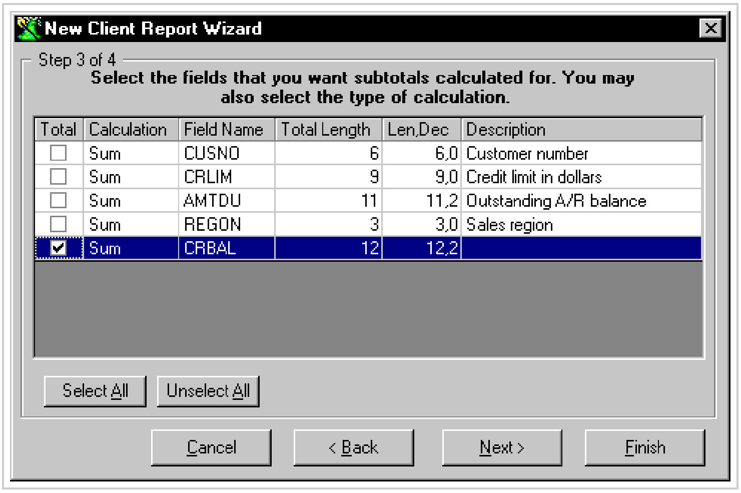
Include	Field Name	Heading 1	Heading 2	Heading 3	Description
✓	CNAME	Name			Customer name
✓	CUSNO	Cust	Number		Customer number
✓	CRLIM	Credit	Limit		Credit limit in dollars
✓	AMTDU	Current	Balance	Due	Outstanding A/R balance
✓	CSTTE	State			Customer state
✓	REGON	Sales	Region		Sales region
✓	X	X			
✓	CRBAL	x			

Press the **Cancel** button to exit the Wizard. Press the **Back** button to go to Step 1. Press the **Next** button to go to Step 3. Press the **Finish** button to complete the report and go directly to the Report Layout screen. (defaults will be supplied for any skipped steps)

Step 3 - Create Subtotals

The third step of the New Client Report Wizard is to choose which summary level function to apply to any of the numeric columns.

All numeric fields from the view will be displayed. Check the field (column) that is to be subtotaled. One of four summary level functions can be applied to the column.



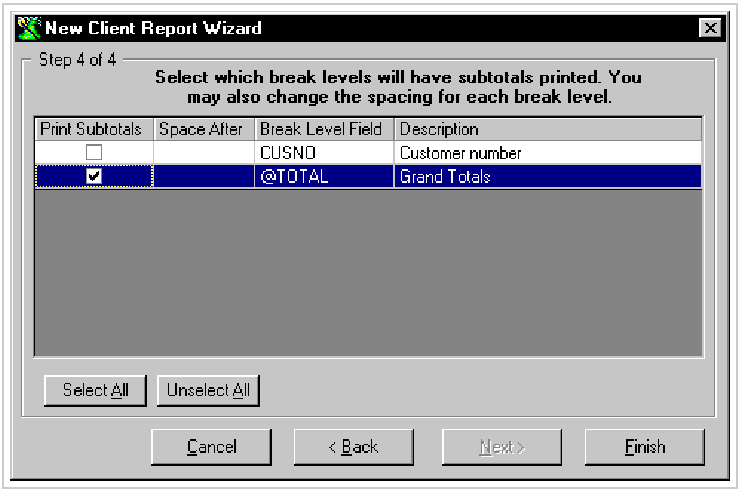
Select the fields that you want subtotals calculated for. You may also select the type of calculation.

Total	Calculation	Field Name	Total Length	Len,Dec	Description
☐	Sum	CUSNO	6	6,0	Customer number
☐	Sum	CRLIM	9	9,0	Credit limit in dollars
☐	Sum	AMTDU	11	11,2	Outstanding A/R balance
☐	Sum	REGON	3	3,0	Sales region
✓	Sum	CRBAL	12	12,2	

Press the **Cancel** button to exit the Wizard. Press the **Back** button to go to Step 2 Press the **Next** button to go to Step 4. Press the **Finish** button to complete the report and go directly to the Report Layout screen. (defaults will be supplied for any skipped steps)

Step 4 - Select Break Levels

The fourth and final step of the New Client Report Wizard is to choose when (at which break level) the break level calculations will print their results.



Select which break levels will have subtotals printed. You may also change the spacing for each break level.

Print Subtotals	Space After	Break Level Field	Description
☐		CUSNO	Customer number
✓		@TOTAL	Grand Totals

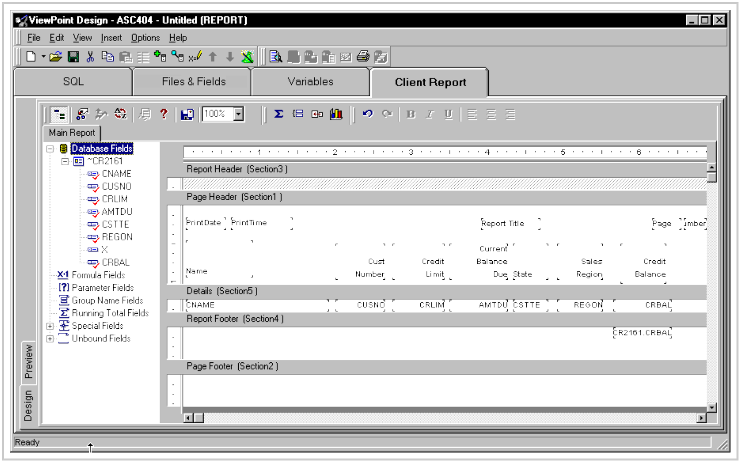
Press the **Cancel** button to exit the Wizard. Press the **Back** button to go to Step 3. Press the **Finish** button to complete the report and go to the Report Layout screen.

CRO Designer Display

The Client Report Designer is displayed whenever an existing report is opened for modification, or as the final step of the New Report Wizard.

The screen is divided into two major sections: The Available Fields and Functions Pane and the Editor Pane. There is also a second toolbar specific to CRO (underneath the standard ViewPoint toolbar) to provide access to various CRO report design functions.

The **Preview** and **Design** tabs make it easy to jump from the Report Designer to a quick 'preview' of the report results.



CRO Designer Toolbar Reference

Tool	Function
	Toggle the Field Selection Panel open and closed
	Open the Select Expert
	Specify the Sort Order
	Display Object Properties
?	Help

💾	Save to Crystal – (function not available)
100% ▾	Adjust the size of the display
Σ	Insert Summary
	Insert Group
	Insert Sub-report
📊	Insert Chart
↶	Undo
↷	Redo
B	Bold
I	Italics
U	Underline
	Align Left
	Align Center
	Align Right

Formatting

Formatting refers to changes in the layout and design of a report, as well as the appearance of text, objects, or entire report sections. This section details methods you can use to draw attention to data, change the presentation of dates, numbers, and other values, hide unwanted sections, and perform a variety of other formatting tasks to give a report a professional appearance.

Click the appropriate link to jump to that section:

- Formatting concepts
- Using the Report Design Environment
- Formatting properties
- Working with absolute formatting
- Working with conditional formatting

Formatting concepts

This section explains how to format a report. Formatting refers to changes you can make to the layout and design of a report, as well as the appearance of text, objects, or entire report sections.

You can use formatting to do many things, including:

- dividing sections of a report
- calling attention to certain data
- changing the presentation of dates, numbers, Boolean values, currency values, and text strings
- hiding unwanted sections
- giving the report a professional appearance.

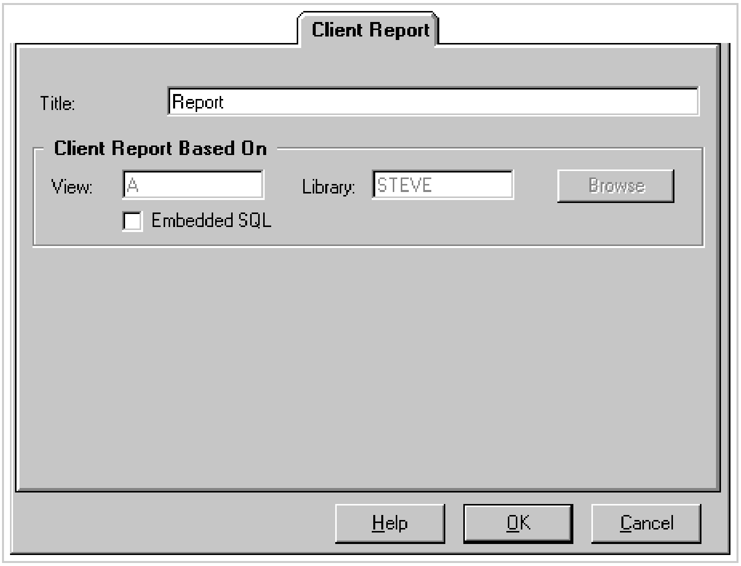
The following topics describe the types of formatting you can do with the Client Report Option, giving step-by-step instructions for performing a variety of formatting tasks.

NOTE:
There are many date formats you can choose to use on an English report, but if you send the report to a Japanese system, there may be some formatting irregularities. Not all English date formats are viewable on a Japanese system, and the same is true going from Japanese to English. For instance, abbreviated English months do not appear on a Japanese system and Japanese eras in short format do not appear on an English system.

Client Report Properties

Choose the Properties option from the Report Designer File menu, or the Properties option from the ViewPoint Explorer File menu to display report properties.

Client Report attributes will remain unchanged from your previous request for the report.



Options

Title - Specify a title or description for the report. This value will be used in the main ViewPoint Explorer (explorer).

Report Based On.. - A report can be based on a View or a Table. In design mode, this value is display only.

Browse Button - From the ViewPoint Explorer, the view or table the report is based on can be changed by typing over the values or by using the browse button to navigate to a different view.

Help Button - Display the online Help system.

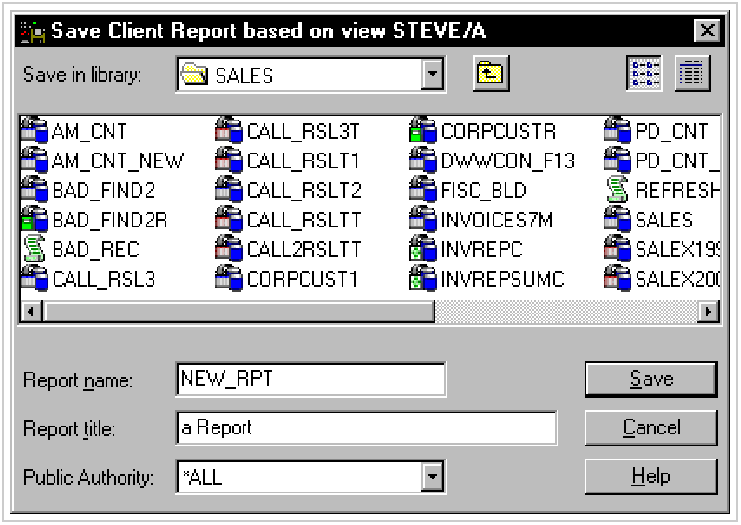
OK Button - Press OK to accept any changes.

Cancel Button - Press Cancel to exit this display without making changes.

Saving Client Report Definitions

The Save Client Report Based On dialog is displayed when the Save or Save As option is chosen from the Report Designer Screen.

Specify a name for the Report (or leave the name unchanged if saving an existing Report), a description or title, and a library. A message will inform you the Report was saved.



Using the Report Design Environment

Design solutions

There are several things to keep in mind when designing reports that are distributed to different environments. For the best results, consider:

- Section characteristics
- Making an object underlay a following section
- Pre-printed forms
- Hiding report sections
- Hiding report objects
- Placing text based objects
- Placing multi-line, text based objects
- Importing text based objects from a file
- Spacing between text based objects
- Overflow Field Representation
- Selecting multiple objects
- Vertical placement
- TrueType fonts
- Page margins
- Default printer
- Printer drivers.

Section characteristics

A report consists of several sections, including the Report Header, Page Header, Group Header, Details, Group Footer, Page Footer, and Report Footer.

Each report section is made up of a series of lines. When a text-based object is placed in a section, it is placed on a line in such a way that the text is aligned to the baseline. The line's height is then adjusted by the printer driver so that it is high enough to accommodate the object.

- if you place another text-based object on the same line with a font size larger than that of the first object, the line's height extends to accommodate the second object
- if you place another text-based object on the same line with a font size even larger than the previous two objects, the line's height extends to accommodate the third object.

A line's height is determined by the text-based object with the largest font size on the line.

As you add text-based objects to a report, either in the same section or other sections, the line height adjusts to accommodate the various fonts. Since this vertical spacing is set up by the printer driver, it is difficult to create reports designed for pre-printed forms when they are printed in various environments.

When designing reports, you should do the following:

- always print a test page
- keep all font sizes the same
- be sure to print pre-printed forms on the same machine.

Making an object underlay a following section

To make an object underlay a following section, first place the object in the section above the section you want it to underlay. Then select the Underlay Following Sections check box in the Section Expert for the section in which the object is placed.

The area in which the picture underlays depends on the following conditions:

- the size of the picture
- the section in which the picture was originally placed
- the position of the picture in the section.

By modifying size and placement of an object, you can create a variety of visual effects, using the underlay feature.

Pre-printed forms

If you print on pre-printed forms, you will be able to:

- scan a form
- place it in the report as a bitmap
- use the underlay feature to line up the bitmap and report, as well as move objects anywhere you want them to appear
- eliminate the need to print the forms separately by printing the report and the form as a single unit.

Adding a title page to the report

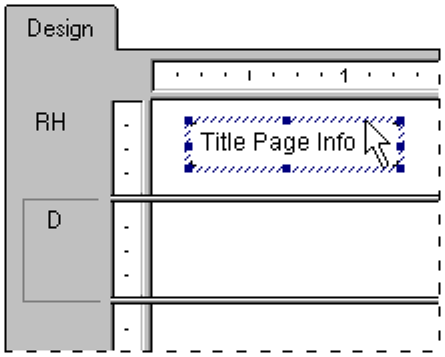
Crystal Reports provides a quick, easy way to add a title page to a report by selecting Report Title from the Special Fields in the Field Explorer dialog box. In order to use this field, you must have a title entered in the Summary tab of the Document Properties dialog box. See Adding summary information to the report.

To add a report title

1. On the Standard toolbar, click **Insert Fields.**

   The Field Explorer dialog box appears with "Database Fields" selected.

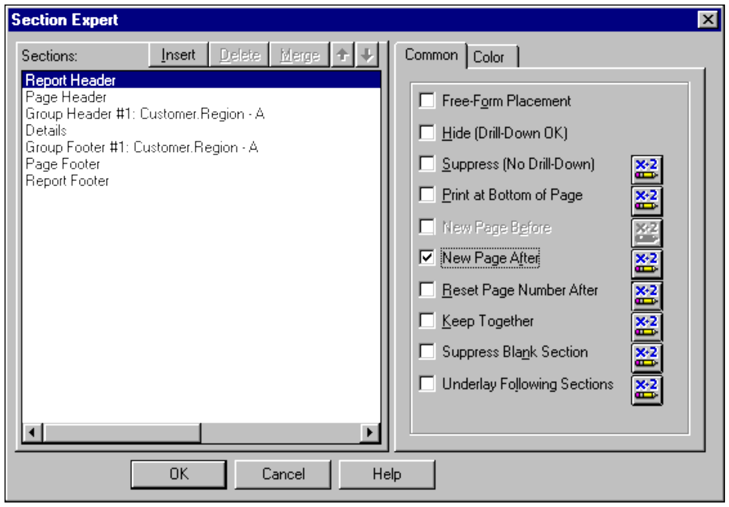
2. Scroll down to **Special Fields** and expand it by clicking.
3. Select Report Title, and click Insert to Report. An object frame appears when the cursor is moved over the report.



4. Move the object frame to the Report Header section and click once to place the frame.

   With the report title selected, click **Section Expert** on the Standard toolbar.

   The Section Expert appears.



Report Header
Page Header
Group Header #1: Customer.Region - A
Details
Group Footer #1: Customer.Region - A
Page Footer
Report Footer

- [ ] Free-Form Placement
- [ ] Hide (Drill-Down OK)
- [ ] Suppress (No Drill-Down)
- [ ] Print at Bottom of Page
- [ ] New Page Before
- [x] New Page After
- [ ] Reset Page Number After
- [ ] Keep Together
- [ ] Suppress Blank Section
- [ ] Underlay Following Sections

5. With the Report Header section highlighted, select the New Page After check box.

Now the title will appear on the first page and the report will begin on the second page.

Top N/Sort Group Expert

This Expert appears when you choose the Top N/Sort Group Expert command from the Report menu.

Use the Top N/Sort Group Expert to identify "top" groups (the best sales representatives in a sales report, the biggest customers in an order report, the states with the most customers in a customer report, and so on).

Use the Top N/Sort Group Expert to sort all of the records in a report or just the top or bottom groups of records. Depending on which option you choose the Expert appears with different options.

Group/Sort

Use this list to choose one of the following sorting options for your Top group:

- Sort All groups
- Sort the Top N group
- Sort the Bottom N group

based on

This list displays a list of summary fields in your report. Select the field you want to select Top N or Bottom N values for from this list. When Sort All groups is selected, use the scroll box below the based on list to see the fields chosen for sorting.

where N is

Enter the number of Top N or Bottom N values (where N is the number of values) you want to select into this edit box.

include Others, with the name

Select this option if you want to include all fields that do not fit into your Top N or Bottom N selection under a specified group name. Enter the group name you want all fields that do not fit into your Top N or Bottom N selection to appear under.

Delete

Click this button to delete the selected summary field. This is only available when Sort All Groups is selected.

Ascending/Descending

Select the Ascending option to sort the fields in ascending (A to Z, 1 to 9) order.

Select the Descending option to sort the fields in descending (Z to A, 9 to 1) order.

Hiding report sections

Crystal Reports has three properties you can set in the Section Expert to hide report sections.

Hide (Drill-Down OK)

The Hide property hides a section whenever you run the report. For example, in a summary report, the Hide property can be used to display only the summaries, but not the details behind the summaries. When the Hide property is applied to a section, it becomes visible when the

Drill-down cursor is used to drill down on the section contents. This property is absolute; it cannot be conditionally applied using a formula.

Suppress (No Drill-Down)

The Suppress property also hides a section when you run the report. Unlike the Hide property, however, you cannot apply the Suppress property, then drill down to reveal the section contents. This property can be applied absolutely, or conditionally using a formula. This is useful for writing form letters. For example, in a form letter, you might create two Details sections: one to suppress when sales are over $X and one to suppress when sales are under $X.

Suppress Blank Section

The Suppress Blank Section property hides a section whenever there is nothing in it. If something is placed within the section, then it becomes visible.

Hiding report objects

Crystal Reports has three formatting options in the Format Editor for hiding individual objects.

Suppress If Duplicated (Common tab)

The Suppress If Duplicated property prevents a field value from printing if it is identical to a duplicate of the value that comes immediately before it.

The value does not print, but the space in which it would have printed remains.

Related topics

- Suppress If Zero (Number tab)
- Suppress (Common tab).

Suppress If Zero (Number tab)

Tip: To find this option, click the Number tab of the Format Editor, then click the Customize button.

The Suppress If Zero property prevents a value from printing if it is a zero value. The value does not print, but the space in which it would have printed remains. To remove the blank space, select the Suppress Blank Section check box in the Section Expert.

Note: This will only work if there are no other objects in the section. Use the Section Expert to eliminate blank lines in this situation, then clear the Suppress Blank Section check box for the section that the field is in. This will eliminate the lines as long as there are no other objects in the section.

Suppress (Common tab)

The Suppress property hides an object when you run the report. For example, it is common to apply this property to formulas that are needed to do some report calculations, but that you do not want to print when you run the report. When you select this property, the selected object does not print.

Note: You can click the Conditional Formula button for any of these properties and create a formula that will make the setting conditional on some event. See Working with conditional formatting.

To set these properties, select the object, then click Format from the Supplementary toolbar to open the Format Editor dialog box. When the Format Editor appears, set the properties.

Placing text-based objects

When a text-based object is placed on a report, the object is represented by an object frame. The height of the object frame is based on the height of the font. The width, however, is determined differently, depending on the object you are working on.

- For database fields that are not memo fields, the width is initially determined by the width of the field as defined in the database, and by the average character width as provided by the selected font and font size.

  For example, you have a database field called {customer.LAST NAME} and your database defines this field as a text field with a length of 35 characters. When you place this field on your report, the width of the boundary is 35 times the average character width of the font and font size that the database field is formatted to. Remember that this is the initial default boundary width. The width can always be resized to increase or decrease as you see fit.

- For text-based objects, the default width is approximately 17 average character widths wide. Objects are different from database fields in that their width automatically expands as you enter in text and/or database fields into the object. As with all other text-based objects, the width can be resized by the user.
- For different number fields (double, single, integer, long integer, and byte) the default widths are all different. As with all text-based objects, the width can be resized by the user.

Related topics

- Preventing the truncation of text inside an object
- Preventing breaks in non-spacing text inside an object.

Preventing the truncation of text inside an object

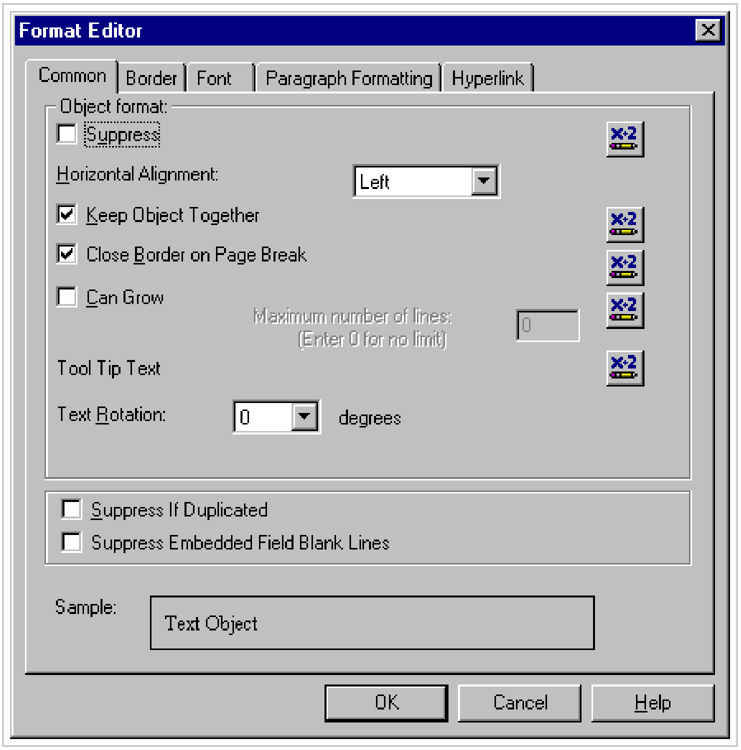
Whether the default widths are accepted or the text-based objects are resized, a problem could arise if the text inside the object prints right to the edge of the object frame. While the report may look fine on the machine it was designed on, when the report is printed using another printer driver that measures the font wider, the length of the text grows, but the object frame remains fixed. The resulting text is cut-off or truncated.

To prevent the truncation of text inside an object

1. Right-click the text object you want to format to bring up the shortcut menu.

   Tip: Another way to do this is to click the Object Properties button on the Supplementary toolbar.

2. On the shortcut menu, click Format.
   The Format Editor dialog box appears.



Object format:

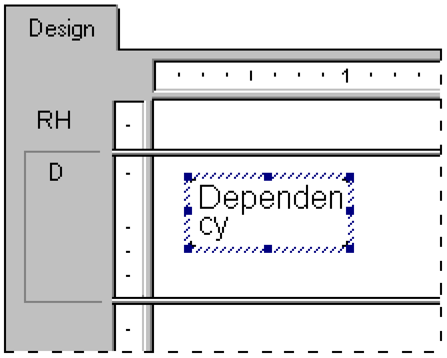
- [ ] Suppress
- Horizontal Alignment: Left
- [✓] Keep Object Together
- [✓] Close Border on Page Break
- [ ] Can Grow
- Maximum number of lines: (Enter 0 for no limit) 0
- Tool Tip Text
- Text Rotation: 0 degrees
- [ ] Suppress If Duplicated
- [ ] Suppress Embedded Field Blank Lines
- Sample: Text Object

3. On the **Common** tab, select the **Can Grow** check box.
4. Click OK to save your changes.

   The object is then formatted to print on multiple lines. If the text prints wider than the object, the text wraps onto additional lines.

Preventing breaks in non-spacing text inside an object

For text strings that do not contain spaces, such as single words, the text string is broken at the edge of the object frame before the line starts to wrap.



To prevent the breaks in non-spacing text inside an object

1. Select the object you want to format.
2. Expand the object frame to make it wider than the widest block of text inside the frame.

There are many times when the actual text in a database field is far less than the maximum amount the field can contain. For example, a {table.LAST NAME} field is designed with a field size of 80 and the longest name in the database is 28 characters. In this case, when you first place the field in your report, the field is 80 times the average character width. Reduce the width of the field, but include enough space to account for growth.

While each of these options offers an effective solution when dealing with a single text-based object in a section, there are still design considerations to take into account when placing more than one object in a section. When sizing an object, consider its placement with regard to other objects in the section.

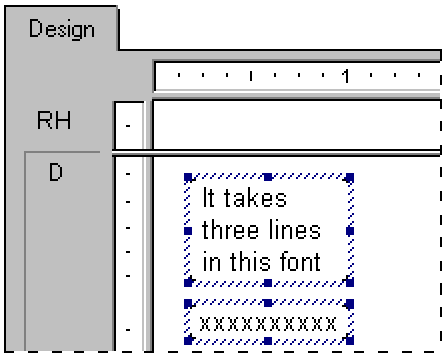
Avoid designing reports where the space between each object is very tight. Leave room for growth by expanding the width of the object by approximately 5 per cent. Or, if this is not possible, consider reducing the size of the font.

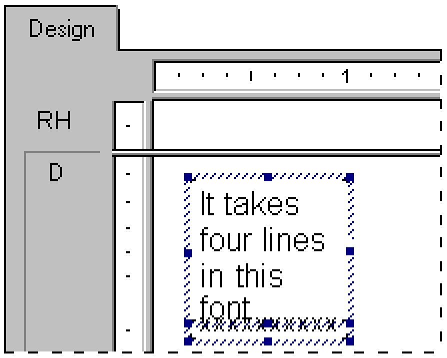
Placing multi-line, text-based objects

While text-based objects that are formatted to print on multiple lines follow the same design rules as other objects, they have one additional characteristic to be considered. If the printer driver expands or contracts the spacing of the text, word wrapping may differ, changing the number of lines necessary to print the object in order to accommodate growth or shrinkage.

When placing multi-line, text-based objects, you could encounter problems if other objects in the same section are placed directly below them.

Unlike single-line, text-based objects, expanding the object frame of a multi-line, text-based object to accommodate growth is not a viable option. When you do this, the line width increases according to the expanded boundaries.





So, when possible, place multi-line, text-based objects at the bottom of a section. If they require more lines to print, the section expands downward to accommodate the growth, and they do not endanger other objects.

Importing text-based objects from a file

Using Crystal Reports, you can import a formatted, text-based object from an existing file onto your report.

To import text-based objects from a file

1. Double-click the text-based object you want to format, then Right-click it to bring up the shortcut menu.
2. On the shortcut menu, click **Insert from file**.
3. In the Open dialog box that appears, select the file in which your text-based object is stored, then click Open.

   The object is imported from the file onto your report, replacing the object you selected to format in Step 1.

Spacing between text-based objects

Use the grid and guidelines options to help evenly align text-based objects.

You can select the Snap To Grid option, set the grid to a maximum of one inch, and make the grid visible or invisible on the Design tab, Preview tab, or both. For more information on working with grids, see Using the grid below.

You can also work without a grid, placing objects wherever you want them on a report. You may want to work in a free-form environment while retaining the ability to align objects, or to move or resize them as a group.

Using the grid

The grid is a series of row and column coordinates. When the grid is selected, Crystal Reports enables you to place text-based objects only at these coordinates, not between them. You can then space data on your report and align objects as needed. If you attempt to place an object between grid coordinates, the object "snaps" to the grid; that is, the object automatically moves to the nearest set of row and column coordinates.

Each report section contains a design grid. You can select the grid on or off, as well as set it to different sizes when required. By default, the grid is not selected. See Selecting the grid.

Once set, the grid remains the same size for all sections. It is measured from the upper-left corner of each section and continues down and to the right, until the end of the section. A new grid of the same size then starts from the upper-left corner of the next section, and so on, through the end of the report.

If you select the Snap To Grid option, the following conditions occur:

- The upper-left corner of all newly placed, text-based and OLE objects snap to a grid point.
- Objects placed on a report before the Snap To Grid option is selected do not snap to the nearest grid point. They remain in the same place.

- If you resize an object, the side or sides that you are resizing snap to the nearest grid point.

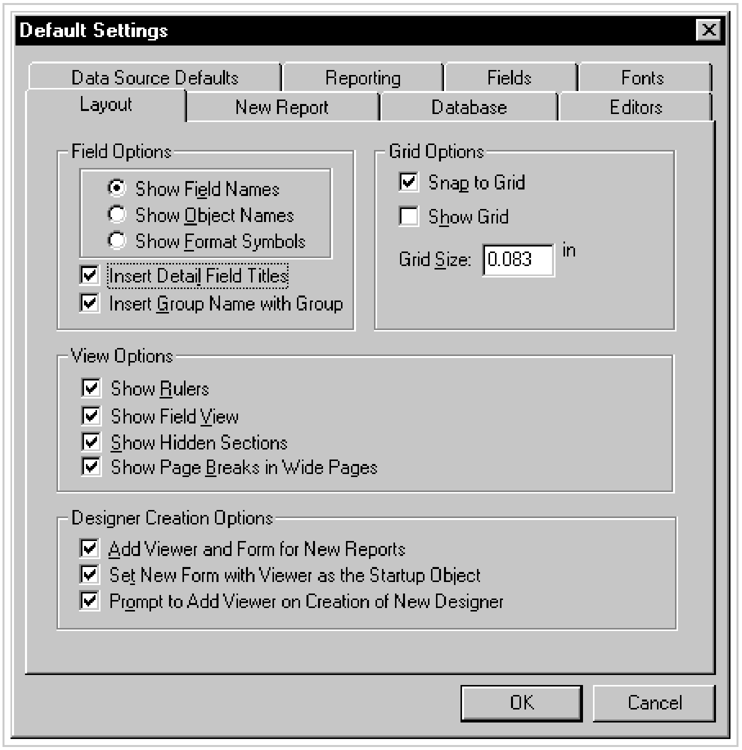
Selecting the grid

The Design and Preview tabs have an underlying grid structure that you can activate on the Layout tab in the Default Settings dialog box.

To select the grid

1. Right-click in the editor. From the drop-down menu, select **Designer=>Grid**. To change grid options click **Designer=>Default Settings=>Layout Tab**.

   The Default Settings dialog box appears.



Default Settings

Data Source Defaults | Reporting | Fields | Fonts | Layout | New Report | Database | Editors

Field Options
- [✓] Show Field Names
- Show Object Names
- Show Format Symbols
- [✓] Insert Detail Field Titles
- [✓] Insert Group Name with Group

Grid Options
- [✓] Snap to Grid
- [ ] Show Grid
- Grid Size: 0.083 in

View Options
- [✓] Show Rulers
- [✓] Show Field View
- [✓] Show Hidden Sections
- [✓] Show Page Breaks in Wide Pages

Designer Creation Options
- [✓] Add Viewer and Form for New Reports
- [✓] Set New Form with Viewer as the Startup Object
- [✓] Prompt to Add Viewer on Creation of New Designer

2. On the **Layout** tab, in the Grid options area, you can show the underlying grid structure on the Design or Preview tab, or both, and specify grid size.
3. Click **OK** to save your changes.

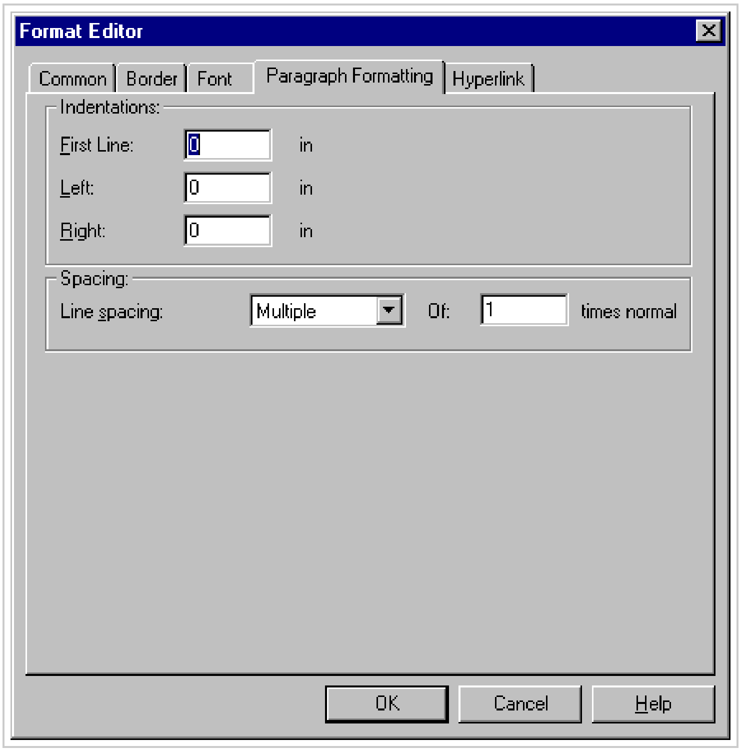
Indenting lines

Using Crystal Reports, you can control line indentation for memo fields, string fields, and text-based objects. For objects, you have the option of indenting lines for a particular paragraph by positioning the cursor at the start of that paragraph. Or, if you select an object in its entirety, you can apply the same indenting specifications to all the paragraphs within that object.

Keep in mind that any line following a carriage return will be considered the first line of a new paragraph.

To indent lines

1. Right-click the field or object you want to format to bring up the shortcut menu.
2. On the shortcut menu, click Paragraph Formatting. The Format Editor dialog box appears with the Paragraph Formatting tab open.



Indentations:

First Line: 0 in

Left: 0 in

Right: 0 in

Spacing:

Line spacing: Multiple Of: 1 times normal

3. In the Indentations area, you can indent the first line of the paragraph; indent every paragraph line from the left margin; and indent every paragraph line from the right margin.

   Note: Only indentation values within the range of the field or object width are accepted.

4. Click OK to save your changes.

Overflow Field Representation

Crystal Reports uses Overflow Field Representation to assist users when working with numeric or currency values in report cells. Normally, if a numeric or currency value is larger than the field containing it, that value is truncated, or "clipped." For example, values like 100,000,000 might appear on the report as 1,000, or as 000 (depending on the properties you have set). This could potentially cause confusion when the report is read.

When the Allow Field Clipping option is cleared, numeric/currency field values that exceed the field size will be represented by number signs (######) in the Preview tab, letting you know immediately when the field is too small.

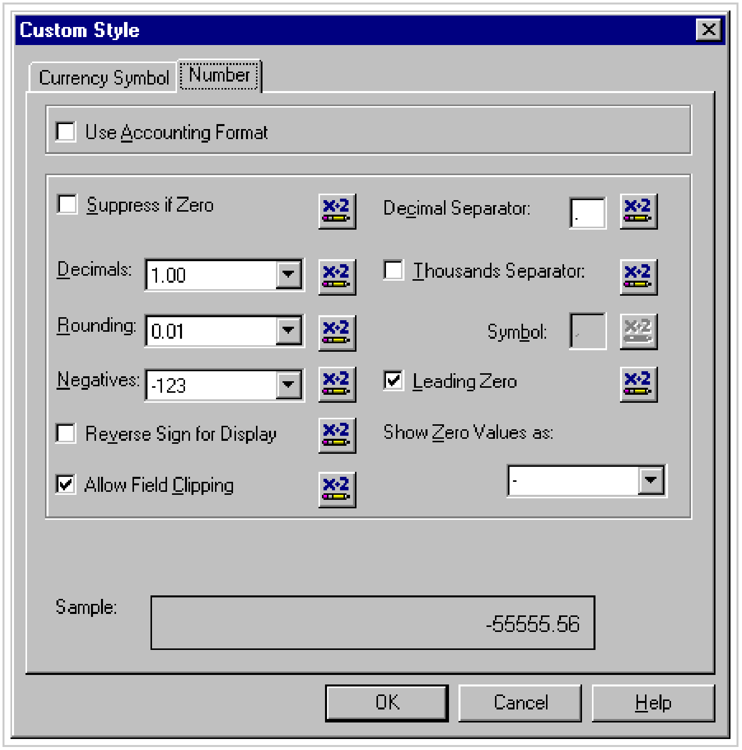
To allow for overflow field representation

1. Right-click the currency field or number field you want to format to bring up the shortcut menu.
2. On the shortcut menu, click Format Field.

The Format Editor dialog box appears with the Number tab open.

3. Click the **Customize** button.

   The Custom Style dialog box appears with the Number tab open.



Custom Style

Currency Symbol | Number

☐ Use Accounting Format

☐ Suppress if Zero

Decimals: 1.00

Rounding: 0.01

Negatives: -123

☐ Reverse Sign for Display

☑ Allow Field Clipping

Decimal Separator: .

☐ Thousands Separator:

Symbol: ,

☑ Leading Zero

Show Zero Values as: -

Sample: -55555.56

4. To allow overflow field representation, clear the Allow Field Clipping check box.

   Note: You also have the option to click the Format button to enter a formula in the Format Formula Editor. In the Format Formula Editor, you can specify that field clipping will be disabled only when certain conditions are met.

5. Click **OK** to save your changes.

To view the results, refresh the report. If you disabled field clipping, any numeric/currency field values that are larger than the fields containing them will be represented by number signs (######).

Selecting multiple objects

You can select multiple objects, including text, field, chart, map, bitmap, OLAP grid, Cross-tab and OLE objects, to format them together.

Once you have selected multiple objects, you can move, align, size, cut, and copy and paste them as a group. You can also change their font, color, and paragraph style.

Objects are moved, aligned, and sized based on a "main" object, which is usually the last object you select. You can change the main object to another by right-clicking the desired object.

Note: When moving multiple objects, if the new location does not accommodate all of the selected objects, but can accommodate the main object, then the main object and those objects that can be accommodated will be moved. The other objects will remain in their original position.

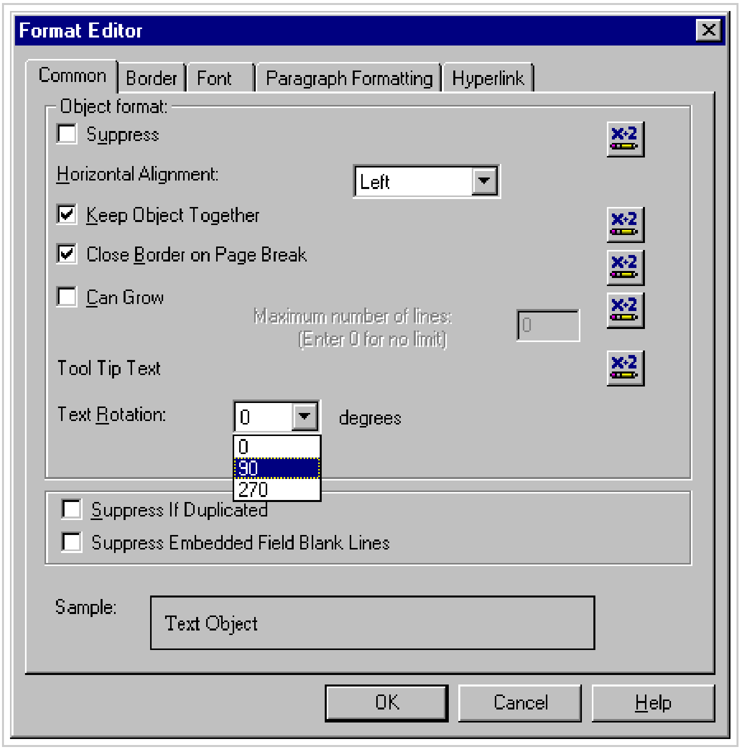
To select multiple objects

1. Click one object, and Ctrl-click the other objects you want to select.
2. Right-click the main object.
3. On the short-cut menu, select the appropriate formatting option.

Vertical placement

On the Common tab of the Format Editor, you can use the text rotation options to vertically align the fields and text-based objects on your report.

When you select a text rotation of 90 degrees, the text shifts 90 degrees in a counter-clockwise direction.


Format Editor

Common | Border | Font | Paragraph Formatting | Hyperlink

Object format:

[ ] Suppress

Horizontal Alignment: Left

[✓] Keep Object Together

[✓] Close Border on Page Break

[ ] Can Grow

Maximum number of lines: (Enter 0 for no limit) 0

Tool Tip Text

Text Rotation: 0 degrees

0
90
270

[ ] Suppress If Duplicated

[ ] Suppress Embedded Field Blank Lines

Sample: Text Object

OK | Cancel | Help

When you select a text rotation of 270 degrees, the text shifts 270 degrees in a counter-clockwise direction.

Note:

- If text rotation is left at 0 degrees, your report is horizontally formatted, left to right.
- For text rotation of text-based objects, the Can Grow option that prevents the truncation of text inside an object is automatically cleared. For more information on the Can Grow option, see Preventing the truncation of text inside an object.
- Vertically formatted text that spans over the edge of the page cannot be displayed as part of your report.

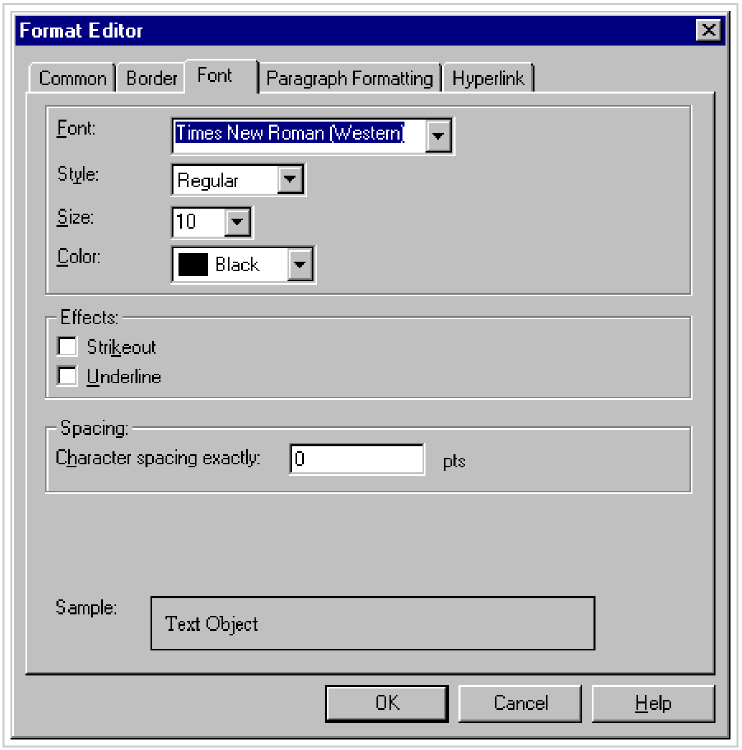
Inserting character and line spacing

With Crystal Reports, you can specify the amount of spacing between characters or lines for memo fields, string fields, and text-based objects.

To insert character and line spacing

1. Right-click the field or object you want to format to bring up the shortcut menu.
2. On the shortcut menu, click Font.

   The Format Editor dialog box appears with the Font tab open.

   Font: Times New Roman (Western)
   Style: Regular
   Size: 10
   Color: Black
   Effects:
   Strikeout
   Underline
   Spacing:
   Character spacing exactly: 0 pts
   Sample: Text Object

   You will use this tab to set up the character spacing values.

3. In the Spacing area, in the **Character spacing exactly** field, specify the value *n* that each character occupies.

   The value n is defined as the distance in number of points measured from the start of one character to the start of the next. When you change the character spacing, you change only the spacing between adjacent characters, not the font size of the characters.

   For example, if you specify a 14-point font with a character spacing of 14 points, each character will remain as a 14-point font size, occupying a space that is 14 points wide.

4. Click the Paragraph Formatting tab.

   You will use this tab to set up the line spacing values.

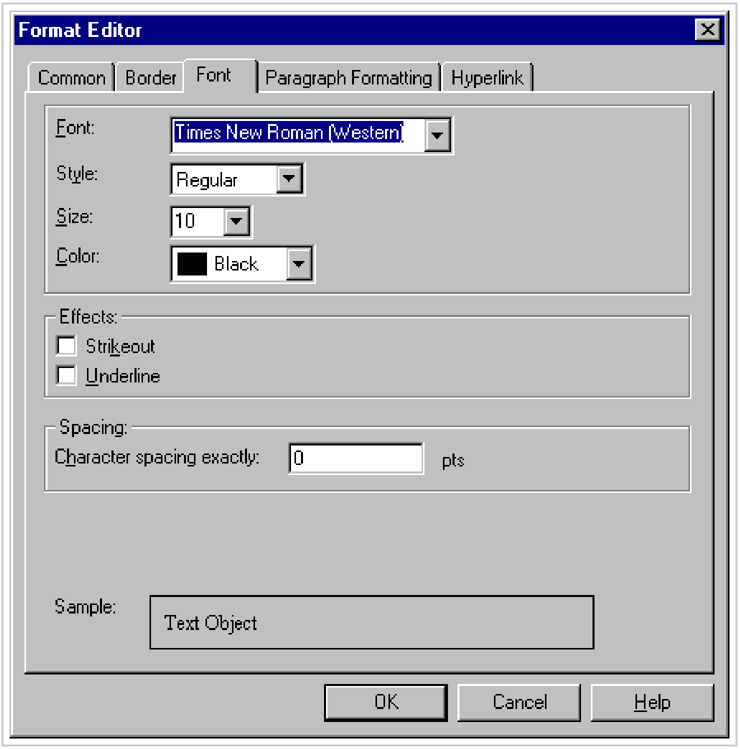
5. In the Spacing area, in the **Line spacing** field, specify the line spacing as a multiple of the font size you are using, or as an exact number of points.
6. Click OK to save your changes.

Setting fractional font sizes

On the Font tab of the Format Editor, you can select a fractional font size for database fields and text-based objects on your report.

To set fractional font sizes

1. Right-click the field or object you want to format to bring up the shortcut menu.
2. On the shortcut menu, click Font. The Format Editor dialog box appears with the Font tab open.



Font: Times New Roman (Western)
Style: Regular
Size: 10
Color: Black
Effects:
☐ Strikeout
☐ Underline
Spacing:
Character spacing exactly: 0 pts
Sample: Text Object

3. In the **Size** list, type the desired fractional font size for the field or object.

   Note: The number you type must be between 1 and 1638. Crystal Reports will automatically round all fractional entries to the nearest 0.5. Consequently, in your report, you can use the fractional font sizes 1.5, 2.5, 3.5, and so on, up to 1637.5.

4. Click OK to save your changes.

Note: When setting fractional font sizes for individual database fields and text-based objects that you've already placed on your report, you must make your changes manually—that is, by following these procedures. (This is because the existing font settings of objects in your report will override your default Options.) However, you can use the Designer=>Default Settings=> Fonts tab to adjust your default font settings: these default Options will affect the new reports that you create, along with any new objects that you add to an existing report.

TrueType fonts

Designing your report with printer-specific fonts can lead to problems when using different printers. The fonts may not be supported by the other printers, or if they are supported, the fonts may not be installed on the printers.

During the printing process, if you encounter printer-specific fonts that are unrecognizable to the printer driver, Crystal Reports substitutes the fonts, creating inconsistent results. To avoid this situation, only common TrueType fonts should be used when designing reports.

Page margins

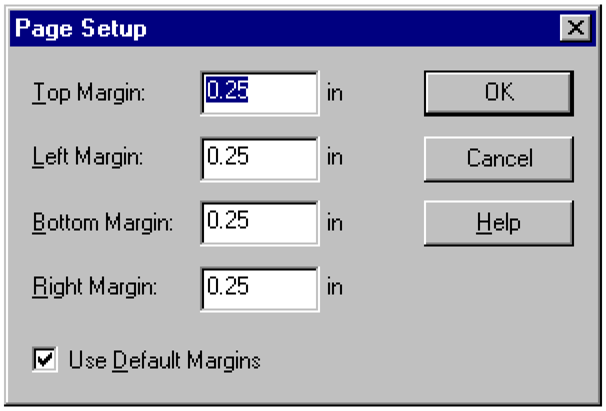
Setting specific margins

Crystal Reports gives you the option of setting margins to meet your specifications. Also see Using default margins.

To set specific margins

1. On the **Designer** menu, click **Page Setup**.

   The Page Setup dialog box appears.



Page Setup

Top Margin: 0.25 in

Left Margin: 0.25 in

Bottom Margin: 0.25 in

Right Margin: 0.25 in

☑ Use Default Margins

2. Change the default page margins to fit your needs.
3. Click **OK** to save your changes.

Note: All margins are calculated from the paper edge. Thus, a left margin of .25 inches causes the printing to start exactly one quarter inch in from the edge of the paper.

Using default margins

If you decide to design your report using the default margins, the following problems may occur.

- When printing a report in another environment where the printer's default margins are *greater* than its setting, the report objects on the right side of the report print off the page.
- When printing a report in another environment where the printer's default margins are *smaller* (allowing a larger printing area), the entire report moves to the left side of the page.

It is recommended that you always set your own margins. Even if the margins you want to use are the same as the default margins, be sure that the Use Default Margins option in the Page Setup dialog box is not selected, and you set your margins manually using Page Setup.

Default printer

In general, it is a good idea not to choose a specific printer. Even though the printer may be identical to the default printer, how the printer is recognized can still vary for different operating systems.

For example, an HP Laser III printer is being installed on three different operating systems.

- With Microsoft Windows 95 and 98, the printer name can be changed so that HP Laser III is Front Reception Printer, but the printer driver will be listed as HPPCL5MS.DRV.
- With Microsoft Windows NT, the printer name is also referenced and can be changed by the user, but the printer driver is always WINSPOOL.

If you select a specific printer, Crystal Reports looks for that printer by name. If the printer you selected cannot be found, the default printer is chosen, resulting in the possibility of printing inconsistencies.

When selecting a specific printer, such as a label printer or a printer dedicated to printing invoices, the printer name must be the same as the name of the printer the report was designed on. Be aware that anyone printing the report must use that same printer or they could encounter problems.

Also see Setting page orientation and paper size.

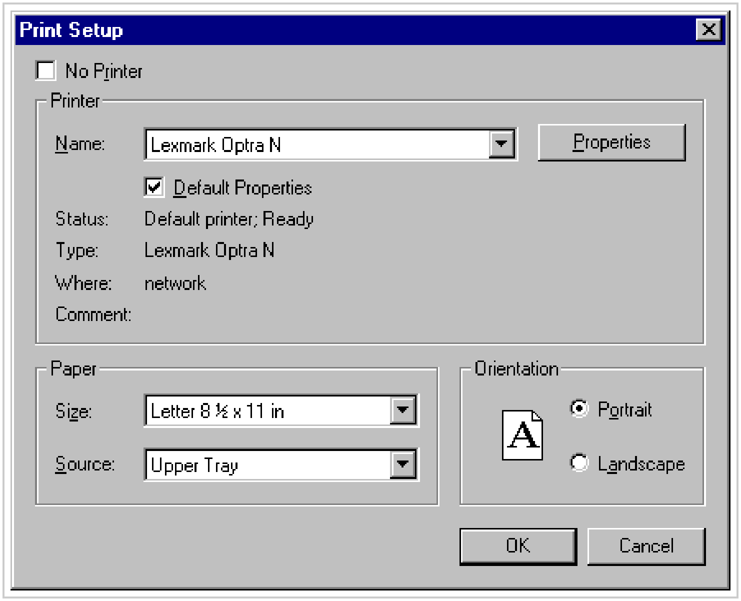
Setting page orientation and paper size

You can print your reports using either portrait or landscape orientation, and in a variety of paper sizes. You can specify these options using the **Printer Setup** command from the Designer menu.

To set page orientation and paper size

1. On the **Designer** menu, click **Printer Setup**.

   The Print Setup dialog box appears.



Print Setup

☐ No Printer

Printer

Name: Lexmark Optra N

☑ Default Properties

Status: Default printer; Ready

Type: Lexmark Optra N

Where: network

Comment:

Paper

Size: Letter 8 ½ x 11 in

Source: Upper Tray

Orientation

[✓ Portrait] Landscape

2. In the Printer area, in the **Name** list, select the printer you want to use, if it is not already the default printer.

   In the Paper area, in the **Size** and **Source** lists, select the desired paper.

   The paper size options are directly related to the printer you select. For example, the HP LaserJet driver (PCL) offers a choice of letter, legal, executive, or A4 paper sizes, whereas the PostScript printer driver lets you choose from letter, legal, note, A4, B5, letter small, and A4 small paper sizes.
3. In the Orientation area, click **Portrait** or **Landscape**.
4. Click OK to save your changes.

Printer drivers

Updating printer drivers

In order to maintain performance, Crystal Reports queries the printer driver for each of the font elements (font metrics), such as average character height, character width, height of the ascenders and descenders, internal leading, and so on. A problem may develop if using an older printer driver that does not return the font metrics accurately. If you are experiencing problems when printing (missing fields, incorrect formatting, and so on), it is recommended that you obtain and install the most recently updated drivers for your printer. In many cases, the newer printer drivers provide accurate font metrics and any printing issues are quickly resolved.

For details on potential printing issues, see Inconsistencies due to printer drivers.

Inconsistencies due to printer drivers

When printing, inconsistencies may occur if different printer drivers are used to create and print your reports. These inconsistencies are a result of the various methods that individual printer drivers use to measure text metrics such as font size. When printed, text-based objects may be misaligned, truncated, or overprint each other. Examples of text-based objects include string or character fields, text objects, memo fields, numeric fields, and formula fields.

Problems such as these may arise when you have:

- two identical printers, but each one is using a different printer driver
- two different printers using the same printer driver
- two different printers using different printer drivers
- one printer driver that uses the TrueType font and a second printer driver that maps TrueType fonts to PostScript fonts
- two identical printers using the same printer driver, but each one is printing from a different version of Microsoft Windows
- two identical printers using the same printer driver, but the printer drivers are different versions
- two identical printers, two identical printer drivers, and two identical operating systems, but the resolution of the video drivers is different.

Therefore, while a document using one printer driver may require six full lines to display a block of text:

- using a second printer driver that measures fonts narrower could result in the same block of text requiring less than six full lines
- using a third printer driver that measures fonts wider could require more than six full lines.

For the most part, this situation cannot be avoided. The goal of the report distributor is to design reports that accommodate printer driver dependency and still print consistently using different printer drivers. To do this, Crystal Reports provides several design solutions. If taken into account when creating your report, these solutions can ensure proper printing and distribution for your report in almost any environment.

Formatting properties

You can set formatting properties using the Format Editor for objects and the Section Expert for report sections. In most cases, you can set one of two types of properties:

- absolute (always apply the property)
- conditional (apply the property only if certain criteria are met).

For more information, see Working with absolute formatting and Working with conditional formatting.

Working with absolute formatting

Absolute formatting is formatting that applies under any condition. This type of formatting property always follows a *select*, *then apply* procedure. For example, you *select* what it is that you want to format (object or section), *then* you *apply* the formatting to the selection using property settings.

You can use the following dialog boxes to format your reports:

- Format Editor to format field values
- Section Expert to format entire sections
- Highlighting Expert to conditionally format currency and number fields.

Each of these dialog boxes contains a number of different formatting properties, as well as the tools for turning the properties on or off and specifying attributes.

Click the appropriate link to jump to that section:

- Adding borders, color, and shading to a field
- Changing your default field formats
- Adding and editing lines
- Adding and editing boxes
- Adding shapes to a report
- Using conventional accounting formats
- Using white space between rows.

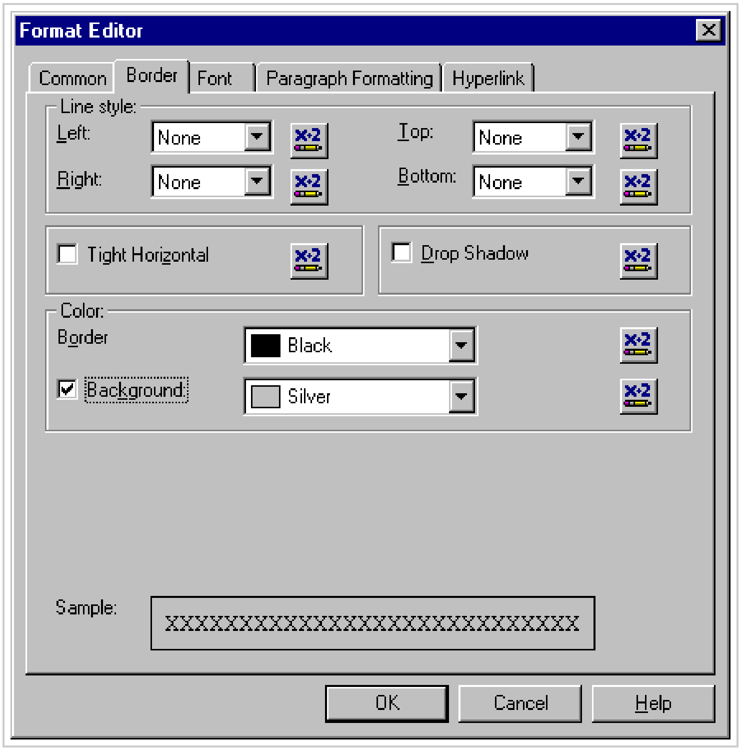
Adding borders, color, and shading to a field

Crystal Reports allows you to add borders, color, and shading to fields on your report in order to emphasize important data and create professional-looking reports.

To add borders, color, and shading to a field

1. Right-click the field you want to format and select Format to bring up the shortcut menu.
2. Click the Border Tab.

   The Format Editor dialog box appears with the Border tab open.



Format Editor

Line style: Left: None; Right: None; Top: None; Bottom: None

☐ Tight Horizontal
☐ Drop Shadow

Color: Border: Black
☑ Background: Silver

Sample: XXXXXXXXXXXXXXXXXXXXXXXXXXXXX

3. Select the line style, color, and background color of the field.
4. Click OK to save your changes.

Changing your default field formats

Crystal Reports allows you to display database fields in almost any format on your report. This section describes how to use the Default Settings command to control the default format settings that Crystal Reports uses when you add a field to any report. In the Default Settings dialog box, you can set the default formats for database fields of the following type: String, Number, Currency, Date, Time, Date/Time, and Boolean.

Note: When you change default field formats, your new settings affect only the objects that you subsequently add to a report. To format fields that you've already added to a report, you must right-click the field in the report and select Format from the shortcut menu.

To specify default formats for fields

1. On the **Designer** menu, click **Default Settings**.
2. In the Default Settings dialog box, click the **Fields** tab.
3. Click the button appropriate to the type of field you want to format (String, Number, Currency, Date, Time, Date/Time, or Boolean).

   The Format Editor appears.

4. Use the Format Editor's tabs to specify the formats you want.
5. Click OK.

For a specific example, see Setting default formats for Date, Time, and Date/Time fields.

Changing your default field formats

Crystal Reports allows you to display database fields in almost any format on your report. This section describes how to use the Default Settings command to control the default format settings that Crystal Reports uses when you add a field to any report. In the Default Settings dialog box, you can set the default formats for database fields of the following type: String, Number, Currency, Date, Time, Date/Time, and Boolean.

Note: When you change default field formats, your new settings affect only the objects that you subsequently add to a report. To format fields that you've already added to a report, you must right-click the field in the report and select Format from the shortcut menu.

To specify default formats for fields

1. On the **Designer** menu, click **Default Settings**.
2. In the Default Settings dialog box, click the **Fields** tab.
3. Click the button appropriate to the type of field you want to format (String, Number, Currency, Date, Time, Date/Time, or Boolean).

   The Format Editor appears.

4. Use the Format Editor's tabs to specify the formats you want.
5. Click OK.

For a specific example, see Setting default formats for Date, Time, and Date/Time fields.

Setting default formats for Date, Time, and Date/Time fields

The following procedures first describe how to specify standard formats for Date, Time, and Date/Time fields, and then describe how to customize the formats for such fields.

Note: These default settings will affect only the objects that you subsequently add to a report. To format fields that you've already added to a report, you must right-click the field in the report and select Format from the shortcut menu.

To set standard default formats for Date, Time, and Date/Time fields

1. On the **Designer menu**, click **Default Settings**.
2. ">In the Default Settings dialog box, click the **Fields** tab.
3. To open the Format Editor, click the button appropriate to the field you want to format (**Date, Time, or Date/Time**).

Note: If you click the Date/Time button in the Format Editor, then any subsequent changes will affect Date/Time fields only. You must click Date or Time to format independent date fields or time fields.

4. ">In the Format Editor dialog box, click the **Date / Time** tab.
5. Select a predefined format from the list (or click **Customize** to create your preferred format). When you click a new format, you can preview the results in the Sample area of the Format Editor.

   Note: From the list of predefined formats, you can choose the System Default settings to ensure that Crystal Reports uses the formats dictated by Windows. You can alter your system's settings in the Regional Settings Properties dialog box, located in the Control Panel.

6. Once you've selected a format, click **OK** in the Format Editor dialog box.
7. Click **OK** in the Default Settings dialog box.

Now, when you add Date, Time, or Date/Time fields to a report, Crystal Reports should use the format you specified.

To customize formats for Date, Time, and Date/Time fields

1. On the **Designer menu**, click **Default Settings**.
2. In the Default Settings dialog box, click the **Fields** tab.
3. Open the Format Editor by clicking the button appropriate to the field you want to format (**Date, Time, or Date/Time**).
4. In the Format Editor dialog box, click the **Date / Time** tab.
5. Click **Customize** to open the Custom Style dialog box.

   Note: If you chose to format Date/Time fields at Step 3, then you will see three tabs in the Custom Style dialog box (Date/Time, Date, and Time). The formats specified in these tabs apply only to the two elements of Date/Time fields, and will not affect the formats specified for independent date fields or time fields.

6. Create your preferred format by adjusting the various options in the Custom Style dialog box.
7. Once you've finished designing your format, click **OK** in the Custom Style dialog box.
8. Click **OK** in the Format Editor dialog box.
9. To format another type of field, click the appropriate button in the Default Settings dialog box. Otherwise, click **OK** to return to Crystal Reports.

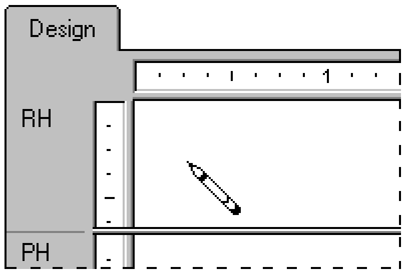
Now, when you add Date, Time, or Date/Time fields to your reports, Crystal Reports should use the customized format that you created.

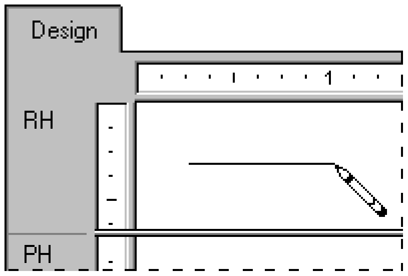
Adding and editing lines

Crystal Reports allows you to add lines to a report to emphasize important data and create professional-looking reports.

To add lines to a report

1. From the Insert menu, , click **Line**.



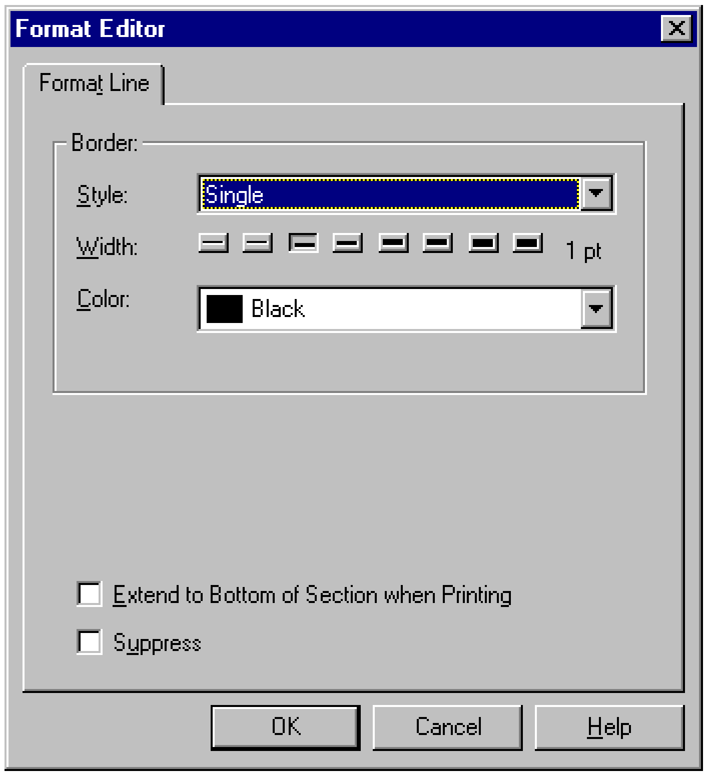


2. Use the pencil cursor to draw the line where desired.

To edit lines on a report

1. Right-click the line you want to format to bring up the shortcut menu.
2. On the shortcut menu, click **Format**.

   The Format Editor dialog box appears.



Border:
Style: Single
Width: 1 pt
Color: Black
Extend to Bottom of Section when Printing
Suppress

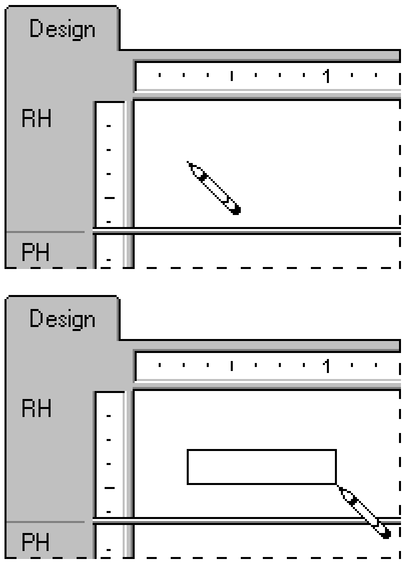
3. On the **Format Line** tab, make the desired changes to the line.
4. Click **OK** to save your changes.

Adding and editing boxes

Crystal Reports allows you to add boxes to a report to emphasize important data and create professional-looking reports.

To add boxes to a report

1. From the Insert menu, click **Box**.

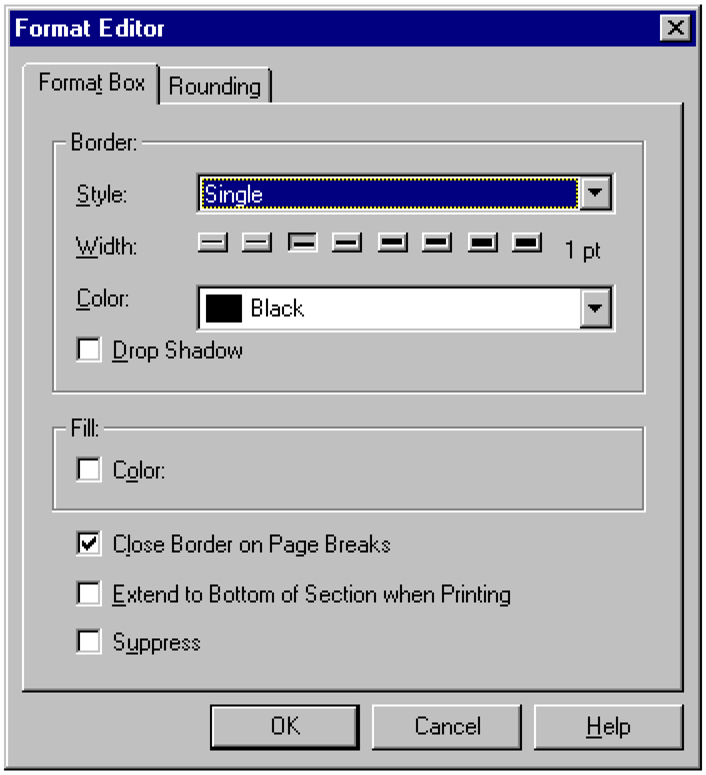


2. Use the pencil cursor to draw the box where desired.

To edit boxes on a report

1. Right-click the box you want to format to bring up the shortcut menu.
2. On the shortcut menu, click **Format**.

   The Format Editor dialog box appears.



Border:

Style: Single

Width: 1 pt

Color: Black

☐ Drop Shadow

Fill:

☐ Color:

☑ Close Border on Page Breaks

☐ Extend to Bottom of Section when Printing

☐ Suppress

3. On the **Format Box** tab, make the desired changes to the box.
4. Click **OK** to save your changes.

Adding shapes to a report

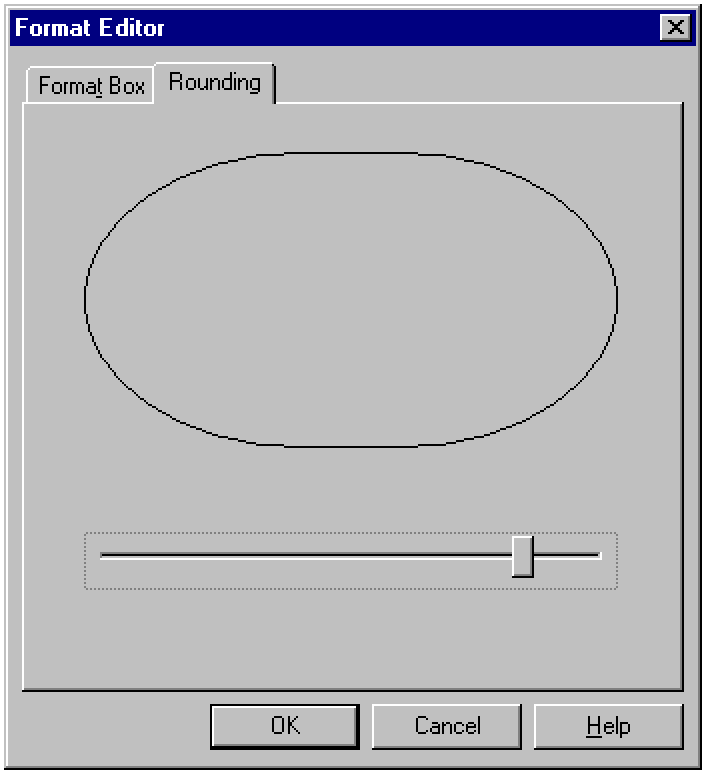
When designing report formats in Crystal Reports, you can insert a variety of shapes such as circles, eclipses, and boxes with rounded corners, as part of your report. This is especially useful for formatting reports in languages that require these shapes to effectively communicate.

To add shapes to a report

1. Add a box to your report.

   See Adding and editing boxes.

2. Right-click the box to bring up the shortcut menu.
3. On the shortcut menu, click **Format**.
4. In the Format Editor that appears, click the **Rounding** tab.



5. Move the slider to the right to increase the curvature of the box corners.

   The box that you started with gradually changes to an ellipse or circle, depending on how far you move the slider to the right.

6. Once the appropriate shape is created, click **OK** to save your changes.

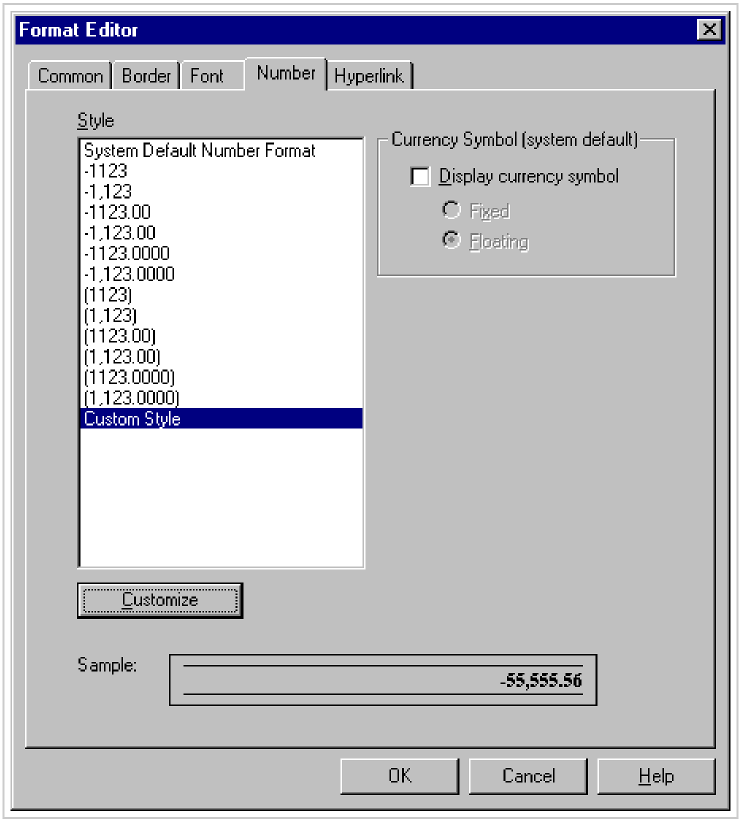
Using conventional accounting formats

As a way of supporting the conventions used in the accounting profession, Crystal Reports lets you decide on how to display the currency symbol, negative values, and zero values on your financial reports. You can also set up your report to reverse the signs for credit and debit amounts.

To use accounting conventions in a report

1. Right-click the currency field or number field you want to format to bring up the shortcut menu.
2. On the shortcut menu, click **Format.**

   The Format Editor dialog box appears with the Number tab open.


Format Editor

Common | Border | Font | Number | Hyperlink

Style

System Default Number Format
-1123
-1,123
-1123.00
-1,123.00
-1123.0000
-1,123.0000
(1123)
(1,123)
(1123.00)
(1,123.00)
(1123.0000)
(1,123.0000)
Custom Style

Currency Symbol (system default)

Display currency symbol

Fixed

Floating

Customize

Sample: -55,555.56

OK | Cancel | Help

3. In the Style area, select how you want the system number format to appear for either positive or negative values.
4. In the Currency Symbol (system default) area, specify how you want the currency symbol to appear with the values on your report.
5. Click **OK** to save your changes.

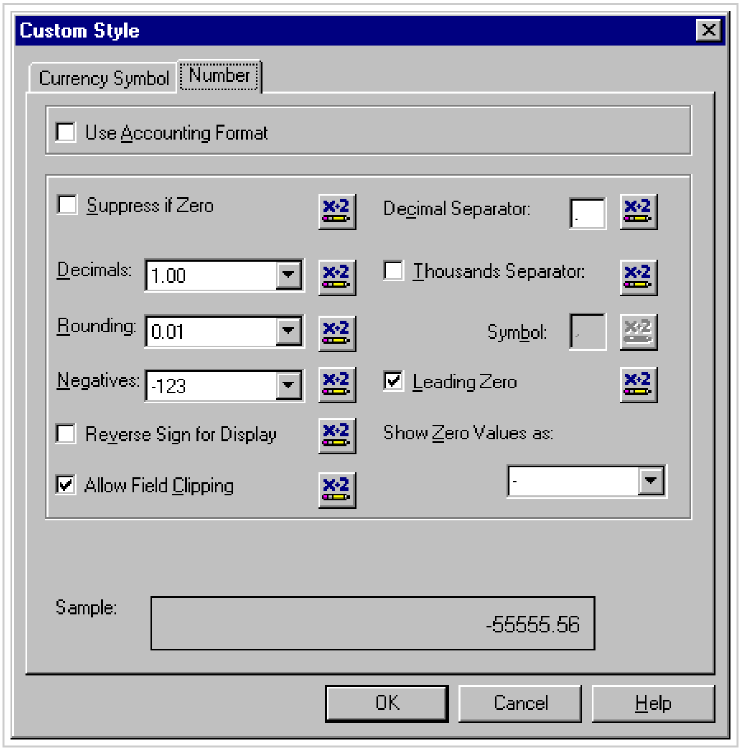
To customize the accounting conventions for a report

1. Right-click the currency field or number field you want to format to bring up the shortcut menu.
2. On the shortcut menu, click **Format**.

   The Format Editor dialog box appears with the Number tab open.

3. In the Style area, select **Custom Style**, then click the **Customize** button.

   The Custom Style dialog box appears with the Number tab open.



- [ ] Use Accounting Format
- [ ] Suppress if Zero
- Decimals: 1.00
- Rounding: 0.01
- Negatives: -123
- [ ] Reverse Sign for Display
- [✓] Allow Field Clipping
- Decimal Separator: .
- [ ] Thousands Separator:
- Symbol: ,
- [✓] Leading Zero
- Show Zero Values as: -
- Sample: -55555.56

4. Select the **Use Accounting Format** check box.

   Once you select this option, the following conditions occur:

- In the Negatives list, how the negative values appear on your report are determined by the Windows locale settings. The negative values are represented by either a minus sign or brackets.
- In the Show Zero Values as list, the dash symbol is automatically selected to represent zero values on your report.
- On the Currency Symbol tab of the Custom Style dialog box, the currency symbol is positioned on the left-side of the currency and numeric values.

  Note: Changes made to the Windows locale settings are implemented only after you exit and restart Crystal Reports.

5. Select the **Reverse Sign for Display** check box to reverse the signs for debit and credit amounts in your financial reports.
6. Click **OK** to save your changes.
7. Click **OK** again to return to your report.

Using white space between rows

The height of a section in relation to the objects within it affects the amount of white space that appears between rows on the report.

The free-form Design tab lets you add and delete white space in two ways:

- using the Resizing cursor to resize the area on the Design tab
- changing the option in the Section Expert.

Click the appropriate link to jump to that section:

- Adding white space by resizing
- Deleting white space by resizing
- Deleting white space by suppressing a section.

Adding white space by resizing

To add extra white space between rows in the report, move the pointer over the lower section boundary line. The pointer changes to a Resizing cursor.

Deleting white space by resizing

To delete unnecessary white space within a section, move the pointer over the lower section boundary line. The pointer changes to a Resizing cursor.

Deleting white space by suppressing a section

If an entire section is blank (for example, if you are not putting anything into the Page Footer section of the report), you can eliminate the unnecessary white space by suppressing the section in the Section Expert.

To delete white space by suppressing a section

1. Right-click menu, click **Format Section**.

   The Section Expert dialog box appears.

   Tip: Another way to do this is to click the Section Expert button on the Standard toolbar.

2. In the Sections area, click the section you want to suppress.
3. On the **Common** tab, select the **Suppress (No Drill-down)** check box.
4. Click **OK** to return to your report.

   The blank section will no longer be printed.

Working with conditional formatting

Conditional formatting is formatting that applies only under certain conditions. For example, in a report you may only want:

- customer balances printed in red if they are past due
- the dates to appear in Day, Month, Year format if the customer is Canadian
- background color to appear on every other line.

Crystal Reports makes it easy to apply conditional formatting in these and hundreds of other situations.

With absolute formatting, you follow the *select, then apply* procedure. For conditional formatting, you follow the same general procedure, but you go a step further and set up conditions that determine whether or not the formatting will be applied. You specify these conditions using simple formulas. For more information on creating formulas using Crystal syntax, see Crystal syntax fundamentals. Or, for more information on creating formulas using Basic syntax, see Basic syntax fundamentals.

When a conditional formatting formula is set up, the formula overrides any fixed settings you have made in the Format Editor. For example, if you select the *Suppress* option, then set up a conditional formula for the *Suppress* option, the property will still apply only if the condition in the formula is met.

Crystal Reports enables you to set both on and off properties and set attribute properties conditionally. However, each of these requires a different kind of formula.

Click the appropriate link to jump to that section:

- Conditional on or off properties
- Conditional attribute properties
- Changing conditional fonts
- Creating footers after the first page
- Using the Highlighting Expert
- Undo/Redo activities.

Conditional on or off properties

A conditional on or off property tests to see if a condition has been met. It is *on* if the condition is met, *off* if the condition is not met. There is no middle ground. Use Boolean formulas for this kind of formatting.

Crystal syntax example

```
condition
```

Basic syntax example

```
formula = condition
```

The program tests each value to see if it meets the condition and it returns a "yes" or "no" answer. It then applies the property to every value that returns a "yes" answer.

Conditional attribute properties

A conditional attribute property tests to see *which* of two or more conditions is met. The program then applies the formatting appropriate to the condition. For example, assume that you want values under quota printed in red and all other values printed in black. The program tests to see whether the value is under quota or not. If it is under quota, then it applies the red attribute; if it is not, then it applies the black attribute.

Use an If-then-else formula for this kind of conditional formatting. For Crystal syntax, see If expressions (Crystal syntax). Or, for Basic syntax, see If statements (Basic syntax).

Crystal syntax example

If Condition A Then
crRed
Else
crBlack

Basic syntax example

```
If Condition A Then
formula = crRed
Else
formula = crBlack
End If
```

When conditional attribute properties are set up, Crystal Reports loads a selection of attributes into the Functions list in the Formula Editor. Double-click any of these attributes to add them to a formula. For example, if you are setting horizontal alignment conditionally, the Functions list contains attributes such as DefaultorHorAligned, LeftAligned, and Justified. If you are setting borders conditionally, the Functions list contains attributes such as NoLine, SingleLine, and DashedLine.

Note: Always include the Else keyword in conditional formulas; otherwise, values that don't meet the If condition may not retain their original format. To retain the original format of values that don't meet your If condition, use the DefaultAttribute function.

Crystal syntax example

```
If Condition A Then
crRed
Else
DefaultAttribute
```

Basic syntax example

```
If Condition A Then
formula = crRed
Else
formula = DefaultAttribute
End If
```

You can take this kind of property one step further. You can specify a list of conditions and a property for each; you are not limited to two conditions. For example, if you have a number field on your report that contains sales figures from countries around the world, you can specify the number attribute(s) that you want to apply to each country. In this case, your conditions would specify that if it is from Country A, the program should apply the Country A attribute; if it is from Country B, the Country B attribute; if it is from Country C, the Country C attribute, and so on.

With more than two alternatives, use this kind of formula:

Crystal syntax example

```
If Condition A Then
crRed
Else If Condition B Then
crBlack
Else If Condition C Then
crGreen
Else
crBlue
```

Basic syntax example

If Condition A Then
formula = crRed
ElseIf Condition B Then
formula = crBlack
ElseIf Condition C Then
formula = crGreen
Else
formula = crBlue
End If

Use a multi-condition If-then-else formula for this kind of conditional formatting.

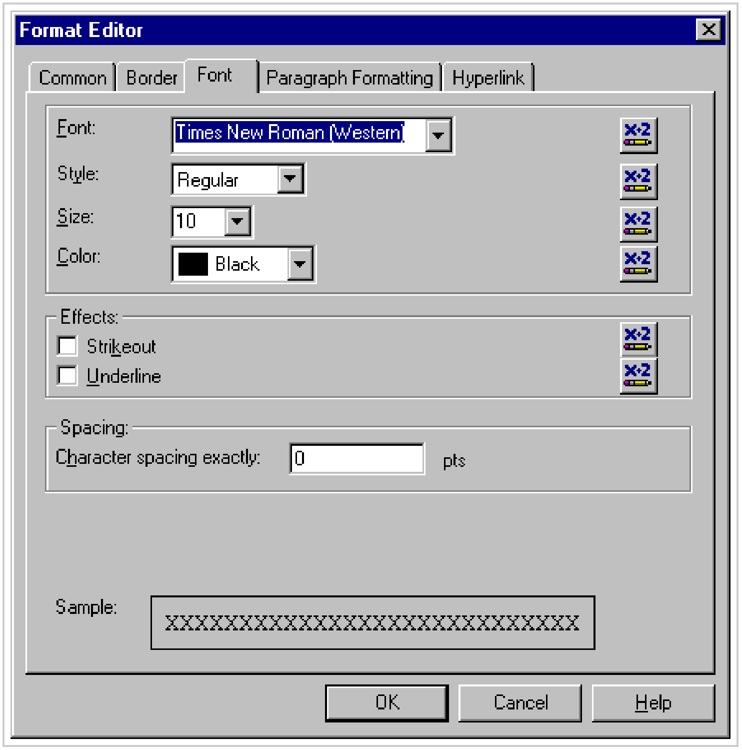
Changing conditional fonts

For memo or string fields that are based on conditions such as a parameter value, you can change the font, font style, size, and color for these fields using the Format Editor.

To change conditional fonts

1. Right-click the field you want to format and then click Format to bring up the shortcut menu.
2. Click the Font Tab.

   The Format Editor dialog box appears.



Format Editor

Font: Times New Roman (Western)
Style: Regular
Size: 10
Color: Black

Effects:
☐ Strikeout
☐ Underline

Spacing:
Character spacing exactly: 0 pts

Sample: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

3. To change any of the font options, click the appropriate **Formula** button, located on the right side of the dialog box.

4. In the Format Formula Editor dialog box, you can specify that conditional fonts will change only when certain conditions are met.
5. Click **Save**.

Note:

- If there is an error in the formula, a message box appears, asking whether to exit without examining the error. If you click No, a second message box will be displayed, detailing the error.
- If there is no error in the formula, you are returned to the Format Editor. Note that the Formula button has changed. This indicates that a formula has been entered for that property. Search for "Formula Compiler Errors" in Crystal Reports online help.

6. Click **OK** to return to your report.

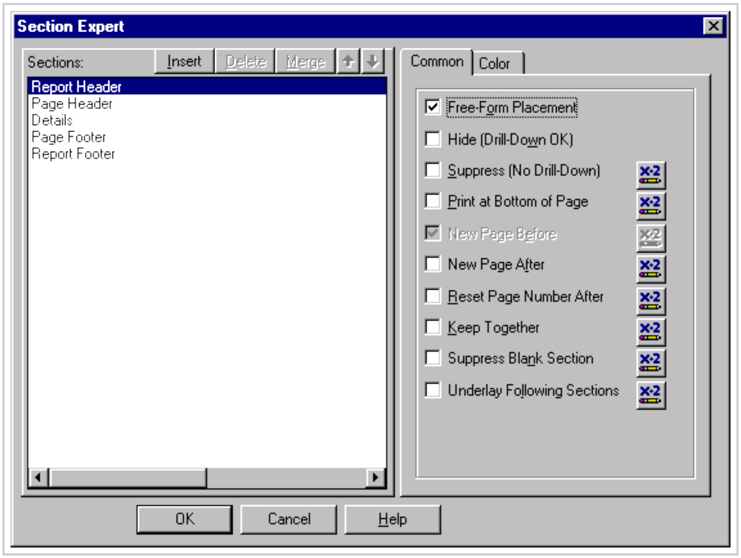
Creating footers after the first page

You may choose to print a page footer on all pages except the first page. You can do this by formatting the Page Footer section conditionally, using an on or off property.

To create footers after the first page

1. Place the field you want displayed as a page footer in the Page Footer section of the report.
2. On the Right-click menu, click **Format Section**.

   The Section Expert dialog box appears.


Section Expert

Sections: Insert Delete Merge

Report Header
Page Header
Details
Page Footer
Report Footer

Common | Color

- [✓] Free-Form Placement
- [ ] Hide (Drill-Down OK)
- [ ] Suppress (No Drill-Down)
- [ ] Print at Bottom of Page
- [✓] New Page Before
- [ ] New Page After
- [ ] Reset Page Number After
- [ ] Keep Together
- [ ] Suppress Blank Section
- [ ] Underlay Following Sections

OK Cancel Help

3. In the Sections area, click **Page Footer**.
4. To open the Format Formula Editor dialog box, click the **Formula** button, located to the right of the **Suppress** (**No Drill-down**) check box.
5. Enter the following formula in the Format Formula Editor:

Crystal syntax example:

```
PageNumber = 1
```

Basic syntax example:

```
formula = PageNumber = 1
```

This formula suppresses the page footer on the first page, but not on any of the other pages.

6. Click **Save**.

   Note:

- If there is an error in the formula, a message box appears, asking whether to exit without examining the error. If you click No, a second message box will be displayed, detailing the error.
- If there is no error in the formula, you are returned to the Section Expert. Note that the Formula button has changed. This indicates that a formula has been entered for that property. Search for "Formula Compiler Errors" in Crystal Reports online help.

7. Click the **Preview Tab** to preview the report and ensure that the page footer appears on all pages but the first.

Note:

- if you have a multi-line page footer and have inserted the lines into separate Page Footer sections, you will need to suppress each section conditionally, using the formula above
- to create a page header that appears on all pages but the first, place the header information in the Page Header section and then suppress that section conditionally, using the same formula that was used for suppressing the Page Footer section.

Using the Highlighting Expert

Crystal Reports includes the Highlighting Expert that allows you to efficiently apply conditional formatting to currency and number fields. The Highlighting Expert is most commonly used for highlighting field values that are in some way distinguished from the other values in the report. For example, if you wanted to highlight the {customer.LAST YEAR'S SALES} field in red whenever the sales are greater than $25,000, you could specify in the Highlighting Expert that when the value is greater than $25,000, a red background color is applied to the field. You can also use the dialog box to set conditional font color and border style.

Think of the Highlighting Expert as an alternative to the Formula Editor. The Highlighting Expert essentially runs the following equation: `If Condition is True, Then Apply These Formatting Specifications`. For this purpose, the dialog box is divided into two areas: the Item list area which displays the formula, and the Item editor area which allows you to set the formula. The Item editor includes the Sample field that lets you view the formatting specifications being applied.

Some of the benefits to using the Highlighting Expert include:

- using the Highlighting Expert for numeric/currency fields in Cross-tabs
- the ability to undo highlighting.

Related topics

- Setting highlighting priorities
- Conditionally formatting fields using the Highlighting Expert.

Setting highlighting priorities

The Priority buttons in the Item list area of the Highlighting Expert allow you to set the priorities for your formulas. This is useful when you have two or more formulas that could offer conflicting results in some situations.

For example, suppose you are highlighting the Unit Price field on the report. You assign to this field a highlighting formula that shows a yellow background when a unit price is greater than $100. Imagine then, that on this same report, you create another highlighting formula that shows a red background when a unit price is greater than $200. Considering that 100 is a subset of 200, you could have Unit Price fields with yellow backgrounds when, in fact, those fields should have red backgrounds. In other words, a unit price of $300 could receive either a red or a yellow background, depending on which formula has been assigned priority.

To set the priority for a formula

1. Right-click on the formula, click **Change Highlighting**.
2. Highlight the formula.
3. In the Item list area, click the **Priority** arrows to move the selected formula to a position above or below the formula(s).

   Note: A formula has priority over another formula when it is higher in the Items list area.

4. Click **OK**.
5. Click the **Preview** tab, or refresh the report to see the highlighting changes.

   Note: The Remove and Remove All buttons in the Item list area can be used to delete formulas.

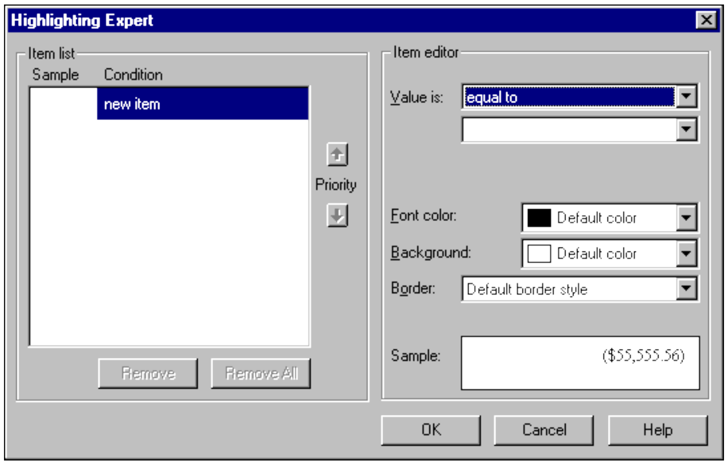
Conditionally formatting fields using the Highlighting Expert

Using the Highlighting Expert, you can conditionally format only currency and number fields.

To conditionally format fields using the Highlighting Expert

1. Right-click the currency or number field you want to format to bring up the shortcut menu.
2. On the shortcut menu, click **Change Highlighting**.

   The Highlighting Expert dialog box appears.



3. In the Item editor area, select a comparison from the **Value is** list, and enter a numeric value in the adjacent box.
4. In the **Font color**, **Background**, and **Border** lists, specify your formatting changes to the selected field.
5. Repeat steps 3 and 4 if you want multiple highlighting settings for the selected field.
6. Click the **Priority** arrows to specify the priority for each additional setting you apply. See Setting highlighting priorities.
7. Click **OK** to return to your report.

Undo/Redo activities

Crystal Reports includes multiple levels of undo. With multiple levels of undo, you can undo any number of changes to an object, in reverse order, until you have your report in the condition you want it.

The program also has a redo feature that reverses an undo. If you move an object, for example, and do not like its new position, you can click Undo to move it back to its original position. If you then change your mind, you can click Redo to restore the latest change.

The Undo and Redo buttons have lists that allow you to undo or redo a number of changes at one time.

- To undo an action, click Undo on the Standard toolbar.

  The first time the button is clicked, it reverses the most recent change made to the report. Each additional time the button is clicked, it reverses the next most recent change.

  To undo several actions at once, click the arrow button to display the list of actions. Select the series of actions you wish to undo.

- To redo a change after you have undone it, click Redo on the Standard toolbar.

  The program disables the Undo button and the Undo/Redo commands whenever there is nothing to undo/redo or when you have made a change that cannot be reversed.

  To redo several actions at once, click the arrow button to display the list of actions. Select the series of actions you wish to redo.

Note: You can only undo or redo actions in order from the most recent backward. You cannot undo an action without undoing more recent actions.

Distributing and Viewing Reports

This section provides you with information about how to distribute finished reports using a variety of methods (printing, faxing, exporting).

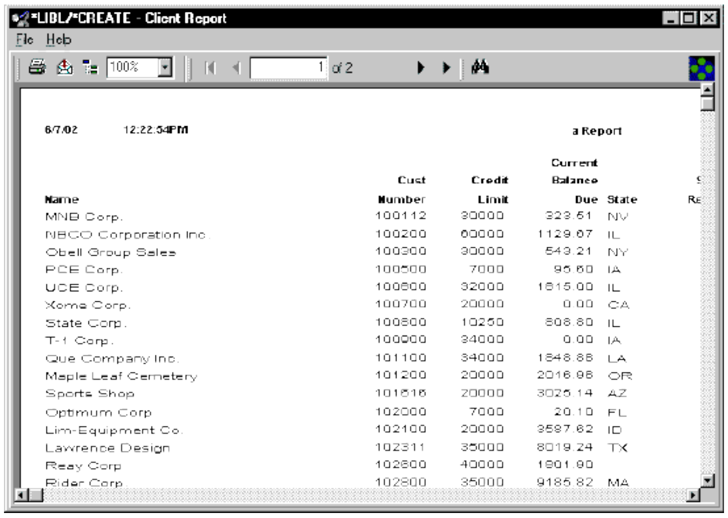
Click the appropriate link to jump to that section:

- Viewing a report (on screen)
- Printing a report
- Faxing a report
- Exporting a report

Viewing a report

On the Viewpoint Explorer File menu, select **Display Results**. You can also, press the Display button on the toolbar.

The Results screen appears. The Results File menu allows you to Print, Fax and Export the report after it is displayed.



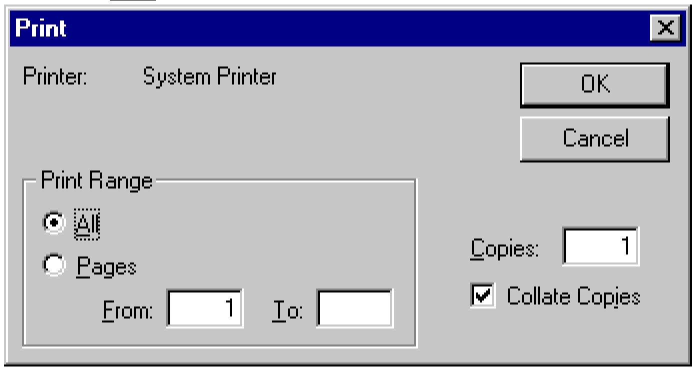
6/7/02 12:22:54PM a Report

Name	Cust Number	Credit Limit	Current Balance Due	State
MNB Corp.	100112	30000	323.51	NV
NBCO Corporation Inc.	100200	60000	1129.67	IL
Obell Group Sales	100300	30000	543.21	NY
PCE Corp.	100500	7000	95.60	IA
UCE Corp.	100600	32000	1615.00	IL
Xoma Corp.	100700	20000	0.00	CA
State Corp.	100800	10250	808.80	IL
T-1 Corp.	100900	34000	0.00	IA
Que Company Inc.	101100	34000	1848.88	LA
Maple Leaf Cemetery	101200	20000	2016.98	OR
Sports Shop	101616	20000	3025.14	AZ
Optimum Corp	102000	7000	20.10	FL
Lim-Equipment Co.	102100	20000	3587.62	ID
Lawrence Design	102311	35000	8019.24	TX
Reay Corp	102600	40000	1901.90	
Rider Corp.	102800	35000	9185.82	MA

Printing a report

NOTE:
To print a report results have to be displayed on screen first. (see Viewing a report)

1. On the **File** menu, select **Print**, then click **PC**. You can also press the Send results to PC Printer button on the Standard toolbar. The Print dialog box appears.



2. Set the following parameters:

- Print range: all pages or a specific range.
- Number of copies.
- Collate copies: if this option is selected, the report will print each page in order. For example, if you are printing two copies of a report with four pages your report will print page one, two, three, and four of the first copy, then print the second copy.

3. Click **OK**.

   The Printing Records dialog box appears showing the progress of your print job.

Faxing a report

Many fax applications, such as Microsoft® Fax and Delrina™ WinFax, allow you to set up a printer driver that will fax documents over a modem. When using one of these applications, you can fax a report from Crystal Reports.

NOTE:
To print a report results have to be displayed on screen first. (see Viewing a report)

1. Run the report.
2. On the **File** menu, click **Printer Setup**.

   The Print Setup dialog box appears.
3. Select the fax driver from the **Name** drop-down list.
4. Click **OK.**
5. On the **File** menu, click **Print**.

   The Print dialog box appears.
6. Click **OK**.

   Your fax application appears, prompting you to select a cover page and to fill in the appropriate fax information.

Exporting a report

Finished reports can be exported to a number of popular spreadsheet and word processor formats, as well as to HTML, ODBC, and common data interchange formats. This makes the distribution of information easier. For example, you may want to use the report data to project trends in a spreadsheet package or to enhance the presentation of data in a desktop publishing package.

The exporting process requires you to specify a format and a destination. The format determines the file type, and the destination determines where the file is located. For more details, see Destination.

NOTE:
All the export feature below are performed from the Crystal report designer.

Related topics

- Exporting to an application
- Exporting to a disk file
- Exporting to a Microsoft Exchange folder
- Exporting to Lotus Domino
- Exporting to Microsoft Mail (MAPI)
- Exporting to HTML
- Exporting to an ODBC data source
- Exporting to CC Mail

Format Types

Crystal Reports provides you with over twenty different export format types. They include:

- Adobe Acrobat (PDF)
- Character-separated values
- Comma-separated values (CSV)
- Crystal Reports (RPT)
- Crystal Reports 7 (RPT)
- Data Interchange Format (DIF)
- Microsoft® Excel
- HTML 3.2
- HTML 4.0 (DHTML)
- Lotus 1
- ODBC
- Paginated Text
- Record style (columns of values)
- Report Definition
- Rich Text Format

- Tab-separated text
- Tab-separated values
- Text
- Word Format
- Word for Windows document
- XML.

In addition to the standard export format types installed on your machine you may find additional export format types are available to you. These are determined by the DLL files on your local machine.

Note: When you export a report to a file format other than Crystal Reports format (.RPT), you may lose some or all of the formatting that appears in your report. However, the program attempts to preserve as much formatting as the export format allows.

Destination

The destination determines the export location of your report. Crystal Report enables you to choose one of six destinations. They include:

- Application
- Disk file
- Microsoft Exchange folder
- Lotus Domino
- Lotus cc:Mail (VIM)
- Microsoft Mail (MAPI).

Exporting to an application

If you export to an application, the program exports the report to a "temp" file in the specified format and then opens the file in the appropriate application.

Note:

- when exporting in ODBC format, "Application" and "Disk file" are the same
- you will need to specify a file path if you are exporting in HTML format
- the file name of the report and the file name of the "temp" file cannot be the same.

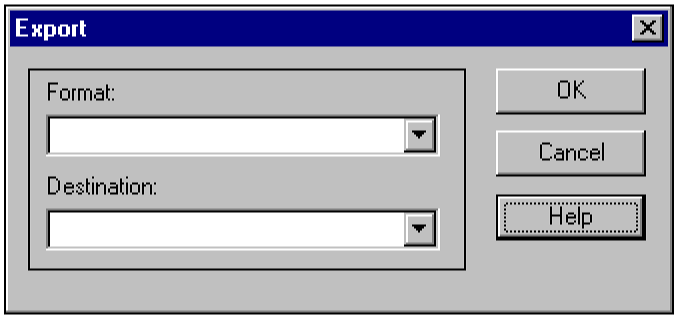
The sections below provide instructions on how to export a report to the Microsoft Excel 7.0 Extended format for each of the different destination types.

To export to an application *(results have to be displayed on screen first – see Viewing a repor t)*

1. Run the report you would like to export.
2. On the **File** menu, click **Export**.

   Tip: Another way to do this is to click the Export button on the Standard toolbar.

   The Export dialog box appears.



3. Select the export format type from the **Format** drop-down list. In this case, choose **Excel 7.0 (XLS) (Extended)**.
4. Select **Application** from the **Destination** drop-down list.
5. Click **OK**.

   The Format Options dialog box appears.

6. Change the formatting options as needed.
7. Click **OK**.

   The Exporting Record dialog box appears.

Note: Click Cancel Exporting to cancel the export process.

The program exports the report and opens it in the appropriate application. In this example, Microsoft Excel opens with the exported data.

Exporting to a disk file

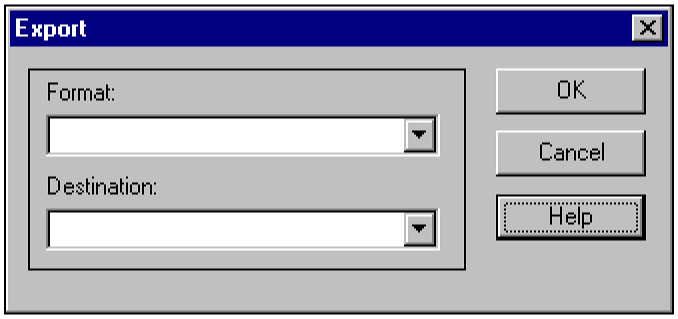
If you export to a disk file, the program saves the report to the disk or diskette you have specified.

To export to a disk file *(results have to be displayed on screen first – see Viewing a repor t)*

1. Run the report you would like to export.
2. On the **File** menu, click **Export**.

   Tip: Another way to do this is to click the Export button on the Standard toolbar.

   The Export dialog box appears.



3. Select the export format type from the **Format** drop-down list. In this case, choose **Excel 7.0 (XLS) (Extended)**.
4. Select **Disk file** from the **Destination** drop-down list.
5. Click **OK**.

   The Format Options dialog box appears.

6. Change the formatting options as needed.
7. Click **OK**.

   The Choose Export File dialog box appears.

8. Select the appropriate directory.
9. Enter the File name.
10. Click **Save**.

Note: Click Cancel Exporting to cancel the export process.

The program exports the report and saves it in the designated directory. In this example the file is saved in the "temp" directory in an .xls format.

Exporting to a Microsoft Exchange folder

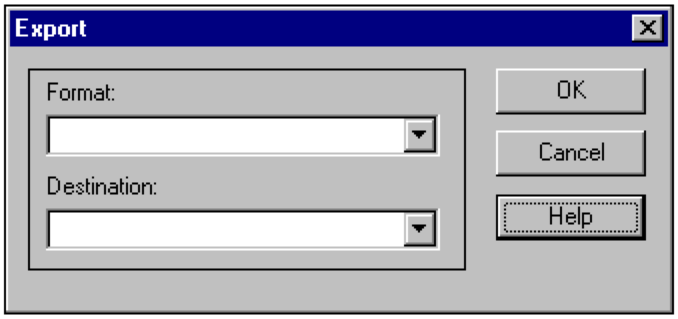
Crystal Reports enables you to export a report file to a Microsoft Exchange folder. You select the folder, and the report is stored there in the format that you specify. A Microsoft Exchange folder can contain standard notes (mail), files, and instances of Microsoft Exchange forms.

To export to a Microsoft Exchange folder *(results have to be displayed on screen first – see Viewing a repor t)*

1. Run the report you would like to export.
2. On the **File** menu, click **Export**.

   Tip: Another way to do this is to click the Export button on the Standard toolbar.

   The Export dialog box appears.



3. Select the export format type from the **Format** drop-down list. In this case, choose **Excel 7.0 (XLS) (Extended)**.
4. Select **Exchange Folder** from the **Destination** drop-down list.
5. Click **OK**.

   The Format Options dialog box appears.

6. Change the formatting options as needed.
7. Click **OK**.

   The Choose Profile dialog box appears.

   Note: You will be prompted by the Microsoft Outlook™ Setup Expert if Microsoft Exchange, Microsoft Mail, or Internet E-mail is not set up on your machine.

8. Select the desired profile from the **Profile Name** drop-down list. If the profile is not listed, click **New** to create it.
9. Click **OK** when finished.
10. When the Select a Folder dialog box appears, select the folder in the profile in which you want the report to appear, and click **OK**.

The report is exported to the Microsoft Exchange folder you selected. The exported report can now be accessed through the Microsoft Exchange client.

Exporting to Lotus Domino

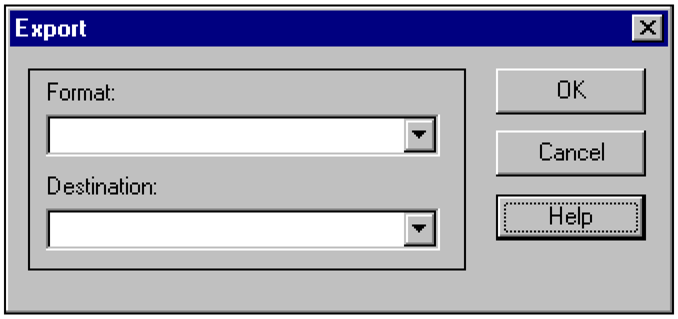
You must have Version 3.0 or later of the Lotus Domino client. You will also require, at a minimum, depositor access. Crystal Reports will not export to a Lotus Domino OS/2 client.

To export to Lotus Domino *(results have to be displayed on screen first – see Viewing a repor t)*

1. Run the report you would like to export.
2. On the **File** menu, click **Export**.

   Tip: Another way to do this is to click the Export button on the Standard toolbar.

   The Export dialog box appears.



3. Select the export format type from the **Format** drop-down list. In this case, choose **Excel 7.0 (XLS) (Extended)**.
4. Select **Lotus Domino Database** from the **Destination** drop-down list.
5. Click **OK**.

   The Format Options dialog box appears.

6. Change the format settings as required.
7. Click **OK**.

   The Select Database dialog box appears.

8. Double-click the Lotus Domino server you would like to export your report to.

   The file name defaults.

9. Select the database you would like to export the report to.
10. Click **OK**.

    The Comments dialog box appears.

11. Type in any comments that are to appear when another user selects your report from the Lotus Domino Desktop.
12. Click **OK**.

    The export process begins.

The next time a user logs onto the Lotus Domino database you specified, they will see the report in their desktop. The user can double-click the report file name to display the comments you wrote and then double-click the report icon to view the report.

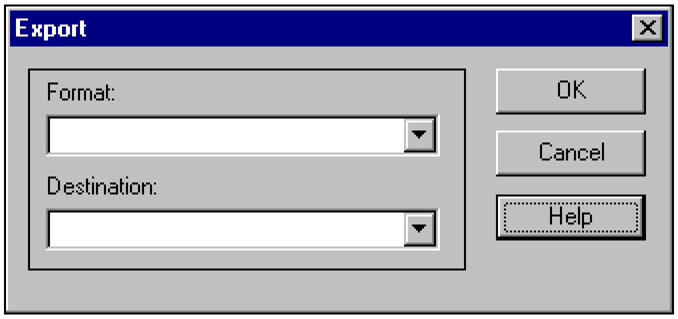
Exporting to Microsoft Mail (MAPI)

(results have to be displayed on screen first – see Viewing a repor t)

1. Run the report you would like to export.
2. On the **File** menu, click **Export**.

   Tip: Another way to do this is to click the Export button on the Standard toolbar.

   The Export dialog box appears.



3. Select the export format type from the **Format** drop-down list. In this case, choose **Excel 7.0 (XLS) (Extended)**.
4. Select **Microsoft Mail (MAPI)** from the **Destination** drop-down list.
5. Click **OK**.

   The Format Options dialog box appears.

6. Change the formatting options as needed.
7. Click **OK**.

   The Send Mail dialog box appears.

8. Enter the address details, then click **Send**.

   The Exporting Record dialog box appears.

Note: Click Cancel Exporting to cancel the export process.

Exporting to HTML

Providing support for the Internet and corporate intranets becomes more important with every passing day. Crystal Reports recognizes this importance and provides World Wide Web support. Although incorporated as an export format, HTML represents a whole new export destination for reports as well.

By exporting reports in HTML format, Crystal Reports provides you with a new option for rapid, convenient distribution of important company data. Once exported, your reports become accessible with many of the most popular Web browsers, including Netscape and Microsoft Internet Explorer.

For more details, see To export to HTML.

To export to HTML

(results have to be displayed on screen first – see Viewing a repor t)

1. With the report you want exported active, click **Export** on the Standard toolbar.
2. From the **Format** drop-down list, select HTML 3.2 or HTML 4.0 (DHTML).
3. Select a destination from the **Destination** drop-down list. The rest of this section assumes you select **Disk file** to store the HTML document in a directory on a Web server.
4. Click **OK**.

   The Export To Directory dialog box appears. When exported to HTML format, a report is saved to one HTML file. If you prefer, click the check box to have each page saved to a separate file. For this reason, the program asks you for the name of a directory for an export destination, and uses default names for the HTML files. The initial HTML page will be saved as Reportfilename.HTM. This is the file you open if you want to view your report through your Web browser.

   Note: When you are exporting to a single file (that is, you have cleared the "Separate HTML Pages" check box) all blank space in headers and footers, as well as top and bottom page margins, is included. If you don't want to see blank spaces in your HTML file, suppress empty header and footer sections and set the top and bottom page margins to zero in the Page Setup dialog box.
5. Select an existing directory, or create a new directory for your report.
6. To navigate through separate HTML files click the Page Navigator option.
7. Click **OK**.

   The program exports the report to HTML format.

Exporting to an ODBC data source

Crystal Reports allows you to export reports to any ODBC data source. If you have an ODBC data source set up for a database or data format, you can export your report to that data format through ODBC.

For instance, you may have an ODBC data source set up through ODBC Administrator that you normally use to access database tables designed in Microsoft SQL Server. Using the Export dialog box, however, you can select your SQL Server data source and export your report as a new SQL Server database table.

Note: You must have an ODBC data source set up through ODBC Administrator in order for the program to export to a particular ODBC database format. See How to set up an ODBC data source.

Exporting to an ODBC data source allows you to:

- change data from a centralized database format into a format compatible with a local DBMS application,
- change data from a local database format into a format compatible with a centralized database,
- create a new database table that can be used as a separate data set in future reporting,
- create a mini data-warehouse, and
- manipulate database data by filtering records, adding formulas, and removing fields to create a new database table that provides the data you need most for your work.

For more details, see To Export to an ODBC data source.

To Export to an ODBC data source

(results have to be displayed on screen first – see Viewing a repor t)

1. With the report you want exported active, click **Export** on the Standard toolbar.
2. From the **Format** drop-down list, select the ODBC data source for the format in which you want to export your report. For example, *ODBC - CRSS* allows you to export your report to a Microsoft SQL Server database.
3. **Destination** is ignored when you are exporting a report to an ODBC data source. You do not need to make any changes to the **Destination** edit box. Click **OK** in the Export dialog box.
4. If your ODBC data source specifies a particular database, the report will be exported to that database. Otherwise, the Select Database dialog box appears. Select the database to which this report will be added as a new table, and then click **OK**.
5. If the ODBC data source you selected requires a Logon ID and password, the Login or SQL Server Login dialog box appears. Enter your ID and Password, and then click **OK**.

   The Enter ODBC Table Name dialog box appears.
6. Enter the name you want to give to the new table in the database, and then click **OK**.

The program exports the report as a new table to the database you specified.

Note: If your report contains a binary field, you will be unable to export it to an ODBC data source successfully.

Exporting to CC Mail

(results have to be displayed on screen first – see Viewing a repor t)

1. With the report you want to export active, click **Export** on the Standard toolbar.

   Tip: Another way to do this is to click the Export button on the Standard toolbar.

   The Export dialog box appears.

2. From the **Destination** drop-down list, select Lotus cc:Mail (VIM).
3. Click **OK**.

   The Character-separated Values dialog box appears.

4. Change the default Separator and Delimiter as required, then click **OK**.

   The number and Date Format dialog box appears.

5. Select the Same number and Same date formats check boxes as required, then click **OK**.
6. Enter password information as required, then click **OK**.

   The Send Mail window appears.

7. Complete the fields as required, then click **Send**.

Using Formulas

This section explains the basics of formulas and introduces you to the Formula Editor in order for you to begin to create formulas.

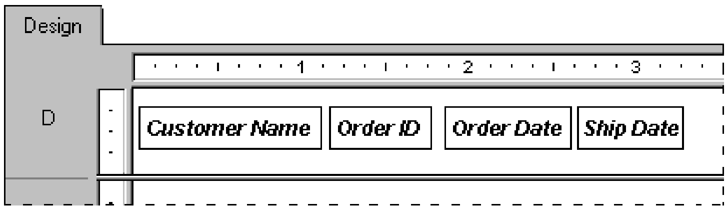
Click the appropriate link to jump to that section:

- Formulas overview
- Formula components and syntax
- Specifying formulas
- Creating and modifying formulas
- Deleting formulas
- Error Messages and Formula Compiler Warnings
- Formulas in action

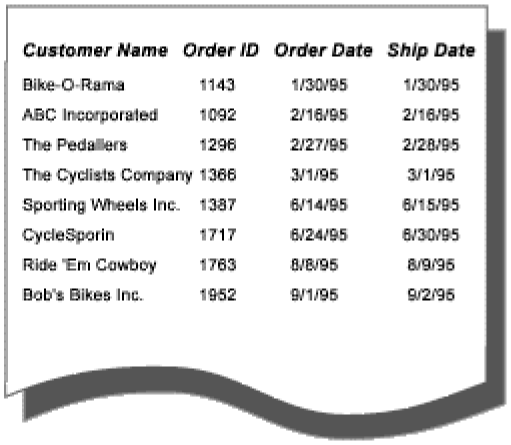
Formulas overview

In many cases, the data needed for a report already exists in database table fields. For example, to prepare an order list you would place the appropriate fields on the report.

By placing these fields ...



You get this kind of report:



Customer Name	Order ID	Order Date	Ship Date
Bike-O-Rama	1143	1/30/95	1/30/95
ABC Incorporated	1092	2/16/95	2/16/95
The Pedallers	1296	2/27/95	2/28/95
The Cyclists Company	1366	3/1/95	3/1/95
Sporting Wheels Inc.	1387	6/14/95	6/15/95
CycleSporin	1717	6/24/95	6/30/95
Ride 'Em Cowboy	1763	8/8/95	8/9/95
Bob's Bikes Inc.	1952	9/1/95	9/2/95

Sometimes, however, you need to put data on the report that does not exist in any of the data fields. In such cases, you need to create a formula. For example, to calculate the number of days it takes to process each order, you need a formula that determines the number of days between the order date and the ship date. Crystal Reports makes it easy for you to create such a formula.

For more details, see Typical uses for formulas.

Typical uses for formulas

There are many uses for formulas. If you have a need for specialized data manipulation, you can do it with a formula.

Creating calculated fields to add to your report

To calculate a price discounted 15%:

Crystal syntax example:

```
{Orders_Detail.Unit Price}*.85
```

Basic syntax example:

```
formula = {Orders_Detail.Unit Price}*.85
```

Formatting text on a report

To change all the values in the Customer Name field to uppercase:

Crystal syntax example:

```
UpperCase ({Customer.Customer Name})
```

Basic syntax example:

```
formula = UCase ({Customer.Customer Name})
```

Pulling out a portion, or portions, of a text string

To extract the first letter of the customer name:

Crystal syntax example:

```
{Customer.Customer Name} [1]
```

Basic syntax example:

```
formula = {Customer.Customer Name} (1)
```

Extracting parts of a date

To determine what month an order was placed:

Crystal syntax example:

```
Month ({Orders.Order Date})
```

Basic syntax example:

```
formula = Month ({Orders.Order Date})
```

Formula components and syntax

Formulas contain two critical parts: the components and the syntax. The components are the pieces that you add to create a formula while the syntax is the rules that you follow to organize the components.

Formula components

Creating a formula in Crystal Reports is like creating one in any spreadsheet application. You can use any of the following components in your formula:

Fields

Example: {customer.CUSTOMER LAST NAME}, {customer.LAST YEAR'S SALES}

Numbers

Example: 1, 2, 3.1416

Text

Example: "Quantity", ":", "your text"

Operators

Example: + (add), / (divide), (negate)

Operators are actions you can use in your formulas.

Functions

Example: Round (x), Trim (x)

Functions perform calculations such as average, sum, and count. All functions available are listed with their arguments and are arranged by their use.

Control Structures

Example: "If" and "Select", "For" loops

Group field values

Example: Average (fld, condFld), Sum (fld, condFld, "condition")

Group field values summarize a group. For example, you could use group field values to find the percentage of the grand total contributed by each group.

Other formulas

Example: {@GrossProfit}, {@QUOTA}

For more details, see Formula syntax.

Formula syntax

Syntax rules are used to create correct formula. Some basic rules are:

- enclose text strings in quotation marks
- enclose arguments in parentheses (where applicable)
- referenced formulas are identified with a leading @ sign.

Crystal and Basic Syntax

When creating formulas, you have the option of using either Crystal or Basic syntax. Almost any formula written with one syntax can be written with the other. Reports can contain formulas that use Basic syntax as well as formulas that use Crystal syntax.

Crystal syntax is the formula language included in all prior versions of Crystal Reports. Basic Syntax is new to Crystal Reports 8.

If you are familiar with Microsoft Visual Basic or other versions of Basic, then Basic syntax may be more familiar to you. In general, Basic syntax is modeled on Visual Basic except that it has specific extensions to handle reporting.

If you are already comfortable with Crystal syntax, you can continue to use it, and benefit from the new functions, operators and control structures inspired by Visual Basic.

Note: Report processing is not slowed down by using Basic syntax. Reports using Basic syntax formulas can run on any machine that Crystal Reports runs on. Also, using Basic syntax formulas does not require distributing any additional files with your reports.

Related topics

To learn about Basic syntax, see Creating Formulas with Basic Syntax.

To learn about Crystal syntax, see Creating Formulas with Crystal Syntax.

Specifying formulas

There are four different groups of formulas in Crystal Reports: report, formatting, selection, and search formulas. The majority of formulas in a report are report formulas and conditional formatting formulas.

Report formulas

Report formulas are formulas that you create to stand alone in a report. For example, a formula that calculates the days between the order date and the shipping date is a report formula.

Conditional formatting formulas

Formatting formulas change the layout and design of a report, as well as the appearance of text, database fields, objects, or entire report sections. You format text through the Format Editor. If you need to create a formatting formula, you access the Formula Editor from the Format Editor. See Working with conditional formatting.

Selection formulas

Selection formulas specify and limit the records and groups that appear in a report. You normally do not enter these formulas directly, but instead specify the selection using the Select Expert. Crystal Reports then generates the record selection and group selection formula. You have the option to manually edit these formulas, but you must use Crystal syntax. See Selecting records.

Note: If you already know Basic syntax, you need to know only a small amount of Crystal syntax to modify most selection formulas.

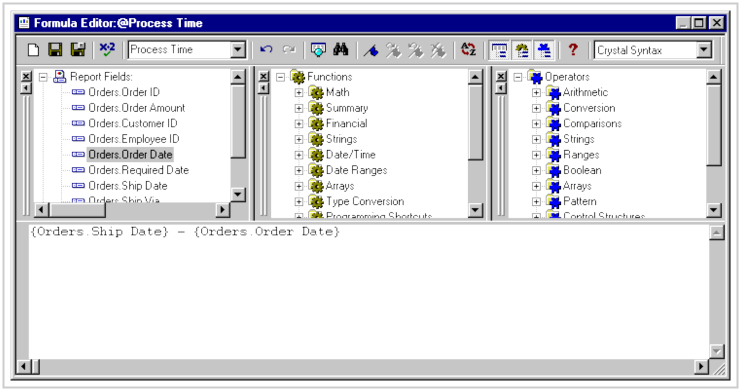
Working with the Formula Editor

In cases other than selection formulas—which you can define using the Select Expert and Search Expert—you create all formulas in the Formula Editor of Crystal Reports.

Accessing the Formula Editor

1. In the field view list, right-click on **Formula Fields**, the click **New**.
2. In the Formula Name dialog box, enter the name of the formula you will create.
3. Click **OK**.

   The Formula Editor appears.



```
{Orders.Ship Date} - {Orders.Order Date}
```

Related topics

- Understanding the sections of the Formula Editor
- Choosing the syntax
- Entering formula components
- Formula Editor buttons
- Changing the font size of the Formula Editor
- Formula Editor Key Controls

Understanding the sections of the Formula Editor

The Formula Editor contains four main windows.

Window	Description of contents
Report Fields	Report fields contain all database fields accessible for your report. They also contain any formulas or groups already created for the report.
Functions	Functions are prebuilt procedures that return values. They perform calculations such as average, sum, count, sin, trim, and uppercase.
Operators	Operators are the "action verbs" you use in formulas. They describe an operation or an action to take place between two or more values. Examples of operators: add, subtract, less than, and greater than.
Formula text window	Area where you create a formula.

Choosing the syntax

The top right corner of the Formula Editor contains the drop-down list where you choose either Crystal or Basic syntax for the formula you are creating.

Note: Changing the syntax from Crystal syntax to Basic syntax or vice versa will change the list of functions in the Functions window as well as the list of operators in the Operators window. The functions and operators differ from syntax to syntax.

The available report fields remain the same since the report fields are available to each syntax.

Setting the default syntax

When you open the Formula Editor, Crystal syntax appears as the syntax default. If you want to change the syntax default, select Options from the File menu, then click the Reporting tab. Choose the preferred syntax from the Formula Language drop-down list and click OK. When you access the Formula Editor, the syntax you selected appears as the default.

Entering formula components

The Report Fields, Functions, and Operators tree at the top of the Formula Editor contain the primary formula components. Double-click any component from these trees to add this component to your formula.

For example, if you set the syntax to Basic Syntax and double-click the Operators > Control Structures > Multi-line If in the Operators tree, the following text is transferred to the Formula text window with the cursor between the If and Then:

```
If | Then
ElseIf Then
Else
End If
```

The above text helps you organize the parts needed to write your formula.

The following example in Crystal syntax shows how double-clicking on the report fields, functions, and operators insert the required syntax for a formula.

Formula Editor buttons

The buttons on the Formula Editor toolbar perform the following functions:

🗋	Creates a new formula without leaving the Formula Editor.
💾	Saves the formula.
💾	Saves the formula and closes the Formula Editor. You return to the report design window.
✓	Tests the syntax of the formula and identifies syntax errors if they are found.
↶	Undoes the last action performed.
↷	Redoes the last action.

When you select a field from the Report Fields window and click the Browse Data button, a dialog box appears with a list of the values for the selected field.

View values for the selected field and select individual values to insert into your formula.

Searches the Formula Editor for a specified word, number, or formula string.

Inserts a bookmark at the beginning of a selected formula line. Click the button again to remove the bookmark.

Jumps to the next bookmark.

Jumps to the previous bookmark.

Deletes all bookmarks in the current formula.

Arranges all Report Fields, Functions, and Operators trees in alphabetical order.

Hides or views the Report Fields tree.

Hides or views the Functions tree.

Hides or views the Operators tree.

Changing the font size of the Formula Editor

The Default Settings dialog box (Designer menu) allows you to change the font size and the background/foreground color of text, comments, and keywords in the Formula Editor.

Formula Editor Key Controls

The Formula Editor accepts the following key controls:

Keyboard Combination	Action Performed
Alt+B	Opens Browse dialog box for highlighted field.
Alt+C	Checks formula for errors.
Alt+F	Shows Field tree (when it has been closed).
Alt+H	Opens online help for Crystal Reports.
Alt+P	Shows Operator tree (when it has been closed).
Alt+S	Saves formula without closing Formula Editor.
Alt+T	Shows toolbar (when it has been closed).
Alt+U	Shows Function tree (when it has been closed).
Ctrl+A	Selects all.
Ctrl+C	Copies.
Ctrl+End	Goes to the end of the last line of the formula.
Ctrl+F	Opens the Find dialog box (same as clicking the find button).
Ctrl+F2	Sets bookmark.
Ctrl+Shift+F2	Clears all bookmarks.
Ctrl+Home	Goes to the beginning of file.

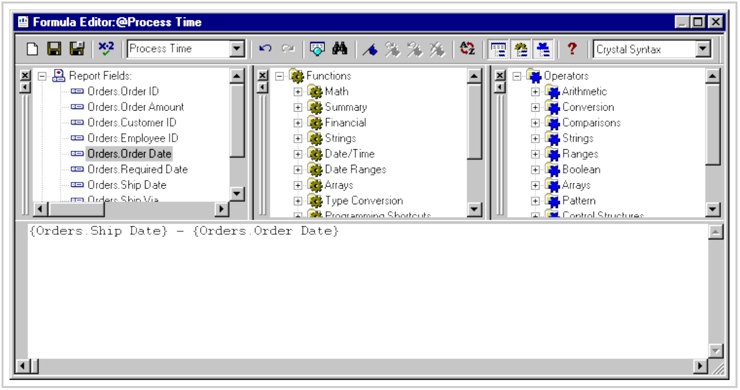
Ctrl+Left Arrow	Goes to the beginning of the left word (+Shift key will do the selection).
Ctrl+M	Sets focus to formula name list box.
Ctrl+N	Opens Formula Name dialog box to create a new formula.
Ctrl+O	Sorts contents of trees.
Ctrl+S	Saves formula and closes Formula Editor.
Ctrl+T	Sets focus to the syntax name list box.
Ctrl+Shift+ Tab	Changes focus to next control box (reverse order of Ctrl-tab).
Ctrl+Tab	Changes focus to next control box.
Ctrl+V	Pastes.
Ctrl+X	Cuts.
Ctrl+Z	Undoes an action.
Ctrl+Shift+Z	Repeats an action.
End	Goes to end of line.
Enter	Copies a selected object from a list to the formula text box.
F2	Goes to next bookmark.
Shift+F2	Goes to previous bookmark.

Creating and modifying formulas

Creating a formula and inserting it into a report

1. In the field view list, right-click on **Formula Fields**, then click **New**
2. In the Formula Name dialog box, enter the name you want to identify the formula.
3. Click **OK**.

   The Formula Editor appears.



```
{Orders.Ship Date} - {Orders.Order Date}
```

4. Choose either Crystal or Basic syntax.

   If you are unsure which syntax to choose see Formula syntax.

5. Enter the formula by typing in the components or selecting them from the component trees.
6. Click **Check** to identify any errors in the formula.
7. Fix any syntax errors the Formula Checker identifies.
8. When the formula has the correct syntax, click **Save and Close**.

   You return to the report design window.
9. Double-click the formula and position the cursor where you want the formula to appear on your report.
10. Click once to set the field in the desired position.

Note: A formula that is placed on a report is indicated by @ (for example, @ProcessTime) on the Design tab.

Related topics

Creating Formulas with Basic Syntax.

Creating Formulas with Crystal Syntax.

Editing formulas

1. Right-click the formula you want to edit and choose Edit.

   The Formula Editor appears.
2. Edit the formula.
3. Click **Check** to identify any errors in the formula.
4. Fix any syntax errors the Formula Checker identifies.
5. When the formula has the correct syntax, click **Save and Close**.

   You return to the report designer window.

Searching and replacing text

1. On the **Insert** menu, click **Field Object**.

   The Field Explorer dialog box appears.
2. Right-click the formula you want to edit and choose **Edit**.

   The Formula Editor appears.
3. In the Formula Editor dialog box, click **Find/Replace** to open a Find dialog box.

   From this dialog box, you can search and replace text within the Formula text box.

4. Click the **Mark All** button to mark all occurrences of the search text.
5. Click the **Replace All** button to replace all occurrences of the search text with the contents of the Replace with text box.

You can also search (but not replace) within any of the Formula Editor trees (use the Search options to specify which list boxes you are searching). The Mark All, Replace, and Replace All buttons become inactive when you specify a search within a list box.

Key points for editing a copy of a formula

When making changes, use the following points as a guide:

- All fields, formulas, and group fields referenced in the formula copy must actually exist in the new report. This means that any database referenced in the original formula (or a database with the same structure, field names, and alias) must be active in the new report.
- If such a database is not active, you must change the field, formula, and group field references in the formula copy to correspond to elements in your new report.
- If the formula contains conditional elements, make certain that the conditions apply to the data in the new report. For example, if the formula in your old report performed an action when the quantity was greater than 100, make sure that the greater than 100 condition makes sense in the new formula. When modifying a formula, you may find that greater than 10 or greater than 2000 makes more sense with your new data.
- If you are using the formula with new data, and if your report contains statements similar to the following:

  ```
  If {file.FIELD} = "text string"
  ```

  Make sure that the text strings used in the formula match values that actually exist in the new data.

Deleting formulas

When a formula is created and added to a report, the Report Designer:

- Stores the specification for creating the formula, using the name you assigned to it.
- Places a working copy of that formula at the point you specify in the report. A working copy is any occurrence of the formula in the report.

In order to completely delete formulas, you must delete the specification and all working copies of the formula.

Note: You cannot delete the specification without deleting all working copies of the formula.

Removing the working formula from your report

1. Right-click the formula you want to delete from the report.
2. Select **Delete**.

Note: Even after the working copies of a formula have been deleted from the report, the formula specification remains unchanged. The specification is listed in the field view list. It is available if you wish to enter the formula in the report again.

Deleting the formula specification

1. Right-click the formula you want to delete and choose **Delete**.

Note: A dialog box appears if this formula is currently in use in a report. If you delete this formula, you will delete all references of it in reports. Click Yes to delete.

Error Messages and Formula Compiler Warnings

A) is missing.

Parentheses must be used in pairs; each opening parenthesis must be matched with a closing parenthesis. One of your opening parentheses is not matched by a closing parenthesis. Insert the missing parenthesis and recheck.

A] is missing.

Brackets must be used in pairs; each opening bracket must be matched with a closing bracket. One of your opening brackets is not matched by a closing bracket. Insert the missing bracket and recheck.

A Boolean range variable is not allowed.

You have entered a Boolean range variable. Range variables are allowed in all data types other than Boolean. Either change the data type to something other than Boolean, or enter a Boolean item variable to replace the Boolean range variable.

Access denied.

DOS will not allow access to a file specified. Make certain the file is not in use by another program (or another user on a network), and/or make certain you have the right network permissions and try again.

A day number must be between 1 and the number of days in the month.

You have entered a day number that does not fit the month. The Formula Checker displays this warning if, for the month of January, for example, you enter a day number of zero (0) or a number 32 or greater. Change the day number to fit the month and recheck.

A field is required here.

You have entered something in your formula other than a field at a position where a field is expected. Correct the problem and recheck.

A formula cannot refer to itself, either directly or indirectly.

You cannot enter a formula that refers to itself. For example, in creating the formula @Profit, you cannot use @Profit as the argument to a function. Remove the reference and recheck.

A function is required here.

The Formula Editor is expecting a function but none was entered. Review your formula and enter the required function or correct the formula if it is in error.

A memo field cannot be used in a formula.

You have picked a memo field for use in a formula. The use of memo fields in formulas is not allowed. Remove the memo field from the formula and try again.

A month number must be between 1 and 12.

You have entered a month number that falls outside the allowable range. Enter a month number between 1 and 12 and recheck.

A string can be at most 254 characters long.

The program allows strings in formulas to be up to 254 characters long. You have entered a string that exceeds that limit. Reduce the length of the string and recheck.

A subscript must be between 1 and the length of the string.

You have entered a subscript number that specifies a character that does not exist. If you enter a subscript that references the 6th or the 8th character in a five character string, for example, you will get this warning. Change the subscript to a value that exists and recheck.

A subscript must be between 1 and the size of the array.

You have entered a subscript that specifies an array item that does not exist. If you enter a subscript that references the 6th or 8th item in a five item array, for example, you will get this warning. Change the subscript to a value that exists and recheck.

Your current license specifies the maximum number of users that can use the program on a network at any given time. The maximum number of users is currently using the program. You can increase the number of users allowed on the system at a given time by purchasing additional Network LanPaks through Seagate Software, Information Management Group.

A subtotal condition is not allowed here.

You have entered a subtotal condition for a subtotal that uses something other than a date or Boolean field as the sort and group by field. Your subtotal does not require a condition. Delete the condition and continue.

A subtotal condition must be a string.

You have entered a subtotal condition that is not in string format. Make certain when you enter the condition in the formula that it is surrounded by single or double quotation marks.

A variable cannot be redeclared with a different type.

You have declared a variable with the same name but a different data type than a variable already declared. This is not allowed. Either change the name of the variable or change the data type so it conforms with the original data type.

A variable is required here.

You have used the assignment operator **(=:)** in a formula without preceding it with a variable. The program expects to see a variable immediately before (to the left of) the assignment operator. Enter a variable and try again.

A variable name is expected here.

You have declared a variable data type without declaring a variable name. You must enter a variable name to complete the declaration. Enter the variable name and continue.

Dates must be between year 1 and year 9999.

You have entered a date that falls outside the allowable range. Enter a date that falls within the range of years 1 to 9999 (including the end values), and then recheck.

Cannot allocate memory.

This message typically indicates that there is not enough memory available. Close any reports that are not needed, and exit any programs that are not essential. Then try again.

Cannot reallocate memory.

This message typically indicates that there is not enough memory available. Close any reports that are not needed, and exit any programs that are not essential. Then try again.

Disk full.

You have attempted to save a report to a disk that is full. Either save to a different disk, or delete unnecessary files from the current disk and try again.

Division by zero.

You have entered a formula that attempts a division by zero. The program does not allow such a division. Edit the formula so it does not attempt to divide by zero, and then recheck.

To avoid this type of problem, you can use a test such as this:

```
If {file.FORECAST} = 0 Then
0
Else
{file.SALES} / {file.FORECAST}
```

Error in formula code. Please contact Seagate Software, Information Management Group.

There is something unusual about the formula that was not foreseen. Please save the formula text that produced this warning and contact the company.

Error in parse tree. Please contact Seagate Software, Information Management Group.

In parsing your formula, the program encountered a situation that the parse tree could not process. Please save the formula text that produced this warning and contact the company.

Field still in use.

The field you are requesting is currently in use. Try again once the field becomes available.

File name already in use. Please close the window for xxx before saving under this name.

You have tried to save a file under the name of a file already in use in an open report. Close that report first, and then try again.

File not found.

The file name you specified cannot be found. Either the filename or the path is incorrect. Enter the correct filename/path and try again. In some instances the file WBTRVDEF.DLL is missing from your directory. This file is required for reading Data Dictionary files along with WBTRCALL.DLL.

File permission error.

You have requested a file for which you do not have permission. You must gain the necessary permission before you can activate the file.

Incorrect Borland Custom Control DLL (BWCC.DLL) installed. Version m.n or higher required.

The program is finding and using a version of BWCC.DLL that is too old for proper program operation. Here's how this can happen:

BWCC.DLL is installed in the CRW directory (the same directory in which CRW.EXE or CRW32.EXE resides) during program installation.

The CRW directory is added to the end of the path statement in AUTOEXEC.BAT during installation (if you allowed the installation program to update the path statement).

If an older version of BWCC.DLL has been installed in the Windows directory, the Windows System directory, or a directory that appears earlier in the path than the CRW directory (the result of an earlier installation), the program picks up that version, not the newer version in the CRW directory.

To correct this problem

The correct version of BWCC.DLL is shipped with Crystal Reports. To correct the problem, delete older versions of BWCC.DLL that reside in directories earlier in the path than CRW.

If this does not solve the problem, move the latest version of BWCC.DLL from the CRW directory to the Window's directory.

Internal Error: PrintDlg fail: 4100.

There is no printer driver installed in the Windows Control Panel. When Crystal Reports opens a report, it looks for the printer that was saved with the report. If that printer cannot be found, it looks for the default printer. If there is no default printer set, the error message results.

Insufficient memory available.

There is not enough memory available to do what you want the program to do. Free up memory and try again.

Invalid DOS version.

You are using a version of DOS earlier than Version 3.0. Install DOS Version 3.0 or higher and try again.

Invalid file handle.

You have specified a file handle that does not exist. Enter the correct file handle and continue.

No default printer selected. You may use the Control Panel to select a default printer.

Please use the Control Panel to select a printer and start the program again.

You cannot begin to utilize the program unless you have a default printer selected. Trying to start the program without a default printer results in this error message.

To select a default printer:

Click the Printers icon in the Windows Control Panel; the Printers dialog box appears with all installed printers listed in the Installed Printers box.

If you have not yet installed the printer, install it first, and then double-click its listing.

Note: A printer must first be given the status Active before it can be selected as the default printer.

Note: For additional information in installing printers and default printers, please refer to the documentation that came with Microsoft Windows.

Non unique table reference: tablename.

This is usually caused if a table name contains an underscore or is more than 15 characters long, or starts with a number.

Not enough arguments have been given to this function.

The function requires more arguments than you have entered. Enter the missing argument(s) and recheck.

Not enough memory.

There is not enough memory available to process the command. Close any reports that are not needed, and exit any programs that are not essential. Then try again.

Numeric overflow.

An intermediate result or the final result cannot be represented because it is too big. Restructure or subdivide the formula to create smaller results, and then recheck.

Physical database not found.

The program is unable to locate either a DLL or the database. Check to make certain that the directories that hold these files are listed in the path statement.

Please cancel printing before closing.

Your Report Engine call is attempting to close a job while it is still in progress. Make certain that you cancel the printing before you close the print job.

Printer not available.

There are problems connecting to the selected printer. Reselect the printer through the Windows Control Panel and try again.

Report file already exists. Overwrite sample.rpt?

You are attempting to save a report under the same name as an existing report. This will overwrite the existing report and make it no longer available. Click Yes to overwrite the report, No to stop the saving process to give you a chance to select a different name.

Report has changed. Save changes to sample.rpt before closing?

You are attempting to close a report window without first saving it, even though you have made changes to the report since you opened it. The changes will be lost unless you save the report before closing. Click Yes to save the changes, No to close the report without saving the changes.

Sorry, this feature is not yet implemented. Try again later.

You have attempted to use a feature that has not been implemented in the current release. Wait till an upgrade that implements the feature and try again.

The formula cannot be evaluated at the time specified.

You are trying to force a field, formula, or function to be evaluated earlier than is possible. Evaluation time functions can only force a later evaluation time, never an earlier one. Change the formula to accommodate the required evaluation time.

This field cannot be used because it must be evaluated later.

You are trying to force a field, formula, or function to be evaluated earlier than is possible. Evaluation time functions can only force a later evaluation time, never an earlier one. Change the formula to accommodate the required evaluation time.

This formula cannot be used because it must be evaluated later.

You are trying to force a field, formula, or function to be evaluated earlier than is possible. Evaluation time functions can only force a later evaluation time, never an earlier one. Change the formula to accommodate the required evaluation time.

This function cannot be used because it must be evaluated later.

You are trying to force a field, formula, or function to be evaluated earlier than is possible. Evaluation time functions can only force a later evaluation time, never an earlier one. Change the formula to accommodate the required evaluation time.

The formula is too complex. Try simplifying it.

The formula could not be evaluated because it exceeds the limit of 50 pending operations. Pending operations are operations that are on hold due to order of precedence rules; they will be performed once the operations with higher level precedence are finished.

Sometimes it is possible to rearrange the formula and calculate the same value without requiring as many pending operations. As a very simplified example, in the formula 2+3*4, the addition cannot be performed until the multiplication has been done. The addition thus becomes pending, on hold until the multiplication is complete. If the formula is written as 3*4+2 instead, the operations can be performed left-to-right with the same result, thus eliminating the pending operation.

Correct the formula and recheck.

The matching } for this field name is missing.

Field names must be enclosed in braces { }. You have entered one of the required braces but not the other. Insert the missing brace and recheck.

The matching ' for this string is missing.

A string that begins with a ' must end with a ' before the end of the line. You have used the ' in one of those positions but not the other. Insert the missing punctuation and recheck.

The matching " for this string is missing.

A string that begins with a " must end with a " before the end of the line. You have used the ' in one of those positions but not the other. Insert the missing punctuation and recheck.

The number of copies of the string is too large or not an integer.

Using the ReplicateString function, you have requested too many copies or you are requesting a non-integer number of copies. Lower the number of copies requested or specify an integer number of copies and try again.

The number of days is too large or not an integer.

When adding days to dates, or subtracting days from dates, you can use only an integer number of days (a whole number); you cannot add or subtract non-integer numbers of days (1/2 days, 3.6 days, etc.). Additionally, once you add or subtract days from a date, the resulting date must fall within the allowable (year) date range, 0000999. If you enter a non-integer number of days or if your result falls outside the allowable range, the Formula Editor displays this warning. Correct the problem and recheck.

The number of decimal places is too large or not an integer.

The second argument to the Round(x, # places) or ToText(x, # places) functions must be a small integer (whole number). You have entered a number as the second argument (# places) that specifies too many decimal places or that is not an integer. Change the number to a small integer and recheck.

The record selection formula cannot include 'PageNumber', 'RecordNumber', 'GroupNumber', 'Previous', or 'Next'.

You cannot include PageNumber, RecordNumber, GroupNumber, Previous, or Next fields in record selection formulas. Eliminate the field(s) and recheck.

The record selection formula cannot include a summary field.

You have included a summary field in a record selection formula. The program does not allow this. Remove the summary field and recheck.

The remaining text does not appear to be part of the formula.

You have provided a formula operand (the item on which a formula operation is to be performed) where none is expected. Often this means that you have forgotten an operator, or an earlier part of a function, or some required syntax item. Correct the error and then recheck.

The result of a formula cannot be a range.

You have created a formula that results in a range. A formula must result in a single value. Correct the formula and recheck.

The result of a formula cannot be an array.

You have created a formula that results in an array. A formula must result in a single value. Correct the formula and recheck.

The result of the selection formula must be a Boolean.

You have created a selection formula that returns something other than a Boolean value. Reconstruct the formula using comparison operators (=, etc.) and recheck.

The string is non-numeric.

The argument to the ToNumber function must be a number stored as a string (for example, a customer number, an ID number, etc.). The string may be preceded by a minus sign and may contain leading and trailing blanks. You have used an argument that is non-numeric and therefore cannot be converted to a number. Change the argument to numeric and recheck.

The variable could not be created.

The variable you declared could not be created. Check the spelling and syntax of your declaration statement and try again.

The word 'Else' is missing.

In an If-then-else expression, you have left out (or misplaced) the 'Else' component and the formula will not function. Insert (or reposition) the 'Else' component and recheck.

The word 'then' is missing.

In an If-then-else expression, you have left out (or misplaced) the 'then' component and the formula will not function. Insert (or reposition) the 'then' component and recheck.

There are too many characters in this field name.

A field name may have at most 254 characters. You have entered a field name that exceeds that number. Enter a field name that has an allowable number of characters and try again.

There are too many characters in this string.

Strings in formulas are allowed to be up to 200 characters long. You have entered a string that exceeds that limit. Reduce the length of the string (or break it into 2 or more concatenated strings) and recheck.

There are too many digits in this number.

Numbers in formulas are allowed to have up to 25 digits before the decimal point. You have entered a number that exceeds that limit. Reduce the size of the number (or break it into 2 or more smaller numbers) and recheck.

There are too many letters and digits in this name.

A variable name can have at most 254 characters. You have entered a name that exceeds that number. Shorten the name to conform to the limit and continue.

There is an error in this formula. Please edit it for more details.

You have tried to accept a formula (via the Accept button in the Formula Editor) that contains an uncorrected error. Correct the error that was indicated and try again.

There must be a subtotal section that matches this field.

You have entered a subtotal in a formula without there being a corresponding subtotal in the report itself. Any subtotal you enter in a formula must duplicate a subtotal already in your report. Add the required subtotal to the report and then renter the formula, or delete the formula, and then recheck.

The special variable field could not be created.

This message typically indicates that there is not enough memory available. Close any reports that are not needed, and exit any programs that are not essential. Then try again.

The summary field could not be created.

This message typically indicates that there is not enough memory available. Close any reports that are not needed, and exit any programs that are not essential. Then try again.

This field cannot be summarized.

You have entered a summary field that does not already exist in your report. Any summary field you enter in a formula must duplicate a summary field already in your report. Either enter the summary field in your report first and then renter it in the formula, or do not enter the summary field in the formula at all.

This field cannot be used as a subtotal condition field.

The field you are entering as a condition field causes the subtotal in the formula not to match any subtotal in the report. Any subtotal you enter in a formula must duplicate a subtotal already in your report. Either enter the subtotal in your report first and then renter it in the formula, or do not enter the subtotal in the formula at all.

This field has no previous or next value.

You have used a field for which there is no "previous" value as the argument for the Previous or PreviousIsNull function, or you have used a field for which there is no "next" value as the argument for the Next or NextIsNull function. To use either of those functions, replace the argument with a field that contains the appropriate values.

This field must be in the same section as the current formula.

Since the field was put into the formula as an operand, it has been moved to a section where it is no longer a valid operand.

This field name is not known.

You have entered a field name that does not appear in any of the active databases. Correct the spelling of the field name and/or its alias, and then recheck. Or, to enter a field name from a database that is not currently active, activate the database first and then renter the field name.

This group section cannot be printed because its condition field is non-existent or invalid.

Your report contains a group section that is based on a condition field that is either no longer in the report or changed so it is invalid for the group section. Review your grouping criteria to identify and correct the source of the problem.

This array must be subscripted. For example: Array [i].

You have entered an array without enclosing it in brackets. Enclose the array in brackets and recheck.

This subtotal condition is not known.

You have entered a subtotal condition that does not appear anywhere in your report. Any subtotal you enter in a formula must duplicate a subtotal already in your report. Change the condition and recheck.

Too many arguments have been given to this function.

You have entered an array as the argument to a non-array function. This kind of problem can occur, for example, if you forget to use brackets (the required syntax items for an array) to enclose an array. The Formula Checker sees the array values as arguments to a non-array function and displays the error message.

Too many items have been given for this array.

The program allows up to 50 values in an array. You have exceeded this limit. Reduce the number of values in the array and recheck.

Too many open files.

You have too many open files (databases, reports) given the number of files you specified in the CONFIG.SYS FILES = statement. To prevent this error from recurring, either use fewer files or increase the number of files specified in the FILES = statement.

Missing or incorrect operand warnings.

The following warnings appear when the Formula Checker expects to find a specific kind of operand (the item on which a formula operation is to be performed), and finds something different. For example, the formula **`5>a`** is comparing a number to text (the old comparing apples to oranges analogy). When the Formula Checker sees that the number five is being compared to something, it expects that something to be another number. If anything other than a number appears, it displays the warning: *A number is required here.*

A Boolean array is required here.

A Boolean is required here.

A currency amount is required here.

A currency array is required here.

A currency range is required here.

A date array is required here.

A date is required here.

A date range is required here.

A number array is required here.

A number array or currency array is required here.

A number, currency amount, Boolean value, or string is expected here.

A number, currency amount, Boolean, date, or string is required here.

A number, currency amount, date, or string is required here.

A number, currency amount, or date is required here.

A number field or currency amount field is required here.

A number is required here.

A number or currency amount is required here.

A number range is required here.

A string array is required here.

A string is required here.

A string or an array of values is required here.

An array of values is required here.

Formulas in action

Formulas in action have been developed to demonstrate the use of multiple functions and operators in coordination with one another. The formulas have been created to illustrate concepts; they do not represent the only way, or necessarily the best way, to achieve the desired effects.

The following is a listing of complex formulas created to illustrate the use of various operators and functions. Each formula topic includes:

- a report scenario that describes a real-world reporting need
- a formula using Crystal syntax that fills the need described in the scenario
- a dissection of the formula so you can understand the role of each function and operator.

The names of the operators and functions used in the formula follow each numbered formula topic.

Click the formula of interest from the following list for further information:

Formula 1 (sales management, determining representatives closest to hitting quota)

Abs(x), Subtract (-), Divide (/), Multiply (*), ToText, Concatenate (+)

Formula 2 (form letter/extracting first purchase date from customer number, use date to calculate # of years as customer, and use result to customize letter)

If-then-else, Subscript [], Less Than (<), Concatenate (+), Make range (to), ToNumber, Subtract (-), ToText, Parentheses

Formula 3 (inventory analysis based on extracting inventory data from codes imbedded in item numbers)

Concatenate (+), ToText, ToNumber, Multiply(*), Make Range, Subscript [], Parentheses ()

Formula 4 (sales compensation, calculating commissions, flag commissions that exceed certain amount)

Nested formulas, If-then-else, Subtract (-), Multiply (*), Greater than (>), Greater than or equal (>=), Sum, Parentheses ()

Formula 5 (form letter, soliciting orders against available credit line)

Nested formulas, If-then-else, Subtract (-), Not equal to (<>), Less than (<), Concatenate (+), ToText, To dollar ($), Negate (-()), Parentheses ()

Formula 6 (calculating one value as percent of another, flag percentages outside range, disregard statistically insignificant percentages)

If-then-else, Greater than (>), Percentage (%), Greater than or equal (>=), Boolean Operator (And), Boolean Operator (Or), Parentheses ()

Formula 7 (sales compensation, selecting fixed bonus or calculated commission, whichever is higher)

If-then-else, Subtract (-), Greater than (>), Maximum, Multiply (*), Parentheses ()

Formula 8 (purchasing, determining quantity to order based on average sales during rolling quarter)

If-then-else, Less than (<), Negate (-), Add (+), Round, Average([array]), Parentheses ()

Formula 9 (retail, calculating mail order sales tax based on customer ZIP or Postal code)

Nested If-then-else expressions, NumericText, Subscript, Make Range, Equal to (=), ToNumber, In Range, Parentheses ()

Formula 10 (staff scheduling, flagging weekend incoming calls)

If-then-else, Not, DayOfWeek, Make range (to), In range (in), Parentheses ()

Formula 11 (calculating a contribution based on face value of invoice, and then selecting calculated value or agreed upon minimum)

If-then-else, Subscript [], Not equal to (<>), Maximum([array]), Multiply (*), Parentheses ()

Formula 12 (determining monthly compensation based on percent of dollars saved, and comparing result to negotiated maximum)

If-then-else, Average([array]), Subtract (-), Greater than (>), Minimum([array]), Multiply (*), Parentheses ()

Formula 13 (converting one unit of measure to another)

ToText, Truncate, Division (/), Concatenate (+), Remainder

Formula 14 (customer service, determining and identifying warranty plan based on length of product serial number)

If-then-else, Length, TrimLeft, Less than or equal (<=), Parentheses ()

Formula 15 (form letter, personalizing salutation based on degree and sex of recipient)

Nested If-then-else operators, Not equal (<>), Boolean operator (And), Equal (=), Concatenate (+), Parentheses ()

Formula 16 (shipping, calculating discounted value of shipment and adding calculated freight charge to orders that do not meet "free freight" criteria)

If-then-else, Add (+), Less than (<), Multiply (*)

Formula 17 (form letter, splitting a mailing list in half and sending a different offer to each half of the list)

If-then-else, Remainder, ToNumber, Equal, Parentheses ()

Formula 1

Functions/Operators Used

Abs(x), Subtract (-), Divide (/), Multiply (*), ToText, Concatenate (+)

Formula Purpose

As a sales manager with a large sales force, you want to identify those sales representatives who are the most consistent performers with regard to quota. You want to find those who come closest to hitting quota, regardless of whether they are slightly over or slightly under. It does not matter whether the variation is over or under quota; all that matters is the percent of variation from the mark.

Formula

```
ToText(Abs({file.QUOTA}-{file.SALES})/{file.QUOTA} * 100) + "%"
```

Result

Sales	Quota	Quota - Sales	% Variation
8, 000	10, 000	2000	20%
11, 000	10, 000	1000-	10%

Explanation

- The formula uses the Subtract operator (-) to subtract {file.SALES} from {file.QUOTA}. This gives the dollar variation from {file.QUOTA} (+ or -).
- The Abs(x) function converts the dollar variation to an absolute number, ignoring any + or - signs.
- It then uses the Divide operator (/) to divide that result by {file.QUOTA}. This gives the variation expressed as a decimal fraction.
- The formula then uses the Multiply operator (*) to multiply the result by 100 in order to calculate the final result in the form of a percentage.
- ToText is used to convert the calculated percent to text that can then be joined with other text.
- The Concatenate operator (+) is used to join the percentage, once converted to text, to the percent sign (%) character.

Related topics
Formulas in action

Formula 2

Operators/Functions Used

If-then-else, Subscript [], Less Than (<), Concatenate (+), Make range (to), ToNumber, Subtract (-), ToText, Parentheses ()

Formula Purpose

You have the date of first purchase coded as the fourth and fifth characters of the customer number (for example, 1971 as the 71 in ABC7101234, 1988 coded as the 88 in ABC880544, etc.) and you want to customize a letter to thank customers for the number of years they have done business with you. You want the following sentence to appear in your letter:

"You have been a valued customer for" [x] "years."
«Where x is the number of years.»

Formula

```
If {file.CUSTOMER NUMBER}[4 to 5] < "90" Then
"You have been a valued customer for " + ToText(91
- ToNumber({file.CUSTOMER NUMBER}[4 to 5]))
+ " years."
Else
"You are one of our newer customers, and we want you to know how valuable you
are to us."
```

Result

Customer #	Resulting Sentence
ABC7801234	"You have been a valued customer for 13 years."
ABD890337	"You have been a valued customer for 2 years."
ABD904331	"You are one of our newer customers, and we want you to know how valuable you are to us."

Explanation

- The If-then-else expression says, "If the 4th and 5th elements of the customer number, expressed as numbers, are less than 90, print a sentence including the date of first purchase, otherwise print the 'newer customer' sentence."
- The formula above uses the Subscript [ ] operator to extract the 4th and 5th characters (your date code) from the customer numbers which are stored as text in character fields. The Make Range operator (to) is used to establish the range 4 to 5.
- In the first example (ABC7801234) the 4th and 5th digits are 78 representing the year of first purchase as 1978.
- In the second example (ABD8903337), the 4th and 5th digits are 89 representing the year of first purchase as 1989.
- If the extracted characters are less than "90" (Then), a concatenated text string (a sentence) is printed that is customized to indicate the number of years the individual has been a customer. The text string says, "You have been a valued customer for (calculated number, expressed as text) years."
- The calculation of the number of years as a customer involves several steps:

- As was done earlier, the Subscript [ ] operator extracts the 4th and 5th characters (your date code) from the customer numbers which are stored as text in character fields. The Make Range operator (to) is used to establish the range 4 to 5.
- ToNumber converts the extracted date code to a number so it can be used in the calculation 91, where x = the date code expressed as a number.
- 91 subtracts the year of first purchase from 91 (the current year) to get the number of years the individual has been a customer.
- ToText then converts the result of that calculation back to text so it can be used in the expression "You have been a valued customer for (x) years."

- If the characters are "90" or more (Else), the fixed text string "You are one of our newer customers, and we want you to know how valuable you are to us" is printed.
- The Parentheses () control the order of calculation of the formula.

Related topics

Formulas in action

Formula 3

Operators/Functions Used

Concatenate (+), ToText, ToNumber, Multiply (*), Make Range (to), Subscript [], Parentheses ()

Formula Purpose

A sail maker, as part of his loan agreement with his bank, has to submit a detailed inventory analysis monthly. The analysis must include the cost of fabric in raw material inventory and the cost of fabric by item number for each item in finished product inventory.

The company uses one fabric for all of the sails it produces, and it uses the 5th and 6th characters in the item number for each product to represent the number of meters of material (rounded to the nearest meter) necessary to make that item.

In the form letter the manager sends to his banker each month, he wants the computer to automatically insert the quantity on hand, the item number, and the dollar value of the fabric for each item number.

Formula

```
ToText({file.QUANTITY}) + " each, Item " + {file.ITEM} +
", $ " + ToText({file.QUANTITY} * ToNumber({file.ITEM}[5
to 6]) * {file.FABRIC COST})
```

Result

With a fabric cost of $14.88/meter, the formula delivers the following result:

Quantity	Item	Letter Text
46	4423141006	"46 each, Item 4423141006, $ 9582.72"
27	4423081009	"27 each, Item 4423081009, $ 3214.08"

Explanation

- This equation uses ToText to convert the value of {file.QUANTITY} (a numeric field) to text so it can be used as part of a concatenated text string.
- It then uses the Concatenate operator (+) to join the text version of {file.QUANTITY} with the string " each, Item ".
- It again uses the Concatenate operator (+) to join the resulting string with the item number, stored as the value of the text object {file.ITEM}.
- It uses the Concatenate operator (+) one final time to join the resulting text string to the calculated fabric cost (expressed as text).
- To calculate that fabric cost:

- The formula uses the Subscript operator [] to extract the 5th and 6th elements (meters of material used) of {file.ITEM} (a text object).
- The Make Range operator (to) is used to establish the range 5 to 6.
- The ToNumber function converts those elements to a number so it can be used in a numeric calculation.
- It uses the Multiply operator (*) to multiply that number by the value of {file.FABRIC COST} by the number of units in inventory ({file.QUANTITY}) to arrive at the total price.
- Finally, it uses the ToText function to convert the total price (a number) into text that can be used in the concatenated string.

Related topics
Formulas in action

Formula 4

Operators/Functions Used

Nested formulas, If-then-else, Subtract (-), Multiply (*), Greater than (>), Greater than or equal (>=), Sum, Parentheses ()

Formula Purpose

A computer store sells hardware, software, and books. It pays its sales representatives 7% of all hardware sales (monthly) over $5000, 10% of all software sales (monthly) over $10, 000, and 5% of all book sales (monthly) over $1000. The sales manager wants to calculate the commission in each category for each representative, total the commission due each representative, and flag those representatives who are entitled to more than $5000 in total commission for the month. We will do this using nested formulas, that is, using one formula as one of the elements in another formula.

Formula(s)

@HARDWARE

If (({file.HARD SALES}000)> 0) Then
.07 * ({file.HARD SALES}000)
Else
0

@SOFTWARE

If (({file.SOFT SALES}0000)> 0) Then
.10 * ({file.SOFT SALES}0000)
Else
0

@BOOKS

If (({file.BOOK SALES}000)> 0) Then
.05 * ({file.BOOK SALES}000)
Else
0

@TOTCOMM

Sum([{@HARDWARE}, {@SOFTWARE}, {@BOOKS}])

@FLAG

If {@TOTCOMM} >= 5000 Then
" **** "
Else
""

Result

Given this data:

Salesrep	HardSales	SoftSales	BookSales
Salesrep A	4500	21000	985
Salesrep B	31427	41222	4470
Salesrep C	22000	4687	4250
Salesrep D	14000	15678	2200

The formulas return the following results:

Salesrep	Hardware	Software	Books	Total	Flag
SalesrepA	0.00	1100.00	0.00	1100.00	
SalesrepB	3122.20	1849.89	173.50	5145.59	****
SalesrepC	1190.00	0.00	162.50	1352.50	
SalesrepD	630.00	567.80	60.00	1257.80	

Explanation

- The three formulas (@HARDWARE, @SOFTWARE, and @BOOKS) work in the same manner.
- They use the If-then-else operator to test for a condition, do one thing if the condition is true, do another thing if the condition is false.
- They take total sales in the category, use the Subtract operator (-) to subtract the amount of sales on which no commission is to be paid, and test to see if the remaining amount is a positive number (>0).
- If it is (>0), they calculate the commissions using the Multiply operator (*) to multiply the appropriate commission percentage (expressed as a decimal: .07, .10, .05) times the commission-table amount ({file.HARD SALES}000}, etc.).
- If the remaining amount is a negative number (<0), no commission is computed and the formula prints a zero amount (0.00).
- @TOTCOMM uses the Sum function to total the commissions due. Instead of including the calculations for each formula (a duplication of time and effort) it substitutes the formula name instead of the calculations. When the program encounters the formula name it knows to use the underlying calculations from the formula(s) referenced.
- @FLAG uses the If-then-else operator to evaluate the values calculated by @TOTCOM and to flag (****) those values of $5000 or more (>= 5000). @FLAG effectively nests two levels of other formulas: it nests @TOTCOM, which is a formula that itself nests three other formulas, @HARDWARE, @SOFTWARE, and @BOOKS. When the program sees @TOTCOM, it performs all of the underlying calculations referenced by that formula and the formulas that are used in @TOTCOM.
- Parentheses () are used throughout the formulas to control the order of calculation.

Related topics

Formulas in action

Formula 5

Operators/Functions Used

Nested formulas, If-then-else, Subtract (-), Not equal to (<>), Less than (<), Concatenate (+), ToText, To dollar ($), Negate (-()), Parentheses ()

Formula Purpose

A company wants to stimulate new credit sales for those customers with open credit limits and to make customers who are over their limits aware of their over-limit balances. To do this the company wants to include a brief "PS" to its already customized (company name, contact name, etc.) sales letter to each customer.

Formula

```
If {file.CREDIT LIMIT} - {file.BALANCE} <> 0 Then
If {file.CREDIT LIMIT} - {file.BALANCE} <0 Then
"Your account is currently $" +
ToText($ (-{file.CREDIT LIMIT}
- {file.BALANCE}))
+ " over limit. Please contact us if you would
like to discuss an increase in your credit limit."
Else
"Your account has $" + ToText($({file.CREDIT LIMIT}
-{file.BALANCE})) + " available credit.
You can order today with no additional paperwork!"
Else
""
```

Result

CreditLimit	Balance	Result
$5000	$2250	"Your account has $2750.00 available credit. You can order today with no additional paperwork!"
$3000	$3457	"Your account is currently $457.00 over limit. Please contact us if you would like to discuss an increase in your credit line."
$7500	$7500	«no message»

Explanation

- The formula uses two If-then-else operators, one nested inside the other.
- The first begins with the first word If and does not end until the Else at the very end of the expression. This expression says, essentially, "If the credit limit less the balance is less than zero, then print the message based on the If-then-else expression that follows (the expression inside the parentheses). If the credit limit less the balance equals zero, print nothing. Think of this first If-then-else expression in this way: If {file.CREDIT LIMIT} - {file.BALANCE} <> 0 then (do what is in the parentheses) Else "".
- The second If-then-else expression begins "If {file.CREDIT LIMIT}" and ends after the word "paperwork!". This expression says: "If the credit limit less the balance is less than zero, then print the over limit message, otherwise (in those cases where the result is greater than zero) print the available credit message.
- The Then expression ToText($(- {file.CREDIT LIMIT} - {file.BALANCE})) means:
- first use the Subtract operator (-) to subtract {file.BALANCE} from {file.CREDIT LIMIT}.

- Since this part of the expression will only be activated if the result is a negative number (<0), the formula uses the Negate operator (-()) to multiply the result by to return a positive number.
- The To Dollar operator ($) assures that the result will be printed in a dollar and cents format with two decimal places.
- The ToText function takes the resulting number and converts it to text characters so it can be used in the over limit message.
- The Else expression ToText($({file.CREDIT LIMIT}-{file.BALANCE})) differs only slightly from the Then expression above. This expression does not use the Negate operator (-()) because this expression {file.CREDIT LIMIT}-{file.BALANCE} will be used only in those cases where the result is a positive number (>0).
- The Concatenate operator (+) joins the text strings (enclosed in quotation marks) with the number (converted to text using the ToText function) to produce the appropriate message (the Then message [over limit] or the Else message [available credit]).
- Many sets of parentheses are used to control the order of calculation of this formula.
- The characters "" at the very end of the formula are the representation of an empty string. This means print nothing.

Comments

Note the leading space in the string " over limit." Note too, the leading space in the string " available credit." These spaces are purposely entered into the formula so a space occurs between the dollar figure and the words in each message.

Related topics
Formulas in action

Formula 6

Operators/Functions Used

If-then-else, Greater than (>), Percentage (%), Greater than or equal (>=), Boolean Operator (And), Boolean Operator (Or), Parentheses ()

Formula Purpose

The manager of a minor league baseball team wants a column on the statistics report that flags all batting averages of .300 or better and all averages below .200. The manager does not want the batting average flagged for any player who has batted fewer than 100 times because the manager does not feel that such a batting average is statistically significant.

Formula

```
If ({file.ATBAT}>=100) And
(({file.HITS} % {file.ATBAT}>=30) Or
({file.HITS} % {file.ATBAT}<20)) Then
"####"
Else
""
```

Result

Hits	AtBat	Average	Flag	Explanation
31	98	.316		AtBat <100 (not significant)
31	101	.307	####	Average > 30 (.300)
43	216	.199	####	Average < 20 (.200)
19	99	.192		AtBat < 100 (not significant)

Explanation

- The formula uses the If-then-else operator to test for specific conditions.
- Condition A ({file.ATBAT} >= 100) uses the Greater than or equal operator (>=) to make certain that the batter has batted at least one hundred times. If he has batted 100 or more times, this condition is true, otherwise it is false.
- Condition B ({file.HITS} % {file.ATBAT} >30) uses the Percentage operator (%) to calculate {file.HITS} as a percentage of {file.ATBAT}. It then uses the Greater than or equal operator (>=) to test the resulting percentage to see if it is 30% (.300 batting average) or more. If the percentage is greater than or equal to 30, this condition is true, otherwise it is false.
- Condition C ({file.HITS} % {file.ATBAT} < 20) again uses the Percentage operator (%) to calculate {file.HITS} as a percentage of {file.ATBAT}. It then uses the Less than operator (<) to test the resulting percentage to see if it is less than 20% (.200 batting average). If the percentage is less than 20, this condition is true, otherwise it is false.
- The formula uses the Boolean operators And, and Or to evaluate conditions A, B, and C. The logical relationship required is Condition A and either Condition B or Condition C. Thus, the If part of the formula is satisfied if Condition A is TRUE, and either Condition B or Condition C is TRUE.
- If the If part of the formula is satisfied (Then), it flags the batting average by printing four pound signs "####".
- If those conditions are not true (Else), it prints nothing (as indicated by the empty text string "").

Related topics

Formulas in action

Formula 7

Operators/Functions Used

If-then-else, Subtract (-), Greater than (>), Maximum ([array]), Multiply (*), Parentheses ().

Formula Purpose

A company has the following bonus/commission structure for its sales force: when a sales representative sells at or over quota, the representative earns a $250 bonus or a 15% commission (on the amount over quota), whichever is highest. The company pays no bonus or commission on sales less than quota. The sales manager wants bonus/commission calculated and included on a report.

Formula

If ({file.SALES}-{file.QUOTA})>=0 Then
Maximum([250,.15*({file.SALES} - {file.QUOTA})])
Else
0

Result

Sales	Quota	5%	Amount Paid
10,000	8000	300	540 (calculated commission)
8000	8000	0	250 (guaranteed bonus)
7999	8000	0	0 (sales not >= Quota)

Explanation

- This formula uses the If-then-else operator to test sales to see if they are greater than or equal to quota.
- If they are (Then) it calculates the commission and pays the higher of commission or guaranteed bonus.
- If they are not (Else), it does nothing.
- To test sales to see if they are at least equal to quota, the formula uses the Subtract operator (-) to subtract {file.QUOTA} from {file.SALES}.
- If {file.SALES} is greater than or equal to {file.QUOTA}, this subtraction will produce the result of zero or a positive number (>=0). This will trigger the Then part of the formula.
- If {file.SALES} is less than {file.QUOTA}, this subtraction will produce a negative result. This will trigger the Else part of the formula. The Then part of the formula uses the Multiply operator (*) to compute a 15% commission on the amount of sales in excess of quota: .15 * ({file.SALES} - {file.QUOTA}).
- It then uses the Maximum([array]) function to determine if the calculated commission or the guaranteed bonus of $250 offers the highest payout Maximum([commission, 250]), and it returns this highest value.
- Calculating this highest payout is the bottom line of the Then part of the formula.
- The Else part of the formula makes no calculations and returns 0.

Related topics
Formulas in action

Formula 8

Operators/Functions Used

If-then-else, Less than (<), Negate (-), Add (+), Round, Average([array]), Parentheses ().

Formula Purpose

To better control her purchasing and inventory, a store owner wants the inventory report to include a QtyToOrder column. She wants that column to tell her what quantity she needs to order to keep on hand the average quantity sold monthly during the previous three months.

- If there is stock on hand, she will want to order the average quantity sold less the stock on hand.
- If there is no stock on hand but no backorders, she will want to order the average quantity sold.
- If there is no stock on hand and backorders, she will want to order enough to clear the backorders plus the average quantity sold.

Formula

```
If {file.ONHAND} < 0 Then
((- ({file.ONHAND})) + Round
((Average([{file.MONTH1}{file.MONTH2},
{file.MONTH3}]))))
Else
(Round((Average([{file.MONTH1}{file.MONTH2},
{file.MONTH3}])) - {file.ONHAND}))
```

Result

OnHand	Month1	Month2	Month3	Avg.	Order
2	16	6	30	17	15
5-	7	3	19	10	15
0	9	8	18	12	12

Explanation

- The formula uses the If-then-else operator to set up one set of calculations if there are backorders ({file.ONHAND} < 0), and a different set of calculations if there are no backorders.
- The If part of the formula uses the Less than operator (<) to see if the quantity on hand is less than zero. This indicates that, not only is all stock gone but there are unfilled orders (a backorder situation).
- If there is a backorder, the Then part of the formula calculates the amount to order as the average monthly sales over a specified three month period plus the amount backordered (expressed as a positive number).
- It uses the Negate operator (-) to convert the negative value of {file.ONHAND} to a positive value.
- It uses the Average([array]) function to average the product sales for {file.MONTH1}, {file.MONTH2}, and {file.MONTH3} (the three months used to determine a typical sales pattern for the product).
- It uses the Round function to round the average product sales to the nearest whole number.

- It uses the Add operator (+) to add the quantity on hand (now expressed as a positive) to the rounded average monthly sales figure.
- The result is the amount to order.
- If there is no backorder, the Else part of the formula calculates the amount to order as the average monthly sales over the specified three month period less the quantity already on hand.
- It uses the Average function to average the product sales for {file.MONTH1}, {file.MONTH2}, and {file.MONTH3} (the three months used to determine a typical sales pattern for the product).
- It uses the Round function to round the average product sales to the nearest whole number.
- It uses the Subtract operator (-) to subtract the quantity on hand from the rounded, average monthly sales figure.
- The result is the amount to order.

Related topics

Formulas in action

Formula 9

Operators/Functions Used

Nested If-then-else expressions, NumericText, Subscript, Make Range, Equal to (=), ToNumber, In Range, Parentheses ()

Formula Purpose

A retailer has a customer list that includes both customers living inside the state and customers living in Canada. For customers inside the state, some have 5 digit ZIP codes, others have 9 digit ZIP codes. Canadian customers have Postal Codes beginning with a letter.

The company wants a column on the list that shows the sales tax that needs to be levied on orders from each customer. Local customers (those within the greater metropolitan area) get assigned a 7.5% sales tax (6.5% state and 1% local); customers in the state but outside the metropolitan area are assigned a 6.5% sales tax (state only); and Canadian customers are assigned no sales tax.

Formula

```
If NumericText({customer.POSTAL CODE}[1 to 5])Then
If ToNumber({customer.POSTAL CODE}[1 to 5]) in
92200 to 92399 Then
"7.5"
Else
"6.5"
Else
"0"
```

Result

ZIP/Postal	Tax	Explanation
91134	6.5	Inside state/outside metro area
92305	7.5	Inside state/inside metro area
92288423	7.5	Inside state/inside metro area (ZIP + 4)
96544333	6.5	Inside state, outside metro area (ZIP + 4)
T5A 9S2	0.0	Canadian customer

Explanation

- This formula uses nested If-then-else operators.
- The first of these If-then-else expressions begins with the initial If and ends with the final "0". It says:
- If the first five characters in the {customer.POSTAL CODE} field are all numbers, then compute and print the sales tax using the methodology specified in the second If-then-else expression.
- If the first five characters are not all numbers, print "0".
- The second of the If-then-else expressions begins immediately after the initial Then and ends immediately before the final words Else "0". It says, now that we have already tested and know that the first five characters of {customer.POSTAL CODE} are numbers:
- If the first five characters of {customer.POSTAL CODE}, converted to numbers, fall in the range of 92200 to 92399, print "7.5" (the amount of state and local sales taxes combined).
- If the first five characters of {customer.POSTAL CODE}, converted to numbers, fall outside that range, print "6.5" (the amount of state sales tax only).
- If NumericText ({customer.POSTAL CODE} [1 to 5]) uses the If-then-else operator to test for the condition in parentheses:
- {customer.POSTAL CODE} [1 to 5]) uses the Subscript operator [] and Make Range operator to extract characters 1 to 5 in the {customer.POSTAL CODE} field.
- The NumericText function evaluates the extracted characters to determine if they are all numbers or not.
- If they are all numbers (YES) the first If condition is satisfied and the formula moves to the first Then consequence.
- If they are not all numbers, the first If condition is not satisfied, and the formula moves to the final Else consequence.
- Then if ToNumber({customer.POSTAL CODE} [1 to 5]) in 92200 to 92399 Then "7.5" Else "6.5" Else "0" shows what is to take place if the original If conditions have been satisfied.
- ({customer.POSTAL CODE} [1 to 5]) uses the Subscript operator [ ] and Make Range operator to extract characters 1 to 5 in the {customer.POSTAL CODE} field.
- ToNumber converts these extracted characters to a number that can then be used in a numeric expression.

- in 92200 to 92399 checks to see if the {customer.POSTAL CODE} (now converted to a number) falls anywhere within the range 92200 to 92399.
- If it does, the If condition (inside the parentheses) is satisfied and the Then consequence is performed by printing the text string "7.5".
- If it does not, the If condition is not satisfied, and the Else consequence is performed by printing the text string "6.5".

Comments

These tax figures, rules, and Postal codes are for illustration only; they are not meant to accurately represent state and local taxing regulations.

Related topics
Formulas in action

Formula 10

Operators/Functions Used

If-then-else, Not, DayOfWeek, Make Range (to), In Range (in), Parentheses ()

Formula Purpose

A seven day per week customer service department logs all calls and stores the records in a database. As an aid in scheduling weekend staff, management wants to flag the weekend calls so they stand out in the call report.

Formula

```
If Not(DayOfWeek({file.CALL DATE}) in 2 to 6) Then
"Weekend"
Else
""
```

Result

Day of Week	#	Flag
Sunday	1	Weekend
Monday	2	
Tuesday	3	
Wednesday	4	
Thursday	5	
Friday	6	
Saturday	7	Weekend

Explanation

- The formula uses the If-then-else operator to say, "If the day of the week is not a weekday, then print "Weekend", otherwise (Else) print nothing (as indicated by the empty text string "").
- The If part of the formula sets up the condition, "If the day of the week is not a weekday".
- The DayOfWeek function evaluates {file.CALL DATE} and returns a number from 1 to 7 (Sunday being 1, Saturday being 7) to indicate the day of the week on which a call was made.
- In 2 to 6 uses the In Range function (in) and the Make Range function (to) to determine if the day of the week the call was made was in the range 2 to 6 (Monday to Friday).
- The Not function negates the expression that follows.
- Without the Not function preceding the expression, the If expression reads "If the day of the week number indicates the call date was a weekday."
- With the Not function, the If expression reads, "If the day of the week number indicates the call date was not a weekday."
- If the call date is not a weekday, the If expression is satisfied, thus triggering the Then consequence.
- "Weekend" tells the program that when the If condition is satisfied, print the word "Weekend".
- If the call date is a weekday, the If condition is not satisfied, thus triggering the Else consequence.
- "" tells the program to print nothing if the If condition is not satisfied.

Related topics
Formulas in action

Formula 11

Operators/Functions Used

If-then-else, Subscript [], Not equal to (<>), Maximum([array]), Multiply (*), Parentheses ().

Formula Purpose

As part of a charity fund raiser, a company agrees to donate 1% of the invoice amount (or $1.00, whichever is larger) for each invoice it cut during the previous quarter. The only invoices that it has exempted are Credit Invoices (identified with the letter "C" as the 6th character in the invoice number) and invoices for non-profit agencies (identified with the letter "N" as the 6th character in the invoice number). Management wants a Contribution column to appear on the quarterly detail sales report.

Formula

```
If {file.INV#}[6]<> "C" and {file.INV#}[6]<>"N" Then
$(Maximum([.01 * {file.AMT}), 1])
Else
0
```

Result

Invoice #	Amount	Amt. * 1%	To Charity	Reason
21523R	143.27	1.43	$1.43	
21524C	223.46	N/A	0	«credit»
21538R	47.15	.47	$1.00	
21575N	1312.49	N/A	0	«nonprofit»

Explanation

- The If part of the expression tests for two conditions: that the invoice is not a credit invoice and that it is not a non-profit invoice. Both conditions must be true to trigger the Then consequence.
- Condition A: {file.INV#} [6] < > "C" uses the Subscript operator [] to extract the 6th character of the value stored in {file.INV#}. It then compares that character to "C". If the character is not C (not a credit invoice), Condition A is satisfied.
- Condition B: {file.INV#} [6] < > "N" uses the Subscript operator [] to extract the 6th character of the value stored in {file.INV#}. It then compares that character to "N". If the character is not N (not a non-profit invoice), Condition B is satisfied.
- The Boolean operator and indicates that both Condition A and Condition B must be TRUE in order to satisfy the If part of the expression.
- If Condition A and Condition B are both TRUE, the If part of the expression is satisfied, thus triggering the Then consequence.
- .01 * {file.AMT} uses the Multiply operator (*) to multiply the amount of the invoice by 1% (.01).
- Maximum([.01 * {file.AMT}, 1]) returns the highest contribution for a given invoice: either 1% of the invoice amount or $1.00.
- The To Dollar ($) operator converts the amount to a dollar value.
- If either Condition A or Condition B is FALSE, or both are FALSE, the If part of the expression is not satisfied, thus triggering the Else consequences.
- If the If part of the expression is not satisfied, the formula returns the value 0.

Comments

An alternative rendering follows:

```
If {file.INV#}[6] in ["C", "N"] Then
0
Else
Maximum([.01*{file.AMT}, 1])
```

Related topics

Formulas in action

Formula 12

Operators/Functions Used

If-then-else, Average([array]), Subtract (-), Greater than (>), Minimum([array]), Multiply (*), Parentheses ().

Formula Purpose

A consultant has contracts that pay him a percentage of the amount he saves his clients with a cap or ceiling on the amount he can earn. His contracts pay him 25% of the monthly savings up to a maximum of $10,000 per month. If his percentage amounts to more than $10,000, he collects the $10,000 maximum; if the percentage is less than $10,000, he collects the actual percentage. The base period against which savings are calculated is the average of three typical months in a prior year.

Formula

```
If (Average([{file.MON1}, {file.MON2}, {file.MON3}]) -
{file.CURRENTMON})> 0 Then
Minimum([.25*((Average([{file.MON1}, {file.MON2},
{file.MON3}) - {file.CURRENTMON}]), 10000])
Else
0
```

Result

Average	CurMon	Savings	25%	Amt.Due
90, 000	65, 000	25, 000	6250	6250
120, 000	60, 000	60, 000	15, 000	10, 000
75, 000	77, 000	N/A	N/A	0

Explanation

- The formula uses the If-then-else operator to say: if the average outlay during the three months specified was higher than the current month, then return 25% of the difference, up to a maximum of $10,000. If the average outlay was equal to or less than the outlay during the current month, return 0.
- The formula uses the Average function to calculate the average outlay during {file.MON1}, {file.MON2}, and {file.MON3} - the baseline period.
- It uses the Subtract operator (-) to subtract the outlay in the current month from the average during the baseline period, thus giving the difference between the current month and the baseline.
- It uses the Greater than operator (>) to compare that difference to 0.

- If the difference is greater than 0 (there actually was a savings), the If part of the expression is satisfied, thus triggering the Then consequences.
- If the Then consequences are triggered:
- The formula uses the Average function to calculate the average outlay during {file.MON1}, {file.MON2}, and {file.MON3} - the baseline period.
- It uses the Subtract operator (-) to subtract the outlay in the current month from the average during the baseline period.
- It uses the Multiply operator (*) to calculate 25% of the difference (.25 *...).
- It uses the Minimum ([array]) function to return the smaller amount: either 25% of the difference, or $10,000. Using the Minimum function in this way effectively puts a ceiling on the result of the calculation.
- If the difference is equal to or less than 0 (there was no savings), the If part of the expression is not satisfied, thus triggering the Else consequence.
- The Else consequence (Else 0) simply returns the value 0.

Related topics

Formulas in action

Formula 13

Operators/Functions Used

ToText, Truncate, Division (/), Concatenate (+), Remainder

Formula Purpose

Bulk grains and nuts are inventoried by the ounce, but management wants to see a breakdown in pounds and ounces on the inventory report.

Formula

ToText((Truncate({file.OZ}/16)),0) + " pounds, " +
ToText((Remainder ({file.OZ}, 16)),0) + " ounces "

Result

Ounces	Resulting Text
433	"27 pounds, 1 ounces"
278	"17 pounds, 6 ounces"
1455	"90 pounds, 15 ounces"

Explanation

- The formula uses the Divide operator (/) to divide {file.OZ} by 16, thus calculating the number of pounds of the item in inventory. Unless {file.OZ} is perfectly divisible by 16, the quotient will be a whole number with decimal places.
- The Truncate function returns only the whole number (integer) part of the quotient.
- The ToText function converts the number to text so it can be used in a string with other text, and formats the number to zero decimal places (the second parameter).
- The formula uses the Concatenate operator (+) to connect the number of pounds with the text string " pounds" to give the new string " n pounds" (where n is the calculated number of pounds).
- The Remainder function determines the remainder that results from the division {file.OZ}/16 ({file.OZ} the numerator, 16 the denominator). The remainder is a number of ounces less than 16 (less than a full pound).
- The ToText function converts the remainder to text so it can be included in the final text string, and formats the number to zero decimal places.
- The Concatenate operator (+) connects the number of ounces (now converted to text, to the string before ("n pounds,") and to the string after ("ounces") to produce the final text string "n pounds, y ounces" (where y is the remainder).

Related topics
Formulas in action

Formula 14

Operators/Functions Used

If-then-else, Length, TrimLeft, Less than or equal (<+), Parentheses ().

Formula Purpose

Five and a half years earlier, a company had changed from a lifetime warranty for its products to a five year warranty. The first products that had been manufactured with the shorter warranty were now out of warranty, but the repair department was slow to adapt. It continued to repair all products for free, as if they were all covered by the original lifetime warranty.

All products manufactured under the new, shorter warranty had issued eight character serial numbers instead of the five, six, or seven character serial numbers that had been issued to lifetime-warranted products.

To begin controlling the situation, management has called for a column on its repair report that identifies products serviced as either lifetime warranty or five year warranty products.

Formula

```
If Length(TrimLeft({file.SERIAL#}))<= 7 Then
"Lifetime Warranty"
Else
"5 Year Warranty"
```

Result

Serial Number	# Characters	Flag
BP10001	7	"Lifetime Warranty"
BP1000	6	"Lifetime Warranty"
BP999	5	"Lifetime Warranty"
BP100001	8	"5 Year Warranty"

Explanation

- The formula uses the If-then-else operator to say, "If the serial number is 7 characters long or less (Then), print "Lifetime Warranty", otherwise (Else) print "5 Year Warranty".
- TrimLeft({file.SERIAL#}) removes all of the blank spaces stored to the left of the actual serial number in the right-justified {file.SERIAL#} field. The formula uses TrimLeft to eliminate spaces because Length counts spaces as characters if they are present.
- Length counts the characters in the actual serial number.
- The formula uses the Less than or equal to operator (<=) to make certain that the serial number is 7 characters long (or less).
- If the serial number is less than or equal to 7 characters, the If part of the expression is satisfied, thus triggering the Then consequence.
- If the If part of the expression is satisfied, the formula prints the text string "Lifetime Warranty".
- If the serial number is greater than 7 characters, the If part of the expression is not satisfied, thus triggering the Else consequence.
- If the If part of the expression is not satisfied, the formula prints the text string "5 Year Warranty".

Related topics

Formulas in action

Formula 15

Operators/Functions Used

Nested If-then-else operators, Not equal (<>), Boolean operator (And), Equal (=), Concatenate (+), Parentheses ().

Formula Purpose

The director of a non-profit health care facility wants to automate the salutation in a letter he is sending to his staff. The staff consists of men and women, and professional and non-professional employees. He wants to be certain that all doctors on staff (M.D., Ph.D., and D.D.S.) are given the form of address "Dr." and all non-doctors are given the form of address appropriate to their gender.

Formula

```
If {file.DEGREE} <> "Ph.D." And
{file.DEGREE} <>"M.D." And
{file.DEGREE} <>"D.D.S" Then
If {file.SEX} = "M" Then
"Dear Mr. " + {file.LNAME}
Else
"Dear Ms. " + {file.LNAME}
Else
"Dear Dr. " + {file.LNAME}
```

Result

Degree	Gender	Last Name	Salutation
B.A.	F	Jones	"Dear Ms. Jones"
Ph.D.	F	Smith	"Dear Dr. Smith"
M.D.	M	Jackson	"Dear Dr. Jackson"
M.S.	M	Miller	"Dear Mr. Miller"
D.D.S.	F	Johnson	"Dear Dr. Johnson"

Explanation

- This formula uses nested If-then-else operators.
- The first If-then-else expression begins with the first If and continues to Else "Dear Dr. " + {file.LNAME} at the end of the formula. It says, "If the degree listed is not a doctoral degree (Then), follow the Then consequences, which contains another If-then-else expression. If the degree listed is a doctoral degree (Else), print a doctoral salutation.
- The If part of the expression tests for three different conditions.
- Condition A uses the Not equal operator <> to make certain that the employee's degree is not Ph.D.
- Condition B uses the Not equal operator <> to make certain that the employee's degree is not M.D.
- Condition C uses the Not equal operator <> to make certain that the employee's degree is not D.D.S.
- The two And operators indicate that all three conditions must be met to satisfy the If part of the expression.
- If all three conditions are met, the If part of the expression is satisfied, thus triggering the Then consequences.
- If any one of the three conditions is not met (or two or all three), the If part of the expression is not satisfied, thus triggering the Else consequence.
- The second If-then-else statement begins with if {file.GENDER} = and ends with Else "Dear Ms. " + {file.LNAME}. It says, "If the employee is male (then), print a male salutation. If the employee is anything but male (Else), print a female salutation. This If-then-else expression determines what actually happens if the If part of the first expression is

satisfied.

- If the gender is male (determined by using the Equal operator =), the If part of the second If-then-else expression is satisfied, thus triggering the Then consequence (printing the salutation "Dear Mr. " + {file.LNAME}.
- If the gender is not male, the If part of the second If-then-else expression is not satisfied, thus triggering the Else consequence (printing the salutation "Dear Ms. " + {file.LNAME}.
- The Concatenate operator (+) connects the "Dear x" part of the salutation with the last name (as stored in the {file.LNAME} field).

Related topics

Formulas in action

Formula 16

Operators/Functions Used

If-then-else, Add (+), Less than (<), Multiply (*)

Formula Purpose

A manufacturer of lawn and garden products grants a 10 discount on Category A products and 150 discount on Category B products. It also pays the freight on all orders of $5,000 or more (before discount). It charges a flat 4% freight on all orders of less than $5,000. Management wants a Total Including Freight figure to appear on the daily sales report.

Formula

```
If ({file.CAT A} + {file.CAT B}) < 5000 Then
1.04 * .95 *.95 *.90 *{file.CAT A} + .95 *.90 *.85
*{file.CAT B}
Else
.95 *.95 *.90 *{file.CAT A} + .95 *.90 *.85 *{file.CAT B}
```

Result

Cat A	Cat B	Cat A+Cat B	TotInclFrt
9524	1344	10868	8712.62
3424	1344	4768	3908.21

Explanation

- This formula uses the If-then-else operator to say, if the sum of {file.CAT A} and {file.CAT B} is lower than $5,000 (Then), multiply the discounted price by 104% (1.04) to arrive at the price plus freight. If the sum of {file.CAT A} and {file.CAT B} is higher than $5,000 (Else), simply calculate the discounted price (no freight, since the order is bigger than

$5,000.

- The If part of the expression uses the Add operator (+) to calculate the undiscounted value of {file.CAT A} and {file.CAT B}.
- It uses the Less than operator (<) to determine if the sum of {file.CAT A} and {file.CAT B} is less than $5000.
- If the sum is less than $5000, the If part of the expression is satisfied, thus triggering the Then consequence.
- The formula uses the Multiply operator (*) several times, first to multiply the value of {file.CAT A} by .90 (10% discount), to multiply that result by .95 (5% discount), and to multiply that result by .95 (5% discount) to arrive at the discounted amount for {file.CAT A}.
- It performs the same set of calculations on {file.CAT B} to determine the discounted amount for that category.
- It uses the Add operator (+) to add the discounted amounts of {file.CAT A} and {file.CAT B} to arrive at the discounted total (before freight).
- Finally, it uses the Multiply operator (*) to multiply the discounted total by 1.04 (100% + 4% freight) to arrive at the invoice total including freight.
- If the sum is $5,000 or more, the If part of the expression is not satisfied, thus triggering the Else consequence.
- The formula uses the Multiply operator (*) several times, first to multiply the value of {file.CAT A} by .90 (10% discount), to multiply that result by .95 (5% discount), and to multiply that result by .95 (5% discount) to determine the discounted amount for {file.CAT A} (freight free).
- It performs the same set of calculations on {file.CAT B} to determine the discounted amount for that category.
- Finally, it uses the Add operator (+) to add the discounted amounts of {file.CAT A} and {file.CAT B} to arrive at the final invoice total (freight free).

Related topics

Formulas in action

Formula 17

Operators/Functions Used

If-then-else, Remainder, ToNumber, Equal (=), Parentheses ()

Formula Purpose

A retailer wants to test two different direct mail offers to see which one has the best "pulling" power. He wants to send one offer to all customers on his mailing list with even customer numbers and the second offer to all customers with odd customer numbers.

Formula

```
If Remainder(ToNumber({file.CUSTNUM}), 2 )= 0 Then
{file.OFFER1}
Else
{file.OFFER2}
```

Results

Cust #	Remainder	Result
203104	0	Offer 1 - "You are entitled to a 10% discount."
203105	1	Offer 2 - "You are entitled to a free gift."

Explanation

- The ToNumber function converts the customer number (stored as text) to a number.
- The Remainder function takes that number, divides it by 2, and returns a remainder.
- The Equal operator tests the remainder to see if it is equal to 0.
- If the remainder is equal to zero, the customer number is divisible by two and thus an even number triggering the printing of {file.OFFER1}.
- If the remainder is not equal to zero, the customer number is not evenly divisible by two and thus an odd number, triggering the printing of {file.OFFER2}.

Related topics

Formulas in action

Creating Formulas with Basic Syntax

This section provides you with an overview of various parts of a formula. You learn about Basic syntax and the techniques you can use when creating a formula.

If you are new to Basic, begin with these topics:

- Basic syntax overview
- Basic syntax fundamentals.

Basic syntax overview

When creating formulas, you have the option of using either Crystal or Basic syntax. Syntax rules are used to create correct formulas. Almost any formula written with one syntax can be written with the other. Reports can contain formulas that use Basic syntax as well as formulas that use Crystal syntax.

If you are familiar with Microsoft Visual Basic or other versions of Basic, then Basic syntax may be more familiar to you. In general, Basic syntax is similar to Visual Basic except that it has specific extensions to handle reporting.

Note: Report processing is not slowed down by using Basic syntax. Reports using Basic syntax formulas can run on any machine that Crystal Reports runs on. Also, using Basic syntax formulas does not require distributing any additional files with your reports.

For more details, see Tips on learning Basic syntax.

Tips on learning Basic syntax

If you have no programming experience

In many cases you may not need to use the formula language. Crystal Reports includes several Experts that automatically handle situations where formulas could be used. These include the Select, Search, Running Totals and Highlighting Experts, and the Insert Summary and Insert Grand Total dialog boxes. Before creating formulas, check to see if you can use one of these tools.

However, you may need to create formulas without the help of an Expert. Read this section to learn about Basic syntax and the rules you need to follow to create a formula.

If you know Microsoft Visual Basic, VBScript, or another version of Basic

In order to create formulas using Basic syntax, you need to understand the following:

- How a Basic syntax formula refers to other fields in the report such as database fields, parameter fields, summary fields, running total fields and other formula fields.
- How to return a value from a formula by setting the special variable named formula.

- How to use functions specific to report processing such as ReportTitle and OnFirstRecord.
- How to use the type system. Basic syntax is strongly typed, similar to using Option Explicit in Visual Basic and there is no Variant type. Basic syntax supports separate Date, Time and DateTime types unlike just the Date type in Visual Basic. It also supports range types such as 10 To 20 to allow for ranges of values which are common in many reporting applications.

Read this section to familiarize yourself with the above details.

Familiar features

- Many Basic syntax functions work in the same way as their counterparts in Visual Basic. This includes string functions such as Len, Mid and Filter, math functions such as Abs, Rnd and Sin, financial functions such as PV, programming shortcut functions such as IIF and date functions such as DateSerial, DateAdd and DateDiff.
- Most operators supported by Visual Basic are also in Basic syntax. For example, string concatenation (&) and date-time literals (#...#).
- Most statements and control structures use the same syntax as in Visual Basic. This includes the If, Select, Do While, Do Until, While and For/Next statements.
- The overall look of the formula will be unmistakably Basic like. For example, Basic style comments and line continuation characters are supported as is the Basic language use of new lines, colons, and the equal sign.

If you already know Crystal syntax

The main adjustment is getting used to the parts of the Basic language that are common to every version of Basic. Read this section for a detailed introduction.

Basic syntax fundamentals

The result of a formula

The result of a formula, or the value that is printed when the formula is placed in a report, is called the *value returned by the formula*. Every formula in Crystal Reports must return a value. Basic syntax does this by setting the value of the special variable *formula*. For example, here is a simple Basic syntax formula that returns the value 10:

```
formula = 10
```

The value returned by a formula can be one of the seven simple data types supported: Number, Currency, String, Boolean, Date, Time and DateTime. Crystal Reports also supports range types and array types, but these cannot be returned by a formula.

Related topics

- Restrictions on changing formula variable data types
- How formulas relate to functions in Microsoft Visual Basic
- The variable formula must be assigned a value
- Basic syntax is not case-sensitive
- Practice using the Xtreme sample database.

Restrictions on changing formula variable data types

The *formula* variable can be set several times within a single formula. For example, suppose a company has a shipping policy in which orders over $1,000 are insured, but orders below that amount are not insured:

```
Rem A formula that returns a String value
If {Orders.Order Amount} >= 1000 Then
formula = "Insured shipping"
Else
formula = "Regular shipping"
End If
```

Tip: The text following the keyword Rem is a comment for someone reading this formula and is ignored by the Basic syntax compiler.

The above formula returns the text string value "Insured shipping" if the value of the database field {Orders.Order Amount} is greater than or equal to 1000; otherwise, it returns the text string value "Regular Shipping." Text strings are usually just referred to as strings. Notice that the *formula* variable appears twice in the above example.

If the *formula* variable is set to a value of one type, it cannot be set to a value of another type later in the same formula. For example, replacing the String "Regular shipping" in the above example with the Number 10 would result in an error since the special variable *formula* was first set to the String value "Insured shipping."

The reason for this restriction is that Crystal Reports needs to know in advance what the return type of a formula will be so that it can allocate enough storage for the returned values. This is because different types have different storage requirements. Another reason is that the formatting options available for a formula field depend on its type. For example, a Number field has Number formatting options, such as the number of decimals to display, which do not make sense for a String field.

Note: The special variable *formula* should not be declared, unlike other variables used in a Basic syntax formula. See Variables (Basic syntax).

Restrictions on changing formula variable data types

The *formula* variable can be set several times within a single formula. For example, suppose a company has a shipping policy in which orders over $1,000 are insured, but orders below that amount are not insured:

```
Rem A formula that returns a String value
If {Orders.Order Amount} >= 1000 Then
formula = "Insured shipping"
Else
formula = "Regular shipping"
End If
```

Tip: The text following the keyword Rem is a comment for someone reading this formula and is ignored by the Basic syntax compiler.

The above formula returns the text string value "Insured shipping" if the value of the database field {Orders.Order Amount} is greater than or equal to 1000; otherwise, it returns the text string value "Regular Shipping." Text strings are usually just referred to as strings. Notice that the *formula* variable appears twice in the above example.

If the *formula* variable is set to a value of one type, it cannot be set to a value of another type later in the same formula. For example, replacing the String "Regular shipping" in the above example with the Number 10 would result in an error since the special variable *formula* was first set to the String value "Insured shipping."

The reason for this restriction is that Crystal Reports needs to know in advance what the return type of a formula will be so that it can allocate enough storage for the returned values. This is because different types have different storage requirements. Another reason is that the formatting options available for a formula field depend on its type. For example, a Number field has Number formatting options, such as the number of decimals to display, which do not make sense for a String field.

Note: The special variable *formula* should not be declared, unlike other variables used in a Basic syntax formula. See Variables (Basic syntax).

How formulas relate to functions in Microsoft Visual Basic

Consider the following Basic syntax formula:

```
Rem A formula that returns a String value
 Rem The function Rnd returns a random number
 Rem between 0 and 1
 If Rnd > 0.9 Then
 formula = "You won!"
 Else
 formula = "Sorry, try again."
 End If
```

The above formula returns the text string value "You won!" if the random number returned by Rnd is greater than 0.9 and the text string value "Sorry, try again." otherwise.

The use of the *formula* variable is similar to writing a function named *formula* in Visual Basic.

For example, the above formula could be written as a Visual Basic function as follows:

```
Rem The following code is in Visual Basic
 Function formula()
 If Rnd > 0.9 Then
 formula = "You won!"
 Else
 formula = "Sorry, try again."
 End If
 End Function
```

The variable formula must be assigned a value

If the variable *formula* is not assigned a value, it is not a complete Basic syntax formula.

Note: Some examples in this section are not complete Basic syntax formulas but rather just fragments intended to explain a particular feature.

What if I'm not interested in using the value returned by a formula?

Sometimes you may want to write a formula that just declares and initializes some global variables. These formulas are commonly inserted into the report header section of a report. In such cases, assign any value to the special variable *formula*. Every formula must return a value, even if you are not interested in using that value.

For example:

```
Rem Some Global variable declarations
 Rem Remember to set the value of 'formula'
 Global x As String, y As Number, z As DateTime
 x = "hello"
 y = 10.5
 z = #Aug 6, 1976#
 formula = 10
```

Basic syntax is not case-sensitive

What this means is that *formula*, *Formula*, and *FORMULA* are all considered to be the same. This is true of all variable names, functions, and keywords used in a Basic syntax formula.

Note: The only exception to this rule is for strings. The string "Hello" is not the same as the string "hello".

Practice using the Xtreme sample database

Many of the examples in this section refer to the Xtreme sample database. This database is included with Crystal Reports.

Browse the Samples\En directory to find the Databases folder that contains the Xtreme database.

Comments (Basic syntax)

Formula comments are notes included with a formula to explain its design and operation. Comments do not print and they do not affect the formula; they appear only in the Formula Editor. Use comments to explain the purpose of a formula or explain the steps involved in writing it.

Comments work as in Visual Basic. Begin comments with a Rem or an apostrophe.

Note: A comment beginning with a Rem is a separate statement and must either start on a new line or be separated from the previous statement by a colon.

```
Rem This is a comment
 Rem This is another comment
```

```
formula = 10 'So is any text after an apostrophe
formula = 20 : Rem This is also a comment
'Comments can occur after the formula text
```

Fields (Basic syntax)

Many of the fields you use when creating your report can also be referred to in your formulas. For example, database, parameter, running total, SQL expression, summary, and group name fields can all be used in a formula. You can also refer to other formula fields in your formula.

The easiest way to insert a field into your report is to double-click a field's name in the Report Fields tree. This ensures that the correct syntax for the field is used.

How fields appear in formulas

Database, parameter, formula, running total and SQL expression fields have their names surrounded by braces. Database field names are taken from the database. For example:

- database field: {Employee.Last Name}

Parameter, formula, running total, and SQL expression field names are specified when the fields are created.

- parameter fields also includes a question mark: {?my parameter field}
- formula fields include an at sign: {@another formula }
- running totals fields include a pound sign: {#my running total}
- SQL expression fields include a percent sign: {%my SQL expression}

Summary and group name fields look like function calls. However, they are really shorthand notation for a report field.

- sum summary field: Sum({Orders.Order Amount}, {Orders.Ship Via})
- group name field: GroupName({Orders.Ship Via})

Sample formulas using fields

The formula in this example uses the Xtreme database. To find out how many days it takes to ship the product from the date when the order was placed, subtract the ship date database field from the order date database field:

```
Rem A formula that uses database fields
formula = {Orders.Ship Date} - {Orders.Order Date}
```

To find the total dollar amount of a given product that was ordered, multiply its unit price by the quantity ordered:

```
formula = {Orders Detail.Unit Price} * _
{Orders Detail.Quantity}
```

Note: The example uses the line continuation character " _ " (space underscore). See Statements (Basic syntax).

To calculate a sale price of 80 percent of the original unit price:

```
formula = {Orders Detail.Unit Price} * 0.80
```

Statements (Basic syntax)

A Basic syntax formula consists of a sequence of statements. Each statement must be separated from the previous statement by either a new line or a colon. Typically, each statement takes one line, but you can continue a statement onto the next line by using the line continuation character, which is a space followed by an underscore.

For example:

```
'Declare a variable x to hold a number
Dim x As Number
'Assign the value of 30 to x
x = 10 + 10 + 10
'This also assigns the value of 30 to x
x = 10 + _
10 + 10
'Line continuation characters _
can also be used in comments
Dim y as String
'Three statements separated by two colons
y = "Hello" : x = 30 : formula = True
```

Assignment (Basic syntax)

Use the equal sign (=) when making assignments. The keyword *Let* can be optionally included as well. For example:

```
x = 10
Let y = 20
```

Simple data types (Basic syntax)

The simple data types in Crystal Reports are Number, Currency, String, Boolean, Date, Time and DateTime.

Number (Basic syntax)

Enter numbers without any comma separators or currency symbols. (Generally, you would want to have formatted numbers appearing as the result of a formula and not in the formula itself.)

Examples

```
100000
-20
1.23
```

Currency (Basic syntax)

Use the CCur function to create a Currency value. The initial C in CCur stands for *convert* and it can be used to convert Number values to Currency values.

Examples

```
CCur (10000)
CCur (0)
CCur (1.23)
```

String (Basic syntax)

Strings are used to hold text. The text must be placed between double quotation marks (") and cannot be split between lines. If you want to include double quotes in a string, use two consecutive double quotation marks.

```
"This is a string."
"123"
"The word ""hello"" is quoted."
```

You can extract individual elements or substrings from a string by specifying the character position or a range of character positions. Negative values are allowed; they specify the position starting from the end of the string.

```
"hello" (2) 'Equal to "e"
"hello" () 'Equal to "h"
"60455234" (1 to 3) 'Equal to "604"
"abcdef" ( to ) 'Equal to "def"
```

You can also extract substrings from a string using the Left, Right and Mid functions.

Boolean (Basic syntax)

The Boolean values are:

```
 True
 False
```

Note: *Yes* can be used instead of True and *No* instead of False.

Date, Time, and DateTime (Basic syntax)

The DateTime type can hold date-times, dates only or times only. The Date type holds dates only and the Time type holds times only. The Date and Time types are more efficient than the DateTime type, and so can be used in situations where the added functionality and flexibility of the DateTime type is not needed.

Visual Basic does not support separate types for holding dates only or times only. The Basic syntax DateTime type is similar to Visual Basic's Date type.

You can create DateTime values directly using the date-time literal construction. It is formed by typing in the date-time between 2 pound (#) signs. Many different formats are supported, as in Visual Basic.

Note: Date-time literals cannot be split between lines.

Examples

```
#8/6/1976 1:20 am#
#August 6, 1976#
#6 Aug 1976 13:20:19#
#6 Aug 1976 1:30:15 pm#
#8/6/1976#
#10:20 am#
```

Even though `#10:20 am#` looks like it could have the Time type and `#8/6/1976#` looks like it could have the Date type, they do not. They both have the DateTime type, as do all date-time literals. For example, you can think of `#10:20 am#` as a DateTime value with a null date part. To convert it to the Time type use `CTime (#10:20 am#).`

Instead of using date-time literals you can use CDateTime to convert a String to a DateTime. For example:

```
CDateTime ("8/6/1976 1:20 am")
CDateTime ("10:20 am")
```

However, there is one key difference between using date-time literals and the above usage of CDateTime. Date-time literals always use U.S. English date formats rather than settings from the locale of the particular computer on which Crystal Reports is running. Thus, the date-time literal examples above would work on all computers. On the other hand, on a French system, you could use constructions like:

CDateTime ("22 aout 1997") 'Same as #Aug 22, 1997#

Date values can be constructed with CDate and Time values with CTime:

```
CDate ("Aug 6, 1969")
CDate (1969, 8, 6) 'Specify the year, month, day
'Converts the DateTime argument to a Date
CDate (#Aug 6, 1969#)
CTime ("10:30 am")
CTime (10, 30, 0) 'Specify the hour, minute, second
CTime (#10:30 am#)
```

Range data types (Basic syntax)

Ranges are designed to handle a spectrum of values. Range types are available for all the simple types except for Boolean. That is: Number Range, Currency Range, String Range, Date Range, Time Range and DateTime Range. You can generate ranges using the To, _To, To_, _To_, Is >, Is >=, Is < and Is <= keywords. In general, To is used for ranges with 2 endpoints, and Is used for open ended ranges (only one endpoint). The underscores are used to indicate whether or not the endpoints are in the range.

Examples of Number Range values

The range of numbers from 2 to 5 including both 2 and 5

```
2 To 5
```

The range of numbers from 2 to 5, not including 2 but including 5

```
2 _To 5
```

All numbers less than or equal to 5

```
Is <= 5
```

All number less than 5

```
Is < 5
```

Examples of DateTime Range values

```
#Jan 5, 1999# To #Dec 12, 2000#
Is >= #Jan 1, 2000#
```

Using ranges in formulas (Basic syntax)

There are twenty-seven functions in Crystal Reports that specify date ranges. For example, the function LastFullMonth specifies a range of date values that includes all dates from the first to last day of the previous month. So if today's date is September 15, 1999 then LastFullMonth is the same as the range value CDate (#Aug 1, 1999#) To CDate (#Aug 31, 1999#).

Ranges are often used with If or Select statements. The following example computes student letter grades based on their test scores. Scores greater than or equal to 90 receive an "A", scores from 80 to 90, not including 90 receive a "B" and so on.

```
Rem Compute student letter grades
Select Case {Student.Test Scores}
Case Is >= 90
formula = "A"
Case 80 To_ 90
formula = "B"
Case 70 To_ 80
formula = "C"
Case 60 To_ 70
formula = "D"
Case Else
formula = "F"
End Select
```

The above example uses the Select statement which is discussed in more detail in Control structures (Basic syntax). You can check if a value is in a range by using the In operator. For example:

```
formula = 5 In 2 To 10 'True
formula = 5 In 2 To_ 5 'False
formula = 5 In 2 To 5 'True
```

The Maximum and Minimum functions can be used to find the endpoints of a range:

```
formula = Maximum (2 To 10) 'Returns 10
```

Array data types (Basic syntax)

Arrays in Crystal Reports are ordered lists of values that are all of the same type. These values are known as the array's *elements*. The elements of an array can be any simple type or range type. One way to create an array is using the Array function.

Examples

An array of 3 Number values. The first element is 10, the second is 5 and the third is 20.

```
Array (10, 5, 20)
```

An array of 7 String values:

```
Array ("Sun", "Mon", "Tue", "Wed", "Th", "Fri", "Sat")
```

An array of 2 DateTime Range values (note the line continuation character which is used for readability):

```
Array (#Jan 1, 1998# To #Jan 31, 1998#, _
#Feb 1, 1999# To #Feb 28, 1999#)
```

You can extract individual elements out of an array using parentheses containing the index of the element you want. This is called *subscripting* the array:

```
Array (10, 5, 20) (2) 'Equal to 5
```

Note: Arrays in Basic syntax are indexed from 1 (this means the first element has index 1). This is unlike in Visual Basic where arrays are indexed from 0 by default. However, in Visual Basic, arrays can be indexed from 1 by using the *Option Base* statement.

Number ranges can also be used to subscript arrays. The result is another array. For example:

```
Array (10, 5, 20) (2 To 3) 'Equal to Array (5, 20)
```

Arrays are most useful when used with variables. Using variables, you can change the individual elements of an array and resize the array to accommodate more elements. This capability significantly expands the capabilities of the formula language to do complex calculations.

For example, you can accumulate database field values into a global array variable in a detail level formula, and then use a formula in a group footer to perform a calculation based on those values. This allows you to perform a wide variety of customized summary operations.

Variables (Basic syntax)

This section describes the key components of variables and shows you how to create variables and assign values to them.

Variable overview

A variable represents a specific data item, or value, and acts as a placeholder for that value. When a formula encounters a variable, the formula searches for the value of the variable and uses it in the formula. Unlike a constant value, which is fixed and unchanging, a variable can be repeatedly assigned different values. You assign a value to a variable and the variable maintains the value until you later assign a new value. Because of this flexibility, it is necessary for you to declare variables before you use them so that Crystal Reports is aware of them and understands how you intend to use them.

Example of a variable

If you wanted to report on customers by area code, you could create a variable that extracts the area code from a customer fax number. The following is an example of a variable called *areaCode*:

```
Dim areaCode As String
areaCode = Left ({Customer.Fax}, 3)
Rem could also use: areaCode = {Customer.Fax} (1 To 3)
```

The first line of the variable example is the variable declaration; it gives the variable a name and type. The database field {Customer.Fax} is a String field and the Left function extracts the first 3 characters from its current value. The variable areaCode is then assigned this value.

Variable declarations using Dim (Basic syntax)

Before using a variable in a formula, you must declare it. A variable can hold values of a given type. The allowed types are the seven simple types (Number, Currency, String, Boolean, Date, Time and DateTime), the six range types (Number Range, Currency Range, String Range, Date Range, Time Range and DateTime Range) and variables that hold arrays of the previously mentioned types. This gives a total of 26 different types that a variable can have. When you declare a variable, you also specify its name. A variable cannot have the same name as any function, operator or other keyword that is valid for Basic syntax. For example, your variable cannot be named Sin, Mod or If because Sin is a built in function, Mod is a built in operator and If is a built in keyword. When typing formulas in the formula editor, the names of the built-in functions, operators, and other keywords are highlighted in a different color. This makes it easy to check if the variable name conflicts.

Once a variable is declared, it can be used in the formula. For example, you might want to assign it an initial value:

```
Dim x As Number 'Declare x to be a Number variable
x = 10 'Assign the value of 10 to x
```

Variables can only hold values of one type (Basic syntax)

A variable can only hold values of one type. For example, if a variable holds a Number value you cannot later use it to hold a String.

Example

```
Dim y As String
y = "hello"
'OK - the Len function expects a String argument
formula = Len (y)
'Error- y can only hold String values
y = #Jan 5, 1993#
'Error- y can only hold String values
y = Array ("a", "bb", "ccc")
'Error- the Sin function expects a Number argument
formula = Sin (y)
```

You can declare more than one variable per statement by separating their declaration by commas:

```
Dim x As Number, y as String, z as DateTime Range
x = 10 : y = "hello"
z = #Jan 1, 1999# To #Jan 31, 1999#
```

Declaring variables without immediately specifying their type (Basic syntax)

In general, the type of a variable does not need to be explicitly given when declaring it. In such cases, the variable's type is determined by the first assignment that is made to it. This is similar to the special variable *formula*. This is different from in Visual Basic where a variable whose type is not given at declaration automatically has the Variant type. However, in practice, it means that you can write formulas in a similar style to what you would do if using a Variant in Visual Basic.

```
Dim p 'The type of p is not known yet
p = "bye" 'The type of p is now set to be String
Dim q 'The type of q is not known yet
q = Array ("hello", p) 'q is a String Array
'Error- p is a String variable and cannot hold a Number
p = 25
Dim r
'r is a Number variable, and holds the value 5
r = (10 + 5) / 3
'The types of a and c are not known yet
Dim a, b As Boolean, c
b = False
'The type of a is now set to Boolean
'and its value is False
a = b
'The type of c is now set to Number and its value is 17
c = 2 + 3 * 5
```

Examples of declaring and initializing range variables

```
Dim gradeA, quarter
'The type of gradeA is set to Number Range
gradeA = 90 To 100
'The type of quarter is set to Date Range
quarter = CDate (1999, 10, 1) To CDate (1999, 12, 31)
```

Variable Scope (Basic syntax)

Variable scopes are used to define the degree to which variables in one formula are made available to other formulas. There are three levels of scope in Crystal Reports: local, global and shared. Every variable has a scope, and this scope is specified when the variable is declared.

Local Variables

Variables with local scope, also known as local variables, are declared using either the Dim or Local keywords. For example, all the declarations in the previous section using Dim were declaring local variables. Another example:

```
Local x As Number 'equivalent to Dim x As Number
```

Local variables are restricted to a single formula and a single evaluation of that formula. This means that you cannot access the value of a local variable in one formula from a different formula.

Example

```
Rem Formula A
Local x as Number
x = 10
formula = x
Rem Formula B
EvaluateAfter ({@Formula A})
Local x as Number
formula = x + 1
```

The function call EvaluateAfter ({@Formula A}) ensures that Formula B will be evaluated after Formula A is evaluated. Formula A returns a value of 10 and Formula B returns a value of 1. Formula B does not have access to Formula A's x and thus cannot use the value of 10 and add 1 to it; instead, it uses the default value for the uninitialized local variable x found in Formula B, which is 0, and adds 1 to it to get 1.

You can also create local variables with the same name but different types in different formulas. For example, the type declarations in formulas A and B do not conflict with:

```
Rem Formula C
Local x as String
x = "hello"
formula = x
```

Local variables are the most efficient of the three scopes. In addition, they do not interfere with one another in different formulas. For these reasons, it is best to declare variables to be local whenever possible.

Global variables (Basic syntax)

Global variables use the same memory block to store a value throughout the main report. This value is then available to all formulas that declare the variable, except for those in subreports. Declare a global variable as in the following example:

```
Global y As String
```

Since global variables share their values throughout the main report, you cannot declare a global variable in one formula with one type and then declare a global variable with the same name in a different formula with a different type.

Example

```
Rem Formula A
Global z As Date
z = CDate (1999, 9, 18)
formula = 10
Rem Formula B
Global z As Number
formula = True
```

In this case, if you enter and save Formula A first, Crystal Reports will return an error when you check or try to save Formula B. This is because the declaration of the Global variable z as a Number conflicts with its earlier declaration in Formula A as a Date.

Using Global variables

Global variables are often used to perform complex calculations where the results of a formula depend upon the grouping and page layout of the actual printed report. This is accomplished by creating several formulas, placing them in different sections of the report, and having the different formulas interact via global variables.

Here is an example of the sort of effects that can be produced:

```
Rem Formula C
Global x as Number
x = 10
formula = x
Rem Formula D
'call the function WhileReadingRecords
WhileReadingRecords
Global x as Number
x = x + 1
formula = x
```

If Formula C is placed in the Report Header and then Formula D is placed in a detail section, Formula C will be evaluated before Formula D. Formula C will be evaluated once and then Formula D will be evaluated for each record appearing in the detail section. Formula C returns 10. For the first detail record, Formula D returns 11. This is because the value 10 of x is retained from when it was set by Formula C. Formula D then adds 1 to this value, setting x to 11 and then returns 11. For the second detail record, formula D return 12, adding 1 to the previously retained value of x which was 11. This process continues for the remaining detail records.

The call to WhileReadingRecords tells Crystal Reports to re-evaluate Formula D as it reads in each record of the report. Otherwise, since the formula does not contain any database fields, the program will evaluate it only once before reading the records from the database. The formula will then return the value 11 instead of 11, 12, 13, ... as the successive records are processed.

If the statement `x = x + 1` is replaced by `x = x + {Orders Detail.Quantity}`, you create the effect of a running total based on {Orders Detail.Quantity}, although it is one starting at 10 rather than 0 because of Formula C. In this case, you can omit the call to WhileReadingRecords, since it will automatically occur because the formula contains a database field.

Shared variables (Basic syntax)

Shared variables use the same memory block to store the value of a variable throughout the main report and all of its subreports. Thus shared variables are even more general than global variables. To use a shared variable, declare it in a formula in the main report as in the following example:

```
Shared x As Number
x = 1000
```

and declare it in a formula in the subreport as in the following example:

```
Shared x as Number
```

In order to use shared variables the variable must be declared and assigned a value before it can be passed between the main report and the subreport.

Declaring array variables (Basic syntax)

There are several different ways of declaring array variables. The first way is to use empty parentheses and explicitly specify the type of the array:

```
'Declare x to be a Global variable
'of Number Array type
Global x () As Number
'Initialize x
x = Array (10, 20, 30)
'Declare y to be a Shared variable
'of String Range Array type
Shared y () As String Range
'Initialize y
y = Array ("A" To "C", "H" To "J")
```

The second way is to declare the variable without specifying that it is an array and without giving its type and waiting for the first assignment to the variable to completely specify its type:

```
'Declare y to be a Local variable
'but do not specify its type
Dim y
'The type of y is now set to be a String Array
y = Array ("Sun", "Mon", "Tue", "Wed", "Th", _
"Fri", "Sat")
```

The third way is to declare that the variable is an array but not specify its type fully until the first assignment. Assuming the declaration of y above:

```
'Declare z to be a Local variable that is an Array
Local z()
'z is set to Array ("Mon", "Tue") and is a String Array
z = y(2 to 3)
```

The fourth way is to explicitly specify the size of the array during the declaration. If you use this technique, the array is automatically created and default values are used to fill the array. For example, for a Number Array, each element is initialized to 0 and for a String array each element is initialized to the empty string "". Since this type of declaration actually creates the array, you must specify its type with the As clause so that Crystal Reports knows how much storage space to reserve for the array.

```
Dim a(2) As String
 'Assign a value to the first element of the array a
 a(1) = "good"
 a(2) = "bye"
 'The & operator can be used to concatenate strings
 'the formula returns the String "goodbye"
 formula = a(1) & a(2)
```

Arrays and For/Next loops

Arrays are commonly used with For/Next loops. The following example creates and then uses the array `Array (10, 20, 30, ..., 100)` using a For/Next loop. See For/Next loops (Basic syntax) for more details.

```
Dim b (10) As Number
 Dim i
 For i = 1 To 10
 b(i) = 10 * i
 Next i
 formula = b(2) 'The formula returns the Number 20
```

Using array variables (Basic syntax)

You can assign values to elements of an array and also use the values of the elements for other computations:

```
Global x() As String
x = Array ("hello", "bye", "again")
'Now x is Array ("hello", "once", "again")
x (2) = "once"
'The statement below would cause an error if not
'commented out since the array has size 3
'x (4) = "zap"
'The formula returns the String "HELLO"
formula = UCase (x (1))
```

The Redim and Redim Preserve keywords can be used to resize an array, which is useful if you want to add extra information to it. Redim erases the previous contents of the array first before resizing it whereas Redim Preserve preserves the previous contents.

```
Dim x () As Number
Redim x (2) 'Now x is Array (0, 0)
x (2) = 20 'Now x is Array (0, 20)


Redim x (3) 'Now x is Array (0, 0, 0)
```

```
x (3) = 30 'Now x is Array (0, 0, 30)
Redim Preserve x (4) 'Now x is Array (0, 0, 30, 0)
formula = "finished"
```

Default values for the simple types (Basic syntax)

An uninitialized variable will have the default value for its type. In general, it is not a good programming practice to rely on the default values of types. For example, initialize all local variables in your formula, initialize all global variables in a formula placed in the Report Header, and initialize all shared variables in a formula placed in the Report Header of the main report.

When an array is resized using the Redim keyword, the entries are filled with default values for the type.

Default values

Number

```
0
```

Currency

```
CCur (0)
```

String

```
"" 'The empty string
```

Date

```
CDate (0, 0, 0) 'The null Date value
```

Time

The null Time value. Value held by an uninitialized Time variable.

DateTime

The null DateTime value. Value held by an uninitialized DateTime variable.

Note: It is not recommended that your formulas rely on the values of uninitialized range or array variables.

Automatic type conversions (Basic syntax)

Generally in Crystal Reports, values of one type cannot be used where values of another type are expected without explicitly supplying a type conversion function. For example:

```
Dim postalCode as String
'Error- assigning a Number value to a String variable
postalCode = 10025
'OK- use the type conversion function CStr
'to create "10025"
postalCode = CStr (10025, 0)
```

However, there are a few conversions that are made automatically:

- Number to Currency
- Date to DateTime
- Simple type to Range value of the same underlying simple type

For example, the following assignments are correct:

```
Dim cost As Currency
'Same as: cost = CCur (10)
cost = 10
Dim orderDate As DateTime
'Same as: orderDate = CDateTime (1999, 9, 23, 0, 0, 0)
orderDate = CDate (1999, 9, 23)
Dim aRange As Number Range
'Same as: aRange = 20 To 20
aRange = 20
Dim aRangeArray () As Number Range
'Same as :
'aRangeArray = Array (10 To 10, 20 To 25, 2 To 2)
aRangeArray = Array (10, 20 To 25, 2)
```

Note: The opposite conversions are not allowed. For example:

```
Dim num As Number
num = 5 + CCur (10) 'Error
'OK- convert to Number type using the CDbl function
num = CDbl (5 + CCur (10))
```

5 is converted to CCur (5) and added to CCur (10) to make CCur (15). However, this Currency value cannot be automatically assigned to the Number variable num since automatic conversions from Currency to Number are not allowed. Similarly, functions accepting a Currency argument can be supplied a Number argument instead, and the Number argument will be converted to a Currency, whereas functions accepting a Number argument cannot be supplied a Currency argument without first explicitly converting the Currency to a Number using CDbl.

Functions (Basic syntax)

When using a function in a formula, type the name of the function and supply the arguments required. For example, the Len function requires a String argument and computes the length of the string.

```
Dim x As String
x = "hello"
formula = Len (x) 'The formula returns the Number 5
```

Supplying arguments of the incorrect type required by the function produces an error. For example, calling Len (3) would produce an error since Len does not accept a Number argument. Functions sometimes can accept different numbers of arguments or types of arguments. For example, the CDate function could accept a single String argument to form a Date value or 3 Number values holding the year, month and day respectively and form a Date value from them. See Date, Time, and DateTime (Basic syntax).

Example with the Mid function

```
Dim x as String
x = "hello"
'Start at position 2, go to the end of the string
formula = Mid (x, 2) 'formula is now "ello"
'Start at position 2, extract 1 character
formula = Mid (x, 2, 1) 'formula is now "e"
```

The classes of functions are: Math, Summary, Financial, String, Date/Time, Date Range, Array, Type Conversion, Programming Shortcuts, Evaluation Time, Print State, Document Properties and Additional Functions. There are also some functions specific to conditional formatting formulas.

Functions similar to Visual Basic functions of the same name (Basic syntax)

The Math, Financial, String, Date/Time, Type Conversion and Programming Shortcuts groups consist mainly of functions that are familiar to Visual Basic users. Most of the functions are intended to work in the same way as the Visual Basic function of the same name.

Note:

- Sometimes functions will have more overloads than are available in Visual Basic.

  For example, the CDate function supports the Visual Basic overload of creating a Date value from a String value, such as CDate ("Sept 18, 1999") but it also supports an overload of creating a Date value by supplying the year, month and day as Number arguments e.g. CDate (1999, 9, 18). The overloads are indicated in the Functions tree.

- Some functions that are supported by Basic syntax are not listed in the Basic syntax Functions tree. This is because they are equivalent to Basic syntax functions that are already listed in the tree.

  For example, the Len function is the normal Basic syntax and Visual Basic function for finding the length of a string. However, the Length function is listed in the Functions tree and works the same as the Len function. Length is the traditional Crystal syntax function for this action and it is included for the convenience of Crystal syntax users migrating formulas to Basic syntax.

Summary functions (Basic syntax)

The Summary function group provides functions for creating summary fields such as:

```
Sum({Orders.Order Amount}, {Orders.Ship Via})
```

Summary fields are normally created using the Insert Summary or Insert Grand Total dialogs. They then appear in the Report Fields tree, and can be used in a formula by double clicking there. However, they do not need to be created in this way. You can create a summary field exclusively for use by your formula by filling in the arguments to one of the functions in the Summary functions section. However, any groups that refer to summary fields must already exist in the report.

Date ranges (Basic syntax)

For additional information refer to Range data types (Basic syntax).

Date ranges produced by these functions depend on the current date. For example, if today's date is September 18, 1999, then LastFullMonth is the Date Range value:

```
CDate(#Aug 1, 1999#) To CDate(#Aug 31, 1999#)
```

This functionality is often useful, but if you want to determine a date range based on a database field such as {Orders.Order Date}? The Date/Time functions can be used instead.

Example

```
Dim d As Date
d = CDate ({Orders.Order Date})
Dim dr As Date Range
dr = DateSerial (Year(d), Month(d) - 1, 1) To _
DateSerial (Year(d), Month(d), 1 - 1)
'At this point dr is the Date Range value holding
'the last full month before {Orders.Order Date}
```

The DateSerial function makes this easy because you don't have to worry about special cases. It never lets you create an invalid date. For example, DateSerial (1999, 1 - 1, 1) is December 1, 1998. Note that in the above example, {Orders.Order Date} is actually a DateTime field and so the CDate function is used to convert it to a date by truncating the time part.

Array functions (Basic syntax)

The array functions compute summaries of an array's elements. For example, the Sum function when applied to an array returns the sum of the elements of the array.

Example

The following formula returns 100:

```
formula = Sum (Array (10, 20, 30, 40))
```

Evaluation Time functions (Basic syntax)

These are the report specific functions: BeforeReadingRecords, WhileReadingRecords, WhilePrintingRecords and EvaluateAfter. You can use these functions to guide Crystal Reports as to when your formula should be evaluated.

Should the formula be evaluated before retrieving the records from the database, while reading the records from the database but before the records have been grouped, sorted and summarized, or while printing the report, when the records are grouped, sorted and summarized? In general, Crystal Reports sets an appropriate evaluation time for your formula, based on how much information the formula needs. For example, if a formula uses a database field, then it cannot be evaluated before the records are read from the database. However, you sometimes need to force a later evaluation time than normal to get the desired effect. See Using Global variables for an example.

Normally, the returned value of a function is used further in a formula. However, evaluation time functions are called to change the internal behavior of Crystal Reports and their return value is not used. They can be called by just placing their name in a separate statement, optionally preceded by the keyword Call.

```
WhilePrintingRecords
Call WhilePrintingRecords
```

Print State functions (Basic syntax)

These are once again reporting-specific functions. For example, the notation {Orders.Order Date} refers to the value of the field in the current record where PreviousValue ({Orders.Order Date}) refers to the value in the immediately preceding record. NextValue ({Orders.Order Date}) refers to the value in the next record. IsNull ({Orders.Order Date}) checks if the field's value is null.

Other examples are PageNumber and TotalPageCount. These can be used to access pagination information about your report.

Document Properties functions (Basic syntax)

These are reporting specific functions that refer to the report document as a whole. For example, PrintDate and ReportTitle.

Additional functions (Basic syntax)

These are functions that are in User Function Libraries (UFLs). A UFL is a separate dynamic link library or Automation server that you create and Crystal Reports uses to add your own customized functions to the formula language. Writing a UFL is more involved than writing a formula using Basic or Crystal syntax. See Crystal Reports Developer's Help for more information.

Note: Using UFLs makes your reports less portable because you must distribute your UFL along with the report.

Conditional formatting functions (Basic syntax)

When writing a conditional formatting formula, you may want to use the additional functions that appear at the top of the Functions tree.

Example

If you wanted to format the {Customer.Last Year's Sales} field so that sales of more than $100,000 are printed in green and sales of less than $15,000 are printed in red and all else are printed in black.

```
Rem Conditional formatting example 1
If {Customer.Last Year's Sales} > 100000 Then
formula = crGreen
ElseIf {Customer.Last Year's Sales} < 15000 Then
formula = crRed
Else
formula = crBlack
End If
```

Since this is a font color formatting function, the list of Color Constants appears in the Functions tree. This example uses three: crGreen, crRed and crBlack. You could have used the actual numeric values of the color constants instead. For example, crRed is 255 and crGreen is 32768. However, the formula is easier to understand using the color constants. All constant functions in Basic syntax have the "cr" prefix.

Note: Some formatting attributes do not use constant functions. For example, if you wanted to not print {Customer.Last Year's Sales} values if the sales were less than $50,000, you could write the following conditional formatting formula for the suppress attribute:

```
Rem Conditional formatting example 2
If {Customer.Last Year's Sales} < 50000 Then
formula = True 'suppress the value
Else
formula = False 'do not suppress the value
End If
```

Or more simply:

```
Rem Conditional formatting example 3 -
Rem equivalent to example 2
formula = {Customer.Last Year's Sales} < 50000
```

If the last year's sales are less than $50,000, then the expression

```
{Customer.Last Year's Sales} < 50000
```

is True, and so the formula returns True. On the other hand, if the last year's sales are greater than or equal to $50,000, then

```
{Customer.Last Year's Sales} < 50000
```

is False and so the formula returns False.

General purpose conditional formatting functions (Basic syntax)

There are three general purpose conditional formatting functions:

- CurrentFieldValue
- DefaultAttribute
- GridRowColumnValue.

These functions are displayed at the top of the Functions tree when appropriate. DefaultAttribute can be used for any formatting formula. CurrentFieldValue can be used for any formatting formula where you are formatting a field value. GridRowColumnValue can be used for formatting a formula where you are formatting a field value in a Cross-tab or OLAP grid.

CurrentFieldValue enables you to conditionally format Cross-tab or OLAP grid cells based on their value. GridRowColumnValue enables you to conditionally format the cells of a Cross-tab or OLAP grid based on row or column headings values. These two functions are essential in some situations as there is no other way in the formula language to refer to these fields.

Example

If you wanted Cross-tab cells to be suppressed if the values are less than 50,000:

```
Rem Conditional formatting example 4
formula = CurrentFieldValue < 50000
```

Operators (Basic syntax)

Arithmetic operators

The arithmetic operators are addition (+), subtraction (-), multiplication (*), division (/), integer division (\), modulus (Mod), negation (-) and exponentiation (^). Arithmetic operators are used to combine numbers, numeric variables, numeric fields and numeric functions to get another number.

Examples

```
'Outstanding preferred stock as a percent of
'common stock
formula = ({Financials.Preferred Stock} / _
{Financials.Common Stock}) * 100
'The square root of 9, Sqr(9), is 3.
'The formula returns 17.
formula = 7 + 2 * 3 - 2 + Sqr(6 + 3) * Len("up")
```

Order of precedence

When you create arithmetic expressions that involve several operators, the order that the program evaluates the various elements of the expression becomes important.

In general, the program evaluates expressions in the following order:

- from left to right
- follows the rules of precedence from basic math.

Example

Multiplication and division are performed first from left to right. Then addition and subtraction are performed. For example, 5 + 10 * 3 = 5 + 30 = 35.

You can change this order of precedence by using parentheses. For example,
(5 + 10) * 3 = 15 * 3 = 45. If you are unsure of the order of precedence, it is a good idea to clarify your intentions with parentheses.

The arithmetic operators in Crystal Reports have the same order of precedence as in Visual Basic. Here is the list, from highest precedence to lowest:

- Exponentiation (^)
- Negation (-)
- Multiplication and division (*, /)
- Integer Division (\)
- Modulus (Mod)
- Addition and subtraction (+, -).

Comparison operators (Basic syntax)

The comparison operators are equal (=), not equal (<>), less than (<), less than or equal to (<=), greater than (>) and greater than or equal to (>=).

Comparison operators are usually used to compare operands for a condition in a control structure such as an If statement. Comparison operators as a group all have lower precedence than the arithmetic operators. For example, expressions like 2 + 3 < 2 * 9 are the same as (2 + 3) < (2*9).

Boolean operators (Basic syntax)

The Boolean operators are, in order of precedence from greatest to lowest: Not, And, Or, Xor, Eqv and Imp. Boolean operators are typically used with comparison operators to generate conditions for control structures. Boolean operators as a group have lower precedence than the comparison operators. Thus for example, the expression 2 < 3 And 4 >= is the same as (2 < 3) And (4 >=).

Null fields and how to use IsNull (Basic syntax)

In general, when Crystal Reports encounters a null valued field in a formula, it immediately stops evaluating the formula and produces no value. If you want to handle null field values in your formula, you must explicitly do so using one of the special functions designed for handling them: IsNull, PreviousIsNull or NextIsNull.

Example

The {Product.Color} field contains both basic colors such as "red" and "black" and more descriptive two word colors such as "steel satin" and "jewel green". Here's an example of a formula that writes out "basic" for the basic colors and "fancy" for the others.

```
If InStr({Product.Color}, " ") = 0 Then
formula = "basic"
Else
formula = "fancy"
End If
```

The function call to InStr searches the {Product.Color} string for a space. If it finds a space, it returns the position of the space, otherwise it returns 0. Since basic colors are only one word with no spaces, InStr will return 0 for them.

For some products, such as the Guardian Chain Lock, a color value was not recorded and so the {Product.Color} field has a null value in the database for that record. Thus, the Guardian Chain Lock record does not have any word printed beside it.

Here is an example of how to fix the above example using IsNull:

```
If IsNull({Product.Color}) Or

   InStr({Product.Color}, " ") = 0 Then

   "basic"

Else

   "fancy"
```

Relating to operators, when Crystal Reports evaluates the condition:

```
IsNull({Product.Color}) Or
```

```
InStr({Product.Color}, " ") = 0
```

It first evaluates IsNull ({Product.Color)), and when it determines that this is True, it knows that the whole condition is True, and does not need to check whether

```
InStr({Product.Color}, " ") = 0
```

In other words, Crystal Reports will stop evaluating a Boolean expression when it can predict the results of the whole expression. In the following example, the formula guards against attempting to divide by zero in the case that denom is 0:

```
Local NumberVar num;

Local NumberVar denom;

...

If denom <> 0 And num / denom > 5 Then

...
```

Dealing with null values (Basic syntax)

For some products, such as the Guardian Chain Lock, a color value was not recorded and so the {Product.Color} field has a null value in the database for that record. In general, when Crystal Reports encounters a null valued field in a formula, it immediately stops evaluating the formula and produces no value. Thus, the Guardian Chain Lock record does not have any word printed beside it. If you want to handle null field values in your formula, you must explicitly do so using one of the special functions designed for handling them: IsNull, PreviousIsNull or NextIsNull. Here is an example of how to fix the above example using IsNull:

```
If IsNull({Product.Color}) Or _
InStr({Product.Color}, " ") = 0 Then
formula = "basic"
Else
formula = "fancy"
End If
```

Relating to operators, when Crystal Reports evaluates the condition:

```
IsNull({Product.Color}) Or _
InStr({Product.Color}, " ") = 0
```

It first evaluates IsNull ({Product.Color)), and when it determines that this is True, it knows that the whole condition is True, and does not need to check whether

```
InStr({Product.Color}, " ") = 0
```

In other words, Crystal Reports will stop evaluating a Boolean expression when it can predict the results of the whole expression. In the following example, the formula guards against attempting to divide by zero in the case that denom is 0:

```
Dim num As Number, denom As Number
...
```

```
If denom <> 0 And num / denom > 5 Then
...
```

Note: Visual Basic does not support this technique, since all parts of a Boolean expression in Visual Basic are evaluated, even if not necessary.

Control structures (Basic syntax)

Formulas without control structures execute each statement in the formula only once. When this happens the formula is evaluated. The statements are executed in a sequential fashion, from the first statement in the formula to the last. Control structures enable you to vary this rigid sequence. Depending upon which control structure you choose, you can skip over some of the statements or repeatedly evaluate some statements depending on certain conditions. Control structures are the primary means of expressing business logic and typical report formulas make extensive use of them.

Basic syntax supports many of the main control structures from Visual Basic with the same syntax. One of the advantages of the Basic language is it is easy to read block notation for control structures. This simplifies the writing and debugging of complex formulas.

If statements (Basic syntax)

The If statement is one of the most useful control structures. It enables you to evaluate a sequence of statements if a condition is true and evaluate a different sequence of statements if it is not true.

Note: When formatting with conditional formulas, always include the Else keyword; otherwise, values that don't meet the If condition may not retain their original format. To prevent this, use the DefaultAttribute function (`If...Else formula = DefaultAttribute`).

Example

A company plans to pay a bonus of 4 percent to its employees except for those who work in Sales who will receive 6 percent. The following formula using an If statement would accomplish this:

```
Rem Multiine If example 1
If {Employee.Dept} = "Sales" Then
formula = {Employee.Salary} * 0.06
Else
formula = {Employee.Salary} * 0.04
End If
```

In this example, if the condition {Employee.Dept} = "Sales" evaluates as true, then the

```
formula = {Employee.Salary} * 0.06
```

statement is processed. Otherwise the statement following the Else, namely the

```
formula = {Employee.Salary} * 0.04
```

is processed.

Suppose another company wants to give employees a 4% bonus, but with a minimum bonus of $1,000. The following example shows how. Notice that the Else clause is not included; it is optional, and not needed in this case.

```
Rem Multiine If example 2
formula = {Employee.Salary} * 0.04
If formula < 1000 Then
formula = 1000
End If
```

Now suppose that the previous company also wants a maximum bonus of $5,000. You now need to use an ElseIf clause. Notice that ElseIf is all one word. The following example has only one ElseIf clause, but you can add as many as you need.

Note: There is a maximum of one Else clause per If statement.

The Else clause is executed if none of the If or ElseIf conditions are true.

```
Rem Multiine If example 3
formula = {Employee.Salary} * 0.04
If formula < 1000 Then
formula = 1000
ElseIf formula > 5000 Then
formula = 5000
End If
```

Example

Suppose that a company wants to compute an estimate of the amount of tax an employee needs to pay and write a suitable message. Income below $8,000 is not taxed, income between $8,000 and $20,000 is taxed at 20%, income between $20,000 and $35,000 is taxed at 29%, and income above $35,000 is taxed at 40%.

```
Rem Multiine If example 4
Dim tax As Currency, income As Currency
income = {Employee.Salary}
Dim message As String
If income < 8000 Then
tax = 0
message = "no"
ElseIf income >= 8000 And income < 20000 Then
message = "lowest"
tax = (income - 8000)*0.20
ElseIf income >= 20000 And income < 35000 Then
message = "middle"
tax = (20000 - 8000)*0.20 + (income - 20000)*0.29
Else
message = "highest"
tax = (20000 - 8000)*0.20 + (35000 - 20000)*0.29 + _
(income - 35000)*0.40
End If
Dim taxStr As String
Rem use 2 decimal places
Rem and use the comma as a thousands separator
taxStr = CStr (tax, 2, ",")
```

```
formula = "You are in the " & message & _
" tax bracket. " & _
"Your estimated tax is " & taxStr & "."
```

Notice, the use of variables to simplify the logic of the computation. Also, notice that there are two statements that are executed when one of the conditions are met; one assigns the tax variable, and the other assigns the message variable. It is often useful to have multiple statements executed as a result of a condition.

Single-line and multi-line If statements (Basic syntax)

There are two kinds of If statement, the single-line if statement and the multi-line if statement. Starting on a new line after the first *Then* turns your If statement into a multi-line If statement. Otherwise it is a single-line If statement. The multi-line If statement always includes an End If whereas the single line If statement does not.

Note: Because of the use of line-continuation characters, single-line If statements do not need to be on a single line. In general, it is preferable to use multi-line If statements since they have a clearer layout. However, for simple situations, the single-line If statement is sometimes used.

```
Rem Single-line If example 1
Rem Same result as multi-line If example 1
If {Employee.Dept} = "Sales" Then _
formula = {Employee.Salary} * 0.06 _
Else _
formula = {Employee.Salary} * 0.04
```

Here is an example showing various forms of single-line If statements:

```
Rem Single-line If example 2
Dim per As Number, extra As Boolean
per = 2 : extra = False
'An example with no Else clause
If {Employee.Dept} = "Sales" Then per = 10
'More than 1 statement in the Then or Else part can
'be included by separating them with colons
If {Employee.Dept} = "R&D" Then _
per = 5 : extra = True _
Else _
per = 3
```

Select statements (Basic syntax)

The Select statement is similar to an If statement. Sometimes however, you can write formulas that are clear and less repetitive using the Select statement. This example evaluates the {Customer.Fax} field to determine if the area code is for Washington state (206, 360, 509) or British Columbia, Canada (604, 250):

```
Rem Select example 1
Select Case Left ({Customer.Fax}, 3)
Case "604", "250"
formula = "BC"
Case "206", "509", "360"
formula = "WA"
End Select
```

The expression right after the Select Case keywords is called the *Select condition*. In the above example it is Left ({Customer.Fax}[1 To 3]). The Select statement tries to find the first Case that matches the Select condition, and then executes the statements following it, up until the next Case.

```
Rem Same effect as Select example 1
Dim areaCode As String
areaCode = Left ({Customer.Fax}, 3)
If areaCode In Array ("604", "250") Then
formula = "BC"
ElseIf areaCode In Array ("206", "509", "360") Then
formula = "WA"
End If
```

Example

This formula groups the number of Oscar nominations a movie received into low, medium, high or extreme categories and in the process, shows some of the possibilities for the expression lists following the Case labels. Notice the optional Case Else clause. If none of the Case expression lists are matched by the preceding Case clauses, then the Case Else clause is matched. For example, in the following example, if {movie.NOM} is 11, then the formula returns "extreme".

```
Rem Select example 2
Select Case {movie.NOM}
Case 1,2,3, Is < 1
Rem Can have multiple statements in the
Rem statement blocks
formula = "low"
Case 4 To 6, 7, 8, 9
formula = "medium"
Case 10
formula = "high"
Case Else
formula = "extreme"
End Select
```

For/Next loops (Basic syntax)

For/Next loops enable you to evaluate a sequence of statements multiple times. This is unlike the If and Select statements where the program passes through each statement at most once during the formula's evaluation. For/Next loops are best when you know the number of times that the statements needs to be evaluated in advance.

The syntax of the For loop through examples

Sample 1

Suppose you want to reverse the {Customer.Customer Name} string. For example, "City Cyclists" becomes "stsilcyC ytiC".

```
Rem Reverse a string version 1
formula = ""
Dim strLen
strLen = Len ({Customer.Customer Name})
```

```
Dim i
For i = 1 To strLen
Dim charPos
charPos = strLen - i + 1
formula = formula & _
Mid({Customer.Customer Name}, charPos, 1)
Next i
```

Let us examine how this formula works assuming that the current value of the field {Customer.Customer Name} is "Clean Air". The variable strLen is assigned to be the length of "Clean Air", namely 9. At this time it is also typed to be a Number variable. The variable *i* is known as a *For counter variable* since its value changes with each iteration of the For loop. In other words, it is used to count the iterations of the loop. The For loop will iterate 9 times, during the first time, i is 1, then i is 2, then i is 3 and so on until finally i = 9. During the first iteration, the ninth character of {Customer.Customer Name} is appended to the empty special variable *formula*. As a result formula equals "r" after the first iteration. During the second iteration, the eighth character of {Customer.Customer Name} is appended to formula and so formula equals "ri". This continues until after the ninth iteration, formula equals, "riA naelC" which is the reversed string.

Sample 2

Here is a simpler version of the above formula, that uses a Step clause with a negative Step value of . For the "Clean Air" example, i is 9 for the first iteration, 8 for the second, 7 for the third and so on until it is 1 in the final iteration.

```
Rem Reverse a string version 2
formula = ""
Dim i
For i = Len ({Customer.Customer Name}) To 1 Step formula = formula +
_
Mid({Customer.Customer Name}, i, 1)
Next I
```

Sample 3

The simplest version is to use the built in function StrReverse:

```
Rem Reverse a string version 3
formula = StrReverse ({Customer.Customer Name})
```

The built in String functions in Crystal Reports 8.5 can handle many of the string processing applications which would traditionally be handled using a For/Next loop or some other kind of loop. However, For/Next loops provide the most flexibility and power in processing strings and arrays. This can be essential if the built-in functions do not cover your intended application.

For/Next loop example (Basic syntax)

Here is a more detailed example of Crystal Reports' string processing capabilities. The Caesar cipher is a simple code that is traditionally credited to Julius Caesar. In this code, each letter of a word is replaced by a letter five characters further in the alphabet. For example, "Jaws" becomes "Ofbx". Notice that "w" is replaced by "b". Since there are not 5 characters after "w" in the alphabet, it starts again from the beginning.

Here is a formula that implements applying the Caesar cipher to the field {Customer.Customer Name} in the Xtreme database:

```
Rem The Caesar cipher

Dim inString 'The input string to encrypt
inString = {Customer.Customer Name}
Dim shift
shift = 5

formula = ""

Dim i
For i = 1 To Len(inString)
Dim inC, outC
inC = Mid(inString, i, 1)
Dim isChar, isUCaseChar
isChar = LCase(inC) In "a" To "z"
isUCaseChar = isChar And (UCase (inC) = inC)

inC = LCase(inC)

If isChar Then
Dim offset
offset = (Asc(inC) + shift - Asc("a")) Mod _
(Asc("z") - Asc("a") + 1)
outC = Chr(offset + Asc("a"))
If isUCaseChar Then outC = UCase(outC)
Else
outC = inC
End If
formula = formula & outC
Next i
```

In the above example, there is a multi-line If statement nested within the statements block of the For/Next loop. This If statement is responsible for the precise details of shifting a single character. For example, letters are treated differently from punctuation and spaces. In particular, punctuation and spaces are not encoded. Control structures can be nested within other control structures and multiple statements can be included in the statement block of a control structure.

Using the Exit For statement (Basic syntax)

You can exit from a For/Next loop by using the Exit For statement. The following example searches the Global array names for the name "Fred". If it finds the name, it returns the index of the name in the array. Otherwise it returns . For example, if the names array is:

```
Array ("Frank", "Helen", "Fred", "Linda")
```

Then the formula returns 3.

```
Global names () As String
'The names array has been initialized and filled
'in other formulas
Dim i
formula = 'The UBound function returns the size of its array
'argument
```

```
For i = 1 to UBound (names)
If names (i) = "Fred" Then
formula = i
Exit For
End If
Next i
```

Do Loops (Basic syntax)

Another looping mechanism is the Do loop. A Do loop can be used to execute a fixed block of statement an indefinite number of time.

The 4 different types of Do loops

Type of Do Loop	Explanation	Example
Do While ... Loop	The Do While ... Loop evaluates the condition, and if the condition is true, then it evaluates the statements following the condition. When it has finished doing this, it evaluates the condition again and if the condition is true, it evaluates the statements again. It continues repeating this process until the condition is false.	Do While *condition* *statements* Loop
Do Until ... Loop	The Do Until ... Loop is similar to the Do While ... Loop except it keeps evaluating the statements *until* the condition is true rather than *while* it is true.	Do Until *condition* *statements* Loop
Do ... Loop While	The Do ... Loop While evaluates the statements only once. It then evaluates the condition, and if the condition is true, evaluates the statements again. This process continues until the condition is false.	Do *statements* Loop While *condition*
Do ... Loop Until	Similar to Do ... Loop While except that it evaluates the statements until the condition is true.	Do *statements* Loop Until *condition*

Note: The Do loops support an Exit Do statement to immediately jump out of the loop. The Exit Do statement is similar to the Exit For in For/Next loops.

Do While ... Loop formula example (Basic syntax)

The following example searches for the first occurrence of a digit in an input string. If a digit is found, it returns its position, otherwise it returns . In this case, the input string is set explicitly to a string constant. If preferred, it could be set to be equal to a String type database field instead.

For example, for the input String, "The 7 Dwarves", the formula returns 5, which is the position of the digit 7.

```
Dim inString
inString = "The 7 Dwarves"
Dim i, strLen
```

```
i = 1
strLen = Len (inString)
formula = Do While i <= strLen And formula = Dim c As String
c = Mid (inString, i, 1)
If IsNumeric (c) Then formula = i
i = i + 1
Loop
```

While loops (Basic syntax)

The While loop is similar to the Do While ... Loop except that it does not support an Exit statement. It uses While ... Wend instead of Do While ... Loop as its syntax.

```
While condition statements Wend
```

Safety mechanism for loops (Basic syntax)

There is a safety mechanism to prevent report processing from hanging due to an infinite loop. Any one evaluation of a formula can have at most 30,000 loop condition evaluations per formula evaluation. For example:

```
Dim i
i = 1
Do While i <= 100000
If i > {movie.STARS} Then Exit Do
i = i + 1
Loop
formula = 20
```

If {movie.STARS} is greater than 30,000 then the loop condition (i <= 100000) will be evaluated more than the maximum number of times and an error message is displayed. Otherwise the loop is OK.

Note: The safety mechanism applies on a per formula base, not for each individual loop. For example:

```
Dim i
i = 1
For i = 1 To 10000
formula = Sin (i)
Next i
Do While i <= 25000
i = i + 1
Loop
```

The above formula also triggers the safety mechanism since the 30,000 refers to the total number of loop condition evaluations in the formula and this formula will have 10001 + 25001 such evaluations.

Limitations (Basic syntax)

For reference purposes, here are the sizing limitations of the formula language:

- The maximum length of a String constant, a String value held by a String variable, a String value returned by a function or a String element of a String array is 254 bytes.
- The maximum size of an array is 1000 elements.
- The maximum number of arguments to a function is 1000. (This applies to functions that can have an indefinite number of arguments such as Choose).
- The maximum length of the text of a formula is 64K.
- The maximum number of loop condition evaluations per evaluation of a formula is 30,000. (See safety mechanisms for loops for the precise meaning of this).
- Date-time functions modeled on Visual Basic accept dates from year 100 to year 9999. Traditional Crystal Reports functions accep dates from year 1 to year 9999.

Creating Formulas with Crystal Syntax

This section provides you with an overview of various parts of a formula. You learn about Crystal syntax and the techniques you can use when creating a formula.

If you are new to Crystal syntax, begin with these topics:

- Crystal syntax overview
- Crystal syntax fundamentals.

Crystal syntax overview

When creating formulas, you have the option of using either Crystal or Basic syntax. Syntax rules are used to create a correct formula. Almost any formula written with one syntax can be written with the other. Reports can contain formulas that use Basic syntax as well as formulas that use Crystal syntax. For information on whether to use Crystal or Basic syntax, see Formula syntax.

What's new in Crystal syntax

Many new features have been added to Crystal syntax for Crystal Reports version 8.

Note: Equivalent versions of these features are also supported by Basic syntax.

New control structures

Crystal syntax supports three new controls structures: the *Select* expression, the *While* loop, and the *For* loop. Select is a good alternative to the *If* expression when you want to use the result of a single expression to select from a number of alternatives. The While and For loops introduce looping into Crystal Reports. Looping is useful when you want to evaluate some expressions in your formula multiple times. A typical application is to extract information from string database fields for custom formatting in a report. In this case, you loop through the characters of the string field.

New functions inspired by Visual Basic

These new functions are intended to work like their counterparts in Visual Basic. However, you do not need to learn Visual Basic or Basic syntax to use them. They provide functionality that could be useful in many Crystal syntax formulas.

Math functions:

Atn, Cos, Exp, Int, Log, Pi, Rnd, Sin, Sgn, Sqr, Tan

Financial functions:

DDB, FV, IPmt, IRR, MIRR, NPer, NPV, Pmt, PPmt, PV, Rate, SLN, SYD

Programming shortcut functions:

Choose, IFF, Switch

Type conversion functions:

CBool, CCur, CDbl, CStr, CDate, CTime, CDateTime

Date and time functions:

DateValue, TimeValue, DateTimeValue, IsDate, IsTime, IsDateTime, MonthName, WeekdayName, DateAdd, DateDiff, DatePart, DateSerial, Timer, TimeSerial

Array and string functions:

Filter, Replace, Join, Split, UBound, InStrRev, StrReverse

New operators

Some new operators have been added to make writing expressions in Crystal syntax easier and more flexible.

Visual Basic style string concatenation: &

Integer division: \

Exponentiation: ^

Logical operators: Eqv, Imp, Xor

Modulus: Mod

Date-time literals: #Jan 3, 1999 10:20 am#

Expanded capabilities for array processing

Crystal syntax now supports the Redim and Redim Preserve keywords to enable you to resize arrays dynamically. It also supports the ability to assign values to an individual element of an array. In addition, Crystal syntax now uses more efficient algorithms to speed up operations with arrays.

More types of record selection formulas can be pushed down to the server

In particular, there is now support for pushing down a record selection formula that involves the IsNull function and the Or operator. For example, the following selection formulas can be pushed down:

```
{Orders.Order ID} < 1020 Or
{Orders.Order Amount} >= 1000
```

```
IsNull ({Customer.Region}) Or {Customer.Region} = "BC"
```

The formula language now also performs simplification of most constant and parameter expressions in formulas at compile time rather than formula evaluation time. What this means is that record selection formulas such as:

```
{date field} > CurrentDate - 3
```

```
{date field} In CDate ({?year parameter}, 6, 1) To CDate ({?year
parameter}, 9, 30)
```

can be pushed down to the server since the expressions in italics are replaced by their values when the formula is compiled, and the resulting formula can then be converted to a SQL query.

Formula language support for new functionality specific to Crystal Reports

Conditional formatting of a cell value in a Cross-tab or OLAP grid based on its row or column values is supported with the new GridRowColumnValue function. The new Percentage summaries are supported in the formula language with the PercentOfSum, PercentOfAverage, PercentOfMaximum, PercentOfMinimum, PercentOfCount and PercentOfDistinctCount functions. You can now use the formula language to conditionally format the font, font style, and font size of a field.

Crystal syntax fundamentals

The result of a formula

The result of a formula, or the value that is printed when the formula is placed in a report, is called the *value returned by the formula*. Every formula in Crystal Reports must return a value. For example, here is a simple Crystal syntax formula that returns a value of 10:

```
10
```

The value returned by a formula can be one of the seven simple data types supported. These are Number, Currency, String, Boolean, Date, Time, and DateTime.

Note: Crystal Reports also supports range types and array types, but these cannot be returned by a formula.

For example, suppose a company has a shipping policy in which orders over $1,000 are insured, but orders below that amount are not insured:

```
//A formula that returns a String value
If {Orders.Order Amount} >= 1000 Then
"Insured shipping"
Else
"Regular shipping"
```

Tip: The text following the two forward slashes is a comment for someone reading this formula and is ignored by the Crystal syntax compiler.

The formula returns the text string value "Insured shipping" if the value of the database field {Orders.Order Amount} is greater than or equal to 1000; otherwise, it returns the text string value "Regular Shipping" otherwise.

Related topics

- Crystal syntax is not case-sensitive
- Practice using the Xtreme sample database.

Crystal syntax is not case-sensitive

For example, the keyword *Then* could equivalently be typed in as *then* or *THEN*. This is true of all variable names, functions, and keywords used in a Crystal syntax formula.

Note: The only exception to this rule is for strings. The string "Hello" is not the same as the string "hello".

Practice using the Xtreme sample database

Many of the examples in this section refer to the Xtreme sample database. This database is included with Crystal Reports.

Browse the Samples\En directory to find the Databases folder that contains the Xtreme database.

Comments (Crystal syntax)

Formula comments are notes included with a formula to explain its design and operation. Comments do not print and they do not affect the formula, but they appear in the Formula Editor. You can use comments to explain the purpose of a formula or explain the steps involved in writing it.

Comments begin with two forward slashes (//) and are followed by the text of the comment. Everything that follows the slashes on the same line is treated as being part of the comment:

```
//This formula returns the string "Hello"
//This is another comment
"Hello" //Comments can be added at the end of a line
//Comments can occur after the formula text
```

Fields (Crystal syntax)

Many of the fields used in the construction of a report can be referred to in your formulas. For example, database, parameter, running total, SQL expression, summary, and group name fields can all be used in a formula. You can also refer to other formula fields in your formula.

The easiest way to insert a field into your report is to double-click a field's name in the Report Fields tree. This ensures that the correct syntax for the field is used.

How fields appear in formulas

Database, parameter, formula, running total and SQL expression fields have their names surrounded by braces. Database field names are taken from the database. For example:

- database field: {Employee.Last Name}

Parameter, formula, running total, and SQL expression field names are specified when the fields are created.

- parameter fields also includes a question mark: {?my parameter field}
- formula fields include an at sign: {@another formula }
- running totals fields include a pound sign: {#my running total}
- SQL expression fields include a percent sign: {%my SQL expression}

Summary and group name fields look like function calls. However, they are really shorthand notation for a report field.

- sum summary field: Sum({Orders.Order Amount}, {Orders.Ship Via})
- group name field: GroupName({Orders.Ship Via})

Sample formulas using fields (Crystal syntax)

The formula in this example uses the Xtreme database. To find out how many days it takes to ship the product from the date when the order was placed, you can just subtract the ship date database field from the order date database field:

```
//A formula that uses database fields
{Orders.Ship Date} - {Orders.Order Date}
```

To find the total dollar amount of a given product that was ordered, multiply its unit price by the quantity ordered:

```
{Orders Detail.Unit Price} * {Orders Detail.Quantity}
```

To calculate a sale price of 80 percent of the original unit price:

```
{Orders Detail.Unit Price} * 0.80
```

Expressions (Crystal syntax)

An *expression* is any combination of keywords, operators, functions, and constant values that result in a value of a given type. For example:

```
//An expression that evaluates to the Number value 25
10 + 20 - 5
//An expression that evaluates to the String value
//"This is a string."
"This is a string."
```

A Crystal syntax formula consists of a sequence of expressions. The value of the final expression is the value returned by the formula and what gets printed. Each expression must be separated from the previous expression by a semicolon (;).

Multiple expressions (Crystal syntax)

Typically, each expression takes one line, but you can continue an expression onto the next line if you need more space.

The formula below consists of five expressions. It returns the Number value 25 since that is the value of the last expression in the formula.

Example

```
//Expressions example
//The first expression. Its value is the Number
//value 30
10 + 20;
//The second expression. Its value is the String
//"Hello World". It takes up two lines.
"Hello " +
"World";
//The third expression. Its value is of Number type
{Orders Detail.Quantity} * 2 - 5;
//The fourth expression. Its value is of String type
```

```
If {Orders Detail.Quantity} > 1 Then
"multiple units"
Else
"one unit";
//The fifth and final expression. Its value is the
//Number value 25
20 + 5
```

Placing a semicolon after the last expression in the formula is also allowed, but optional. For example, the above formula could have ended with:

```
20 + 5;
```

Some of the sample formulas in the preceding section do not have semicolons. This is because they consist of a single expression, and a semicolon is optional after the last expression. Many formulas in Crystal syntax can be written as a single expression.

Notice that there is no semicolon after the "multiple units" String. In fact, if you put a semicolon there, the program will report an error. This is because a semicolon separates expressions, and the

```
Else
"one unit";
```

is not a separate expression. It does not stand alone apart from the If. In fact, it is an integral part of the If expression since it describes the value that the If expression will return under certain circumstances.

Note: The example is not a practical example because the first four expressions in the formula did not have any effect on the last expression.

How earlier expressions affect later expressions (Crystal syntax)

The fact that a Crystal syntax formula is a sequence of expressions whose result is the value of the final expression is the most important concept in understanding Crystal syntax. This expression-based syntax allows you to write very short formulas with a lot of functionality.

Example

```
//First expression. It declares the Number variable x
//and then returns the value of an uninitialized
//Number variable, which is 0.
NumberVar x;
//Second expression. It assigns the value of 30 to x,
//and returns 30.
x := 30
```

The above formula would give an error if the first expression were omitted. This is because the second expression refers to the Number variable x, and the program needs to have x declared before it understands expressions involving it.

In general, you use variables to get the earlier expressions in a formula to affect the final expression. See Variables (Crystal syntax) for more information.

Using the If expression (Crystal syntax)

The *If* expression is one of the most widely used features of Crystal syntax. It also provides insight into the nature of expressions. Consider the earlier If expression as a separate formula. Notice that because this formula is a single expression, it does not need a semicolon:

```
If {Orders Detail.Quantity} > 1 Then
"multiple units"
Else
"one unit"
```

Suppose you wanted to modify this formula so that it either prints "multiple units" or the number 1.

```
//An erroneous formula
If {Orders Detail.Quantity} > 1 Then
"multiple units"
Else
1
```

This formula will result in an error. This is because the values in this expression are different types: "multiple units" is a String value and 1 is a Number value. Crystal Reports requires that the value of an expression always be of a single type.

Note: This example can be corrected by using the CStr function to convert the Number 1 to a String value. For example, the Number 1 is converted to the string "1" by calling CStr (1, 0).

```
//A correct formula
If {Orders Detail.Quantity} > 1 Then
"multiple units"
Else
CStr (1, 0) //Use 0 decimals
```

See Control structures (Crystal syntax) for more information on the If expression.

Assignment (Crystal syntax)

The assignment operator is a colon followed by an equals sign (:=).

Example

```
//Assign the Number value of 10 to the variable x
x := 10;
//Assign the String value of "hello" to the
//variable named greeting
greeting := "hello";
```

The equality operator (=) is used to check when two values are equal. A common error is to use the equality operator when the assignment operator is actually intended. This can generate a mysterious error message or no error message at all since it is often syntactically correct to use the equality operator. For example:

```
greeting = "hello";
```

The above formula checks if the value held by the variable greeting is equal to the value "hello". If it is, then the expression's value is *True*, and if is not then the expression's value is

False. In any case, it is a perfectly correct Crystal syntax expression (assuming that the greeting is a String variable).

Simple data types (Crystal syntax)

The simple data types in Crystal Reports are Number, Currency, String, Boolean, Date, Time, and DateTime.

Number
Enter numbers without any comma separators or currency symbols (generally you would want to have formatted numbers appearing as the result of a formula and not in the formula itself).

Example

```
100000
1.23
```

Currency (Crystal syntax)

Use the dollar sign ($) to create a Currency value.

Example

```
$10000
-$20
$1.23
```

You can also use the CCur function. The initial C in CCur stands for convert and it can be used to convert Number values to Currency values:

```
CCur (10000)
CCur (0)
CCur (1.23)
```

String (Crystal syntax)

Strings are used to hold text. The text must be placed between double quotation marks (") or apostrophes (') and cannot be split between lines. If you want to include double quotation marks in a string delimited by double quotation marks, use two consecutive double quotation marks. Similarly, if you want to include an apostrophe in a string delimited by apostrophes, use two consecutive apostrophes.

Example

```
"This is a string."
"123"
"The word ""hello"" is quoted."
'This is also a string.'
'123'
'Last Year''s Sales'
```

If you use double quotes for the left side of the string, you must use double quotes on the right side. Similarly for apostrophes. The following example is incorrect:

```
'Not a valid string."
```

You can extract individual elements or substrings from a string by specifying the character position or a range of character positions. Negative values are allowed; they specify the position starting from the end of the string.

```
"hello" [2] //Equal to "e"
"hello" [] //Equal to "h"
"60455234" [1 to 3] //Equal to "604"
"abcdef" [ to ] //Equal to "def"
```

You can also extract substrings from a string using the Left, Right and Mid functions.

Boolean (Crystal syntax)

The valid Boolean values are:

```
True
False
```

Note: *Yes* can be used instead of True and *No* instead of False.

Date, Time, and DateTime (Crystal syntax)

The DateTime type can hold date-times, dates only, or times only and thus is rather versatile. The Date type holds dates only and the Time type holds times only. The Date and Time types are more efficient than the DateTime type, and so can be used in situations where the added functionality and flexibility of the DateTime type is not needed.

You can create DateTime values directly using the date-time literal construction, which is formed by typing in the date-time between two pound (#) signs. Many different formats are supported.

Note: Date-time literals cannot be split between lines.

Examples

```
#8/6/1976 1:20 am#
#August 6, 1976#
#6 Aug 1976 13:20:19#
#6 Aug 1976 1:30:15 pm#
#8/6/1976#
#10:20 am#
```

Even though #10:20 am# looks like it could have the Time type and
#8/6/1976# looks like it could have the Date type, they do not. They both have the DateTime type, as do all date-time literals. For example, you can think of #10:20 am# as a DateTime value with a null date part. To convert it to the Time type use CTime (#10:20 am#).

Instead of using date-time literals, you can use CDateTime to convert a String to a DateTime. For example,

```
CDateTime ("8/6/1976 1:20 am")
CDateTime ("10:20 am")
```

However, there is one key difference between using date-time literals and the above usage of CDateTime. Date-time literals always use U.S. English date formats rather than settings from the locale of the particular computer on which Crystal Reports is running. Thus, the date-time literal examples above would work on all computers. On the other hand, on a French system, you could use constructions like:

```
CDateTime ("22 aout 1997") //Same as #Aug 22, 1997#
```

Date values can be constructed with CDate and Time values with CTime:

```
CDate ("Aug 6, 1969")
CDate (1969, 8, 6) //Specify the year, month, day
//Converts the DateTime argument to a Date
CDate (#Aug 6, 1969#)
CTime ("10:30 am")
CTime (10, 30, 0) //Specify the hour, minute, second
CTime (#10:30 am#)
```

Range data types (Crystal syntax)

Ranges are designed to handle a spectrum of values. Range types are available for all the simple types except for Boolean. That is: Number Range, Currency Range, String Range, Date Range, Time Range and DateTime Range. You can generate ranges using the To, _To, To_, _To_, UpTo, UpTo_, UpFrom, and UpFrom_ keywords. In general, To is used for ranges with 2 endpoints, and UpTo and UpFrom are used for open ended ranges (only one endpoint). The underscores are used to indicate whether or not the endpoints are in the range.

Examples of Number Range values

The range of numbers from 2 to 5 including both 2 and 5

```
2 To 5
```

The range of numbers from 2 to 5, not including 2 but including 5

```
2 _To 5
```

All numbers less than or equal to 5

```
UpTo 5
```

All number less than 5

```
UpTo_ 5
```

Examples of DateTime Range values:

```
#Jan 5, 1999# To #Dec 12, 2000#
UpFrom #Jan 1, 2000#
```

Using ranges in formulas (Crystal syntax)

There are twenty-seven functions in Crystal Reports that specify date ranges. For example, the function LastFullMonth specifies a range of date values that includes all dates from the first to last day of the previous month. So if today's date is September 15, 1999 then LastFullMonth is the same as the range value CDate (#Aug 1, 1999#) To CDate (#Aug 31, 1999#).

Ranges are often used with If or Select expressions. The following example computes student letter grades based on their test scores. Scores greater than or equal to 90 receive an "A", scores from 80 to 90, not including 90, receive a "B" and so on.

```
//Compute student letter grades
Select {Student.Test Scores}
Case UpFrom 90 :
"A"
Case 80 To_ 90 :
"B"
Case 70 To_ 80 :
"C"
Case 60 To_ 70 :
"D"
Default :
"F";
```

The above example uses the Select expression which is discussed in more detail in the control structures section. You can check if a value is in a range by using the In operator. For example:

```
5 In 2 To 10; //True
5 In 2 To_ 5; //False
5 In 2 To 5; //True
```

The Maximum and Minimum functions can be used to find the endpoints of a range:

```
Maximum (2 To 10) //Returns 10
```

Array data types (Crystal syntax)

Arrays in Crystal Reports are ordered lists of values that are all of the same type. These values are known as the array's *elements*. The elements of an array can be any simple type or range type. Arrays can be created using square brackets ([ ]).

Examples

An array of 3 Number values. The first element is 10, the second is 5 and the third is 20.

```
[10, 5, 20]
```

An array of 7 String values:

```
["Sun", "Mon", "Tue", "Wed", "Th", "Fri", "Sat"]
```

An array of 2 DateTime Range values:

```
[#Jan 1, 1998# To #Jan 31, 1998#,
#Feb 1, 1999# To #Feb 28, 1999#]
```

You can extract individual elements out of an array using square brackets containing the index of the element you want. This is called *subscripting* the array:

```
[10, 5, 20] [2] //Equal to 5
```

Number ranges can also be used to subscript arrays. The result is another array. For example:

```
[10, 5, 20] [2 To 3] //Equal to [5, 20]
```

Arrays are most useful when used with variables and so will be discussed in more detail in the Variables (Crystal syntax) section. Using variables, you can change the individual elements of an array and resize the array to accommodate more elements. For example, you can accumulate database field values into a global array variable in a detail level formula, and then use a formula in a group footer to perform a calculation based on those values. This enables you to perform a wide variety of customized summary operations.

Variables (Crystal syntax)

This section describes the key components of variables and shows you how to create variables and assign values to them.

Variables overview

A variable represents a specific data item, or value, and acts as a placeholder for that value. When a formula encounters a variable, the formula searches for the value of the variable and uses it in the formula. Unlike a constant value, which is fixed and unchanging, a variable can be repeatedly assigned different values. You assign a value to a variable and the variable maintains the value until you later assign a new value. Because of this flexibility, it is necessary for you to declare variables before you use them so that Crystal Reports is aware of them and understands how you intend to use them.

Example

If you wanted to report on customers by area code, you could create a variable that extracts the area code from a customer fax number. The following is an example of a variable called *areaCode*:

```
Local StringVar areaCode;
areaCode := {Customer.Fax} [1 To 3];
```

The first line of the variable example is the variable declaration; it gives the variable a name and type. The database field {Customer.Fax} is a String field and [1 To 3] extracts the first 3 characters from its current value. The variable areaCode is then assigned this value.

Variable declarations (Crystal syntax)

Before using a variable in a formula, you must declare it.

A variable can hold values of a given type. The allowed types are the seven simple types (Number, Currency, String, Boolean, Date, Time and DateTime), the six range types (Number Range, Currency Range, String Range, Date Range, Time Range and DateTime Range) and variables that hold arrays of the previously mentioned types. This gives a total of 26 different types that a variable can have.

When you declare a variable, you also specify its name. A variable cannot have the same name as any function, operator or other keyword that is valid for Crystal syntax. For example, your variable cannot be named Sin, Mod or If because Sin is a built in function, Mod is a built in operator and If is a built in keyword. When typing formulas in the formula editor, the names of the built in functions, operators and other keywords are highlighted in a different color and so it is easy to check if the variable name conflicts.

Once a variable is declared, it can be used in the formula. For example, you might want to assign it an initial value:

```
Local NumberVar x; //Declare x to be a Number variable
x := 10; //Assign the value of 10 to x
```

Note: The keyword for declaring the Number variable has a *Var* at the end. This is true for all the variable types in Crystal syntax.

A variable can only hold values of one type. For example, if a variable holds a Number value, you cannot later use it to hold a String.

Example

```
Local StringVar y;
y := "hello";
//OK- the Length function expects a String argument
Length (y);
//Error- y can only hold String values
y := #Jan 5, 1993#;
//Error- y can only hold String values
y := ["a", "bb", "ccc"];
//Error- the Sin function expects a Number argument
Sin (y);
```

You can combine declaring a variable and assigning it a value in a single expression. For example:

```
Local NumberVar x := 10 + 20;
Local StringVar y := "Hello" + " " + "World";
Local DateVar z := CDate (#Sept 20, 1999#);
Local NumberVar Range gradeA := 90 To 100;
```

This is a good practice because it is more efficient and helps prevent the common mistake of having incorrectly initialized variables.

Here are some more examples of declaring and initializing range variables:

```
Local NumberVar Range gradeA;
Local DateVar Range quarter;
gradeA := 90 To 100;
quarter := CDate (1999, 10, 1) To CDate (1999, 12, 31);
```

Variable Scope (Crystal syntax)

Variable scopes are used to define the degree to which variables in one formula are made available to other formulas. There are three levels of scope in Crystal Reports: local, global and shared. Every variable has a scope, and this scope is specified when the variable is declared.

Local Variables

Variables with local scope, also known as local variables, are declared using the Local keyword followed by the type name (with the Var suffix) followed by the variable name as in the above examples.

Local variables are restricted to a single formula and a single evaluation of that formula. This means that you cannot access the value of a local variable in one formula from a different formula.

Example

```
//Formula A
Local NumberVar x;
x := 10;
//Formula B
EvaluateAfter ({@Formula A})
Local NumberVar x;
x := x + 1;
```

The function call EvaluateAfter ({@Formula A}) ensures that Formula B will be evaluated after Formula A is evaluated. Formula A returns a value of 10 and Formula B returns a value of 1. Formula B does not have access to Formula A's x and thus cannot use the value of 10 and add 1; instead, it uses the default value for the uninitialized local variable x found in Formula B, which is 0, and adds 1 to it to get 1.

You can also create local variables with the same name but different types in different formulas. For example, the type declarations in formulas A and B do not conflict with:

```
//Formula C
Local StringVar x := "hello";
```

Local variables are the most efficient of the three scopes. They also do not interfere with one another in different formulas. For these reasons, it is best to declare variables to be local whenever possible.

Global variables (Crystal syntax)

Global variables use the same memory block to store a value throughout the main report. This value is then available to all formulas that declare the variable, except for those in subreports. You declare a global variable as in the following example:

```
Global StringVar y;
```

You can also omit the Global keyword which creates a Global variable by default:

```
StringVar y; //Same as: Global StringVar y;
```

However, even though global variables are easy to declare, it is recommended that you use them only when local variables do not suffice.

Since global variables share their values throughout the main report, you cannot declare a global variable in one formula with one type and then declare a global variable with the same name in a different formula with a different type.

Example

```
//Formula A
Global DateVar z;
z := CDate (1999, 9, 18)
//Formula B
NumberVar z;
z := 20
```

In this case, if you enter and save Formula A first, Crystal Reports will return an error when you check or try to save Formula B. This is because the declaration of the Global variable z as a Number conflicts with its earlier declaration in Formula A as a Date.

Using Global variables

Global variables are often used to perform complex calculations where the results of a formula depend upon the grouping and page layout of the actual printed report. This is accomplished by creating several formulas, placing them in different sections of the report, and having the different formulas interact via global variables.

Example

```
//Formula C
Global NumberVar x;
x := 10;
//Formula D
//Call the function WhileReadingRecords
WhileReadingRecords;
Global NumberVar x;
x := x + 1
```

If Formula C is placed in the Report Header and then Formula D is placed in a detail section, Formula C will be evaluated before Formula D. Formula C will be evaluated once and then Formula D will be evaluated for each record appearing in the detail section. Formula C returns 10. For the first detail record, Formula D returns 11. This is because the value 10 of x is retained from when it was set by Formula C. Formula D then adds 1 to this value, setting x to 11 and then returns 11. For the second detail record, formula D return 12, adding 1 to the previously retained value of x which was 11. This process continues for the remaining detail records.

The call to WhileReadingRecords tells Crystal Reports to re-evaluate Formula D as it reads in each record of the report. Otherwise, since the formula does not contain any database fields, the program evaluates it only once before reading the records from the database. The formula will then return the value 11 instead of 11, 12, 13, ... as the successive records are processed.

If the expression x := x + 1 is replaced by x := x + {Orders Detail.Quantity}, you create the effect of a running total based on {Orders Detail.Quantity}, although it is one starting at 10 rather than 0 because of Formula C. In this case, you can omit the call to WhileReadingRecords, since it will automatically occur because the formula contains a database field.

Shared variables (Crystal syntax)

Shared variables use the same memory block to store the value of a variable throughout the main report and all of its subreports. Thus shared variables are even more general than global variables. To use a shared variable, declare it in a formula in the main report as in the following example:

```
Shared NumberVar x := 1000;
```

and declare it in a formula in the subreport as in the following example:

```
Shared NumberVar x;
```

In order to use shared variables, the variable must be declared and assigned a value before it can be passed between the main report and the subreport.

Declaring array variables (Crystal syntax)

You can declare array variables by following the type name with the keyword Array.

Example

```
//Declare x to be a Global variable of
//Number Array type
Global NumberVar Array x := [10 , 20, 30];
//cost is a Global variable of Currency Array type
//It is automatically Global since the scope specifier
//(one of Local, Global or Shared) is omitted.
CurrencyVar Array cost := [$19.95, $79.50, $110.00,
$44.79, $223.99];
//payDays is a Global variable of Date Array type
Global DateVar Array payDays := [CDate(1999, 5, 15),
CDate(1999, 5, 31)];
//y is a Shared variable of String Range Array type
Shared StringVar Range Array y := ["A" To "C",
"H" To "J"];
//days is a Local variable of String Array type
Local StringVar Array days;
days := ["Sun", "Mon", "Tue", "Wed", "Th",
"Fri", "Sat"];
```

Using Arrays with For loops

Arrays are commonly used with For loops. The following example creates and then uses the Array [10, 20, 30, ..., 100] using a For loop. See For loops (Crystal syntax) for more details.

```
Local NumberVar Array b;
Redim b[10];
Local NumberVar i;
For i := 1 To 10 Do
(
b[i] := 10 * i
);
b [2] //The formula returns the Number 20
```

Default values for the simple types (Crystal syntax)

An uninitialized variable will have the default value for its type. In general, it is not a good programming practice to rely on the default values of types. For example, initialize all local variables in your formula, initialize all global variables in a formula placed in the Report Header and initialize all shared variables in a formula placed in the Report Header of the main report.

When an array is resized using the Redim keyword, the entries are filled with default values for the type. It is useful to know about default values when using If and Select expressions.

Default values

N umber

```
0
```

Currency

```
$0
```

String

```
"" //The empty string
```

Date

```
Date (0, 0, 0) //The null Date value
```

Time

The null Time value. Value held by an uninitialized Time variable.

DateTime

The null DateTime value. Value held by an uninitialized DateTime variable.

Note: It is not recommended that your formulas rely on the values of uninitialized range or array variables.

Automatic type conversions (Crystal syntax)

Generally in Crystal Reports, values of one type cannot be used where values of another type are expected without explicitly supplying a type conversion function. For example:

```
Local StringVar postalCode;
//Error- assigning a Number value to a String
postalCode := 10025;
//OK - use the type conversion function CStr
//to create "10025"
postalCode := CStr (10025, 0);
```

However, there are a few conversions that are made automatically:

- Number to Currency
- Date to DateTime
- Simple type to Range value of the same underlying simple type

For example, the following assignments are correct:

```
Local CurrencyVar cost;
//Same as: cost := $10
cost := 10;
Local DateTimeVar orderDate;
//Same as: orderDate := CDateTime (1999, 9, 23, 0, 0, 0)
```

```
orderDate := CDate (1999, 9, 23);
Local NumberVar Range aRange;
//Same as: aRange := 20 To 20
aRange := 20;
Local NumberVar Range Array aRangeArray;
//Same as : aRangeArray := [10 To 10, 20 To 25, 2 To 2]
aRangeArray := [10, 20 To 25, 2];
```

Note: The opposite conversions are not allowed. For example:

```
Local NumberVar num;
num := 5 + $10; //Error
//OK- convert to Number type using the CDbl function
num := CDbl (5 + $10) //Could also use ToNumber
```

5 is converted to $5 and added to $10 to make $15. However, this Currency value cannot be automatically assigned to the Number variable num since automatic conversions from Currency to Number are not allowed. Similarly, functions accepting a Currency argument can be supplied a Number argument instead, and the Number argument will be converted to a Currency, whereas functions accepting a Number argument cannot be supplied a Currency argument without first explicitly converting the Currency to a Number using CDbl.

Functions (Crystal syntax)

When using a function in a formula, type the name of the function and supply the arguments required. For example, the Length function requires a String argument and computes the length of the string.

```
Local StringVar x := "hello";
Length (x) //The formula returns the Number 5
```

Supplying arguments of the incorrect type required by the function produces an error. For example, calling Length (3) would produce an error since Length does not accept a Number argument. Functions sometimes can accept different numbers of arguments or types of arguments. For example, the CDate function which could accept a single String argument to form a Date value or 3 Number values holding the year, month and day respectively and form a Date value from them. See Date, Time, and DateTime (Crystal syntax).

Example with the Mid function

```
Local StringVar x := "hello";
Local StringVar y;
//Start at position 2, go to the end of the string
y := Mid (x, 2); //y is now "ello"
//Start at position 2, extract 1 character
y := Mid (x, 2, 1) //y is now "e"
```

These classes of functions are: Math, Summary, Financial, String, Date/Time, Date Range, Array, Type Conversion, Programming Shortcuts, Evaluation Time, Print State, Document Properties and Additional Functions. There are also some functions specific to conditional formatting formulas.

Non reporting-specific functions (Crystal syntax)

The Math, Financial, String, Date/Time, Type Conversion, and Programming Shortcuts groups consist mainly of functions that are not specific to reporting, but might be found in any full featured programming environment. Many of the functions are similar in functionality to Visual Basic functions of the same or similar name.

Note: Some functions that are supported by Crystal syntax are not listed in the Crystal syntax Functions tree. This is because they are equivalent to Crystal syntax functions already listed in the tree.

For example, the Length function is the traditional Crystal syntax function for finding the length of a string. Crystal syntax also supports Len as a synonym. Len is the Visual Basic and Basic syntax function for this and its inclusion is for the convenience of Visual Basic and Basic syntax users who want to write or modify Crystal syntax formulas.

Summary functions (Crystal syntax)

The Summary function group provides functions for creating summary fields such as:

```
Sum({Orders.Order Amount}, {Orders.Ship Via})
```

Summary fields are normally created using the Insert Summary or Insert Grand Total dialogs. They then appear in the Available Fields tree, and can be used in a formula by double clicking there. However, they do not need to be created in this way. You can create a summary field exclusively for use by your formula by filling in the arguments to one of the functions in the Summary functions section appropriately. However, any groups in the report that the summary field refers to must already exist in the report.

Date Ranges (Crystal syntax)

This category of functions was discussed in the section Range data types (Crystal syntax). One additional comment is that the Date ranges produced by these functions depend on the current date. For example, if today's date is September 18, 1999, then LastFullMonth is the Date Range value:

```
CDate(#Aug 1, 1999#) To CDate(#Aug 31, 1999#)
```

This functionality is often useful, but what if you want to determine a date range based on a database field such as {Orders.Order Date}? The Date/Time functions can be used instead.

For example:

```
Local DateVar d := CDate ({Orders.Order Date});
Local DateVar Range dr;
dr := DateSerial (Year(d), Month(d) - 1, 1) To
DateSerial (Year(d), Month(d), 1 - 1);
//At this point dr is the Date Range value holding
//the last full month before {Orders.Order Date}
```

The DateSerial function makes this easy because you don't have to worry about special cases. It never lets you create an invalid date. For example, DateSerial (1999, 1 - 1, 1) is December 1, 1998.

Note: In the above example, {Orders.Order Date} is actually a DateTime field and so the CDate function is used to convert it to a date by truncating the time part.

Array functions (Crystal syntax)

The array functions compute summaries of an array's elements. For example, the Sum function when applied to an array returns the sum of the elements of the array. For example, the following formula returns 100:

```
Sum ([10, 20, 30, 40])
```

Evaluation Time functions (Crystal syntax)

These are the reporting specific functions BeforeReadingRecords, WhileReadingRecords, WhilePrintingRecords and EvaluateAfter. You can use these functions to guide Crystal Reports as to when your formula should be evaluated.

Should the formula be evaluated before retrieving the records from the database, while reading the records from the database but before the records have been grouped, sorted and summarized, or while printing the report, when the records are grouped, sorted and summarized? In general, Crystal Reports sets an appropriate evaluation time for your formula, based on how much information the formula needs. For example, if a formula uses a database field, then it cannot be evaluated before the records are read from the database. However, you sometimes need to force a later evaluation time than normal to get the desired effect. See Global variables (Crystal syntax) for an example.

Print State functions (Crystal syntax)

These are once again reporting specific functions. For example, the notation {Orders.Order Date} is used to refers to the value of the field in the current record whereas Previous ({Orders.Order Date}) refers to the value in the immediately preceding record and Next ({Orders.Order Date}) in the next record. IsNull ({Orders.Order Date}) checks if the field's value is null.

Other examples are PageNumber and TotalPageCount that you can use to access pagination information about your report.

Document Properties functions

These are specific functions, such as PrintDate and ReportTitle, that refer to the report document as a whole.

Additional functions (Crystal syntax)

These are functions that are in User Function Libraries or UFLs. A UFL is a separate dynamic link library or Automation server that you create and that Crystal Reports can use to add your own customized functions to the formula language. Writing a UFL is more involved than writing a formula using Basic syntax or Crystal syntax. See Crystal Reports Developer's Help for details.

Note: Using UFLs makes your reports less portable because you must distribute your UFL along with the report.

Conditional formatting functions (Crystal syntax)

When writing a conditional formatting formula, certain additional functions appear at the top of the Functions tree to help you with this. For example, you can format the {Customer.Last Year's Sales} field to print sales of more than $100,000 in green, sales of less than $15,000 in red, and all other sales in black.

Example

```
//Conditional formatting example 1
If {Customer.Last Year's Sales} > 100000 Then
crGreen
Else If {Customer.Last Year's Sales} < 15000 Then
crRed
Else
crBlack
```

Since this is a font color formatting function, the list of Color Constants appears in the Functions Tree. This example uses three: crGreen, crRed and crBlack. You could have used the actual numeric values of the color constants instead. For example, crRed is 255 and crGreen is 32768. However, the formula is more understandable using the color constants. All constant functions in Crystal syntax can have the "cr" prefix.

Crystal syntax still supports constant functions from previous versions without the "cr" prefix. For example, you can use "Red" instead of "crRed". However, using the "cr" prefix organizes constant functions and is recommended.

Note: Some formatting attributes do not use constant functions. For example, if you wanted to not print {Customer.Last Year's Sales} values if the sales were less than $50,000, you could write the following conditional formatting formula for the suppress attribute:

```
//Conditional formatting example 2
If {Customer.Last Year's Sales} < 50000 Then
True //suppress the value
Else
False //do not suppress the value
```

Or more simply:

```
//Conditional formatting example 3 -
//equivalent to example 2
{Customer.Last Year's Sales} < 50000
```

If the last year's sales are less than $50,000, then the expression

```
{Customer.Last Year's Sales} < 50000
```

is True, and so the formula returns True. On the other hand, if the last year's sales are greater than or equal to $50,000, then

```
{Customer.Last Year's Sales} < 50000
```

is False and so the formula returns False.

General purpose conditional formatting functions (Crystal syntax)

There are three general purpose conditional formatting functions:

- CurrentFieldValue
- DefaultAttribute
- GridRowColumnValue.

These functions are displayed at the top of the Functions tree whenever appropriate. DefaultAttribute can be used for any formatting formula, CurrentFieldValue for any formatting formula where you are formatting a field value, and GridRowColumnValue for any formatting formula where you are formatting a field value in a Cross-tab or OLAP grid.

In particular, CurrentFieldValue allows you to conditionally format the cells of a Cross-tab or OLAP grid based on their value while GridRowColumnValue lets you conditionally format the cells of a Cross-tab or OLAP grid based on the values of the row or column headings. These two functions are essential in this situation since there is no other way in the formula language to refer to the values of these fields. For example, if you wanted Cross-tab cells to be suppressed if the values are less than 50,000:

```
//Conditional formatting example 4
CurrentFieldValue < 50000
```

Operators (Crystal syntax)

Arithmetic operators

The arithmetic operators are addition (+), subtraction (-), multiplication (*), division (/), integer division (\), percent (%), modulus (Mod), negation (-) and exponentiation (^). Arithmetic operators are used to combine numbers, numeric variables, numeric fields and numeric functions to get another number.

Examples

```
//Outstanding preferred stock as a percent of
//common stock
{Financials.Preferred Stock} %
{Financials.Common Stock};
//The square root of 9, Sqr(9) is 3
//The formula returns 17
7 + 2 * 3 - 2 + Sqr(6 + 3) * Length("up");
```

Order of precedence (Crystal syntax)

When you create arithmetic expressions that involve several operators, the order that the program evaluates the various elements of the expression becomes important. In general, the program evaluates expressions from left to right. However, it also follows the rules of precedence from basic math.

Example

Multiplication and division are performed first from left to right and then addition and subtraction are performed.

For example, 5 + 10 * 3 = 5 + 30 = 35. You can change this order of precedence by using parentheses. For example, (5 + 10) * 3 = 15 * 3 = 45. If you are unsure of the order of precedence, it is a good idea to clarify your intentions with parentheses.

List of arithmetic operators, from highest precedence to lowest

- Exponentiation (^)
- Negation (-)
- Multiplication, division, and percent (*, /, %)
- Integer Division (\)
- Modulus (Mod)
- Addition and subtraction (+, -).

Comparison operators (Crystal syntax)

The comparison operators are equal (=), not equal (<>), less than (<), less than or equal (<=), greater than (>) and greater than or equal (>=).

Comparison operators are usually used to compare operands for a condition in a control structure such as an If expression. Comparison operators as a group all have lower precedence than the arithmetic operators. Thus, expressions like 2 + 3 < 2 * 9 are the same as (2 + 3) < (2*9).

Boolean operators (Crystal syntax)

The Boolean operators are, in order of precedence from greatest to lowest: Not, And, Or, Xor, Eqv and Imp. Boolean operators are typically used with comparison operators to generate conditions for control structures. Boolean operators as a group have lower precedence than the comparison operators. Thus for example, the expression 2 < 3 And 4 >= is the same as (2 < 3) And (4 >=).

Null fields and how to use IsNull (Crystal syntax)

In general, when Crystal Reports encounters a null valued field in a formula, it immediately stops evaluating the formula and produces no value. If you want to handle null field values in your formula, you must explicitly do so using one of the special functions designed for handling them: IsNull, PreviousIsNull or NextIsNull.

Example

The {Product.Color} field contains both basic colors such as "red" and "black" and more descriptive two word colors such as "steel satin" and "jewel green". Here's an example of a formula that writes out "basic" for the basic colors and "fancy" for the others.

```
If InStr({Product.Color}, " ") = 0 Then
formula = "basic"
```

```
Else
formula = "fancy"
End If
```

The function call to InStr searches the {Product.Color} string for a space. If it finds a space, it returns the position of the space, otherwise it returns 0. Since basic colors are only one word with no spaces, InStr will return 0 for them.

For some products, such as the Guardian Chain Lock, a color value was not recorded and so the {Product.Color} field has a null value in the database for that record. Thus, the Guardian Chain Lock record does not have any word printed beside it.

Here is an example of how to fix the above example using IsNull:

```
If IsNull({Product.Color}) Or

   InStr({Product.Color}, " ") = 0 Then

   "basic"

Else

   "fancy"
```

Relating to operators, when Crystal Reports evaluates the condition:

```
IsNull({Product.Color}) Or

InStr({Product.Color}, " ") = 0
```

It first evaluates IsNull ({Product.Color}), and when it determines that this is True, it knows that the whole condition is True, and does not need to check whether

```
InStr({Product.Color}, " ") = 0
```

In other words, Crystal Reports will stop evaluating a Boolean expression when it can predict the results of the whole expression. In the following example, the formula guards against attempting to divide by zero in the case that denom is 0:

```
Local NumberVar num;

Local NumberVar denom;

...

If denom <> 0 And num / denom > 5 Then

...
```

Dealing with null values (Crystal syntax)

For some products, such as the Guardian Chain Lock, a color value was not recorded and so the {Product.Color} field has a null value in the database for that record. In general, when Crystal Reports encounters a null valued field in a formula, it immediately stops evaluating the formula and produces no value. Thus, the Guardian Chain Lock record does not have any word printed beside it. If you want to handle null field values in your formula, you must explicitly do so using one of the special functions designed for handling them: IsNull, PreviousIsNull or NextIsNull.

Here is how to fix up the previous example using IsNull:

```
If IsNull({Product.Color}) Or
InStr({Product.Color}, " ") = 0 Then
"basic"
Else
"fancy"
```

How this relates to operators, is that when Crystal Reports evaluates the condition:

```
IsNull({Product.Color}) Or
InStr({Product.Color}, " ") = 0
```

It first evaluates IsNull ({Product.Color}), and when it determines that this is True, it knows that the whole condition is True, and thus does not need to check whether

```
InStr({Product.Color}, " ") = 0
```

In other words, Crystal Reports will stop evaluating a Boolean expression when it can deduce the results of the whole expression.
In the following example, the formula guards against attempting to divide by zero in the case that denom is 0:

```
Local NumberVar num;
Local NumberVar denom;
...
If denom <> 0 And num / denom > 5 Then
...
```

Control structures (Crystal syntax)

Formulas without control structures execute each expression in the formula exactly once when the formula is evaluated. The expressions are executed in a sequential fashion, from the first expression in the formula to the last. Control structures enable you to vary this rigid sequence. Depending upon which control structure you choose, you can skip over some of the expressions or repeatedly evaluate some expressions depending on if certain conditions hold. Control structures are the primary means of expressing business logic and typical report formulas make extensive use of them.

If expressions (Crystal syntax)

The *If* expression is one of the most useful control structures. It allows you to evaluate an expression if a condition is true and evaluate a different expression otherwise.

Note: When formatting with conditional formulas, always include the Else keyword; otherwise, values that don't meet the If condition may not retain their original format. To prevent this, use the DefaultAttribute function (`If...Else DefaultAttribute`).

Example

A company plans to pay a bonus of 4 percent to its employees except for those who work in Sales who will receive 6 percent. The following formula using an If expression would accomplish this:

```
//If example 1
If {Employee.Dept} = "Sales" Then
{Employee.Salary} * 0.06
```

```
Else
{Employee.Salary} * 0.04
```

In this example, if the condition {Employee.Dept} = "Sales" evaluates as true, then the

```
{Employee.Salary} * 0.06
```

expression is processed. Otherwise the expression following the *Else*, namely the

```
{Employee.Salary} * 0.04
```

is processed.

Suppose another company wants to give employees a 4% bonus, but with a minimum bonus of $1,000. The following example shows how. Notice that the *Else* clause is not included; it is optional, and not needed in this case.

```
//If example 2
Local CurrencyVar bonus := {Employee.Salary} * 0.04;
If bonus < 1000 Then
bonus := 1000;
//The final expression is just the variable 'bonus'.
//This returns the value of the variable and is the
//result of the formula
bonus
```

Another way of accomplishing example 2 is to use an Else clause:

```
//If example 3
Local CurrencyVar bonus := {Employee.Salary} * 0.04;
If bonus < 1000 Then
1000
Else
bonus
```

Now suppose that the previous company also wants a maximum bonus of $5,000. You now need to use an Else If clause. The following example has only one Else If clause, but you can add as many as you need. Note, however, that there is a maximum of one Else clause per If expression. The Else clause is executed if none of the If or Else If conditions are true.

```
//If example 4
Local CurrencyVar bonus := {Employee.Salary} * 0.04;
If bonus < 1000 Then
1000
Else If bonus > 5000 Then
5000
Else
bonus;
```

If example (Crystal syntax)

Suppose that a company wants to compute an estimate of the amount of tax an employee needs to pay and write a suitable message. Income below $8,000 is not taxed, income from $8,000 to $20,000 is taxed at 20% income from $20,000 to $35,000 is taxed at 29% and income above $35,000 is taxed at 40%.

```
//If example 5
Local CurrencyVar tax := 0;
Local CurrencyVar income := {Employee.Salary};
Local StringVar message := "";
If income < 8000 Then
(
message := "no";
tax := 0
)
Else If income >= 8000 And income < 20000 Then
(
message := "lowest";
tax := (income - 8000)*0.20
)
Else If income >= 20000 And income < 35000 Then
(
message := "middle";
tax := (20000 - 8000)*0.20 + (income - 20000)*0.29
)
Else
(
message := "highest";
tax := (20000 - 8000)*0.20 + (35000 - 20000)*0.29 +
(income - 35000)*0.40
);
//Use 2 decimal places and the comma as a
//thousands separator
Local StringVar taxStr := CStr (tax, 2, ",");
"You are in the " & message & " tax bracket. " &
"Your estimated tax is " & taxStr & "."
```

Note: The use of variables is to simplify the logic of the computation. Also, there are 2 expressions that are executed when one of the conditions are met; one assigns the tax variable, and the other assigns the message variable. It is often useful to have multiple expressions executed as a result of a condition.

More details on If expressions (Crystal syntax)

The *If* expression is an expression. In other words it evaluates to a value of a given type. If there is no Else clause, and the condition is not true, then the value is the default value for the type. For example:

```
If Length ({Employee.First Name}) < 5 Then
"short"
```

The above If expression returns a String value. The string value is "short" if the Employee's first name has fewer than 5 letters and the empty String "" otherwise.

Consider the formula:

```
If Year({Orders.Order Date}) >= 1995 Then
{Orders.Order Date}
```

For order dates before 1995, the above If expression returns the null DateTime value. It is a DateTime value rather than a Date value since {Orders.Order Date} is a DateTime database field. The null DateTime value is not printed by Crystal Reports so if the above formula is placed in a report, the formula field would be blank for order dates before 1995. Null Time values and null Date values behave similarly.

Here is an example that illustrates the use of parentheses to have more than one expression executed as the outcome of an If condition. A company charges a 5 percent fee for orders shipped within 3 days and a 2 percent fee otherwise. It wants to print messages such as "Rush shipping is $100.00" or "Regular shipping is $20.00" as appropriate.

```
Local StringVar message;
Local CurrencyVar ship;
If {Orders.Ship Date} - {Orders.Order Date} <= 3 Then
(
message := "Rush";
//A semicolon at the end of the next line
//is optional
ship := {Orders.Order Amount} * 0.05
) //A semicolon cannot be placed here
Else
(
message := "Regular";
ship := {Orders.Order Amount} * 0.02;
);
//The preceding semicolon is required to separate the
//If expression from the final expression below
message & " shipping is " & CStr (ship)
```

When expressions are grouped together with parentheses, the whole group is considered as a single expression, and its value and type are the value and type of the final expression inside the parentheses.

```
//The parentheses group expression as a whole has
//Currency type
(
//The first expression in the parentheses has
//String type
message := "Rush";
//The second and final expression in parentheses
//has Currency type
ship := {Orders.Order Amount} * 0.05;
)
```

Thus, for example, the following formula gives an error. The reason is that the Then part of the If expression returns a Currency value while the Else part returns a String value. This is not allowed, since the If expression is an expression and so must always return a value of a single type.

```
//An erroneous formula
Local StringVar message;
Local CurrencyVar ship;
If {Orders.Ship Date} - {Orders.Order Date} <= 3 Then
(
message := "Rush";
ship := {Orders.Order Amount} * 0.05
)
Else
(
//The following 2 lines were interchanged
ship := {Orders.Order Amount} * 0.02;
message := "Regular";
);
message & " shipping is " & CStr (ship)
```

One way of fixing up the erroneous formula without being careful about expression order is just to make the If expression return a constant value of the same type in every branch. For example, the If expression now returns the Number value 0:

```
//Repaired the erroneous formula
Local StringVar message;
Local CurrencyVar ship;
If {Orders.Ship Date} - {Orders.Order Date} <= 3 Then
(
message := "Rush";
ship := {Orders.Order Amount} * 0.05;
0
)
Else
(
ship := {Orders.Order Amount} * 0.02;
message := "Regular";
0
);
message & " shipping is " & CStr (ship)
```

Select expressions (Crystal syntax)

The Select expression is similar to an If expression. Sometimes however, you can write clearer and less repetitive formulas using the Select expression. For example, to evaluate the {Customer.Fax} field to determine if the area code is for Washington state (206, 360, 509) or British Columbia, Canada (604, 250):

```
//Select example 1
Select {Customer.Fax}[1 To 3]
Case "604", "250" :
"BC"
Case "206", "509", "360" :
"WA"
Default :
"";
```

The expression right after the Select keyword is called the *Select condition*. In the above example it is {Customer.Fax}[1 To 3]. The Select expression tries to find the first Case that matches the Select condition, and then executes the expression following the colon for that Case. The Default case is matched if none of the preceding cases match the Select condition. Notice that there is also a colon after the Default.

```
//Same effect as Select example 1
Local StringVar areaCode := {Customer.Fax}[1 To 3];
If areaCode In ["604", "250"] Then
"BC"
Else If areaCode In ["206", "509", "360"] Then
"WA"
Else
"";
```

Example

This formula groups the number of Oscar nominations a movie received into low, medium, high or extreme categories and in the process, shows some of the possibilities for the expression lists following the Case labels:

```
//Select example 2
Select {movie.NOM}
Case 1,2,3, Is < 1 :
(
//Can have expression lists by using
//parentheses
10 + 20;
"low"
)
Case 4 To 6, 7, 8, 9 :
"medium"
Case 10 :
"high"
Default :
"extreme"
```

The Default clause of the Select expression is optional. If the Default clause is missing and none of the cases are matched, then the Select expression returns the default value for its expression type. For example, if in the above example the Default clause were omitted and {movie.NOM} = 11, it would return the empty string "". The Select expression is an expression, and similar comments as in the *More details on If expressions* section apply to it as well.

For loops (Crystal syntax)

For loops enable you to evaluate a sequence of expressions multiple numbers of times. This is unlike the If and Select expressions where the program passes through each expression at most once during the formula's evaluation. For loops are best when you know the number of times that the expressions needs to be evaluated in advance.

The syntax of the For loop through examples

Example 1

Suppose you want to reverse the {Customer.Customer Name} string. For example, "City Cyclists" becomes "stsilcyC ytiC".

```
//Reverse a string version 1
Local StringVar str := "";
Local NumberVar strLen :=
Length ({Customer.Customer Name});
Local NumberVar i;
For i := 1 To strLen Do
(
Local NumberVar charPos := strLen - i + 1;
str := str + {Customer.Customer Name}[charPos]
);
str
```

Examine how this formula works assuming that the current value of the field {Customer.Customer Name} is "Clean Air". The variable strLen is assigned to be the length of "Clean Air", namely 9. The variable *i* is known as a *For counter variable* since its value changes with each iteration of the For loop. In other words, it is used to count the iterations of the loop. The For loop will iterate 9 times, during the first time, i is 1, then i is 2, then i is 3 and so on until finally i equals 9. During the first iteration, the ninth character of {Customer.Customer Name} is appended to the empty string variable str. Thus str equals "r" after the first iteration. During the second iteration, the eighth character of {Customer.Customer Name} is appended to str and so str equals "ri". This continues until after the ninth iteration, str equals, "riA naelC" which is the reversed string.

Example 2

Here is a simpler version of the above formula that uses a Step clause with a negative Step value of . For the "Clean Air" example, i is 9 for the first iteration, 8 for the second, 7 for the third and so on until it is 1 in the final iteration.

```
//Reverse a string version 2
Local StringVar str := "";
Local NumberVar strLen :=
Length ({Customer.Customer Name});
Local NumberVar i;
For i := strLen To 1 Step Do
(
str := str + {Customer.Customer Name}[i]
);
str
```

Example 3

The simplest version is to use the built in function StrReverse:

```
//Reverse a string version 3
StrReverse ({Customer.Customer Name})
```

The built in String functions in Crystal Reports 8.5 can handle many of the string processing applications that would traditionally be handled using a For loop or some other kind of loop. However, For loops provide the most flexibility in processing strings and also power in processing arrays, which can be essential if the built-in functions do not cover your intended application.

For loop example

Here is a more full featured example of Crystal Reports' string processing capabilities. The Caesar cipher is a simple code that is traditionally credited to Julius Caesar. In this code, each letter of a word is replaced by a letter five characters further in the alphabet. For example, "Jaws" becomes "Ofbx". Notice that "w" is replaced by "b"; since there are not 5 characters after "w" in the alphabet, it starts again from the beginning. Here is a formula that implements applying the Caesar cipher to the field {Customer.Customer Name} in the Xtreme database:

```
//The Caesar cipher

//The input string to encrypt
Local StringVar inString := {Customer.Customer Name};
Local NumberVar shift := 5;
Local StringVar outString := "";
Local NumberVar i;
For i := 1 To Length(inString) Do
(
Local StringVar inC := inString [i];
Local StringVar outC;
Local BooleanVar isChar :=
LowerCase(inC) In "a" To "z";
Local BooleanVar isUCaseChar :=
isChar And (UpperCase (inC) = inC);

inC := LCase(inC);

If isChar Then
(
Local NumberVar offset :=
(Asc(inC) + shift - Asc("a")) Mod
(Asc("z") - Asc("a") + 1);
outC := Chr(offset + Asc("a"));
If isUCaseChar Then outC := UpperCase(outC)
)
Else
outC := inC;
outString := outString + outC
);
outString
```

In the above example there is an If expression nested within the expression block of the For loop. This If expression is responsible for the precise details of shifting a single character. For example, letters are treated differently from punctuation and spaces. In particular, punctuation and spaces are not encoded. The general points here are that control structures can be nested within other control structures and that multiple expressions can be included in the (parentheses enclosed) expression blocks of other control structures.

Using Exit For (Crystal syntax)

You can exit from a For loop by using Exit For. The following example searches the Global array names for the name "Fred". If it finds the name, it returns the index of the name in the array. Otherwise it returns .

For example, if the names array is:

```
["Frank", "Helen", "Fred", "Linda"]
```

Then the formula returns 3.

```
Global StringVar Array names;
```

```
//The names array has been initialized and filled
//in other formulas
Local NumberVar i;
Local NumberVar result := ;
//The UBound function returns the size of its array
//argument
For i := 1 to UBound (names) Do
(
If names [i] = "Fred" Then
(
result := i;
Exit For
)
);
result
```

When considered as an expression, the For loop always returns the Boolean value True. Thus you will almost never want a For loop to be the last expression in a formula, since then the formula will then just display the value True rather than your intended result.

While Loops (Crystal syntax)

Another looping mechanism is the While loop. A While loop can be used to execute a fixed block of statement an indefinite amount of time.

The 2 different types of While loops

Type of Do Loop	Explanation	Example
While ... Do	The While ... Do loop evaluates the condition, and if the condition is true, then it evaluates the expression following the Do.	While *condition* Do *expression*
	When it has finished doing this, it evaluates the condition again and if the condition is true, it evaluates the expression following the Do again. It continues repeating this process until the condition is false.	
Do ... While	The Do ... While loop evaluates the expression once no matter what.	Do *expression* While *condition*
	It then evaluates the condition, and if the condition is true, evaluates the expression again. This process continues until the condition is false.	

Note:

- The While loops support an Exit While statement to immediately jump out of the loop. Its use is analogous to the use of Exit For in For loops.
- As with the For loop, the While loop when considered as an expression always returns the Boolean value True.

While ... Do loop example (Crystal syntax)

The following example searches for the first occurrence of a digit in an input string. If a digit is found, it returns its position, otherwise it returns . In this case, the input string is set explicitly to a string constant, but it could be set equal to a String type database field instead. For example, for the input String, "The 7 Dwarves", the formula returns 5, which is the position of the digit 7.

```
Local StringVar inString := "The 7 Dwarves";
Local NumberVar strLen := Length (inString);
Local NumberVar result := ;
Local NumberVar i := 1;
While i <= strLen And result = Do
(
Local StringVar c := inString [i];
If NumericText (c) Then
result := i;
i := i + 1;
);
result
```

Safety mechanism for loops (Crystal syntax)

There is a safety mechanism to prevent report processing from hanging due to an infinite loop. Any one evaluation of a formula can have at most 30,000 loop condition evaluations per formula evaluation. This will be explained by the example below.

For example:

```
Local NumberVar i := 1;
While i <= 100000 Do
(
If i > {movie.STARS} Then
Exit While;
i := i + 1
);
20
```

If {movie.STARS} is greater than 30,000 then the loop condition (i <= 100000) will be evaluated more than the maximum number of times and an error message is displayed. Otherwise the loop is OK.

Note: The safety mechanism applies on a per formula base, not for each individual loop. For example:

```
Local NumberVar i := 1;
For i := 1 To 10000 Do
(
Sin (i);
);
While i <= 25000 Do
(
i := i + 1;
)
```

The above formula also triggers the safety mechanism since the 30,000 refers to the total number of loop condition evaluations in the formula and this formula will have 10001 + 25001 such evaluations.

Limitations (Crystal syntax)

For reference purposes, here are the sizing limitations of the formula language:

- The maximum length of a String constant, a String value held by a String variable, a String value returned by a function or a String element of a String array is 254 bytes.
- The maximum size of an array is 1000 elements.
- The maximum number of arguments to a function is 1000. (This applies to functions that can have an indefinite number of arguments such as Choose).
- The maximum length of the text of a formula is 64K.
- The maximum number of loop condition evaluations per evaluation of a formula is 30,000.
- See Safety mechanism for loops (Crystal syntax).
- Date-time functions modeled on Visual Basic accept dates from year 100 to year 9999. Traditional Crystal Reports functions accept dates from year 1 to year 9999.

Functions

Functions are built-in procedures or subroutines used to evaluate, make calculations on, or transform data.

When you specify a function, the program performs the set of operations built into the function without you having to specify each operation separately. In this way, a function is a kind of shorthand that makes it easier and less time consuming for you to create reports.

Simple, straightforward examples are provided for each function. In addition, for many functions, references are provided to specific formulas. These formulas demonstrate more advanced examples of solving real-world reporting problems.

Click the appropriate link to jump to that section:

- Mathematical functions
- Summary functions
- Financial functions
- String functions
- Date/Time functions
- Date Range functions
- Arrays
- Type Conversion functions
- Programming shortcut functions
- Evaluation Time functions
- Print State functions
- Document properties functions
- Alerts functions
- Additional Functions
- Functions to duplicate group fields
- Grand total functions
- Conditional formatting functions
- Business Calendar.

Mathematical functions

Mathematical functions are used for a variety of mathematics-oriented calculations and operations.

Abs (x)

Sgn (number)

Int (number)

Round (x), Round (x, #places)

Truncate (x), Truncate (x, #places)

Abs (x)

Basic and Crystal syntax.

Arguments

x is the Number for which you want the Absolute Value returned.

Returns

Absolute value of x.

Action

Abs returns the value of x, ignoring any negative values.

Examples

`Abs(1.50)`

Returns 1.50.

`Abs(.50)`

Returns 1.50.

`Abs(10 - 7)`

Returns 3.

`Abs(7 - 10)`

Returns 3.

Rem Basic syntax

If Abs(37 - {file.FIELD}) > 1 Then

formula = "Maintenance, Temperature Check"

End If

//Crystal syntax

If Abs(37 - {file.FIELD}) > 1 Then

"Maintenance, Temperature Check"

Else

"" ;

This flags instances where a laboratory heat block has a temperature variation greater than +/- 1 degree C.

Comments

This function is designed to work like the Visual Basic function of the same name.

Related topics

Sgn (number)

Basic and Crystal syntax.

Argument

Number value

Returns

Returns 1, 0 or

Action

Sgn returns the sign of a given number: 1 if number > 0, 0 if number is 0, and if number < 0.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
Sgn (0)
```

Returns .

```
Sgn (0)
```

Returns 0.

```
Sgn (10)
```

Returns 1.

Comment

This function is designed to work like the Visual Basic function of the same name.

Int (number)

Basic and Crystal syntax.

Argument

Number value

Returns

Whole number value

Action

Int returns the integer portion of a given number by rounding it down to the next smallest integer.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
Int (123.678)
```

Returns 123.

```
Int (23.678)
```

Returns 24.

```
Int (23.1)
```

Returns 24.

Comments

- This function is designed to work like the Visual Basic function of the same name.
- Fix (n) (or Truncate (n)) and Int (n) are the same except when n is negative, in which case Fix and Truncate return the first integer greater than or equal to n, and Int returns the first integer smaller than or equal to n. For example,

```
Fix (0.2)
```

```
Truncate (0.2)
```

Both return 0.

```
Int (0.2)
```

Returns 1.

Related topics

Fix

Truncate (x), Truncate (x, #places)

Round (x), Round (x, #places)

Basic and Crystal syntax.

Arguments

- x is a fractional value to be rounded off.
- #places is a whole number indicating the number of decimal places x is to be rounded off to.

Returns

Fractional Number

Action

Round rounds to the nearest whole number if the #places argument is excluded. If the value to the right of the decimal point is .499 or below, the program rounds to the next lowest number. If the value to the right of the decimal point is .5 or above, it rounds to the next highest number.

If the #places argument is used, the value in x is rounded to the nearest decimal place indicated by #places. Specifying #places as 0 works identical to leaving the #places argument off. You can also specify a negative number for #places; the number will be rounded to the nearest tenth (.1), hundredth (.01), or thousandth (.001), and so on.

Typical uses

Use Round any time rounding off a value to a particular decimal place is more appropriate than using the original value.

Examples

The following examples are applicable to both Basic and Crystal syntax, although Fix is preferred in the latter:

```
Round(1.23456)
```

Returns 1.

Round(1.499)

Returns 1.

```
Round(1.5000)
```

Returns 2.

```
Round(2345.23456,4)
```

Returns 2345.2346.

```
Round(2345.23456,3)
```

Returns 2345.235.

```
Round(2345.23456,2)
```

Returns 2345.23.

```
Round(2345.23456,0)
```

Returns 2345.

```
Round(2345.23456,)
```

Returns 2350.

```
Round(2345.23456,)
```

Returns 2300.

```
Round(2345.23456,)
```

Returns 2000.

```
Round(1.234499,3)
```

Returns 1.234.

```
Round(1.234500,3)
```

Returns 1.235.

```
Round({file.AMOUNT},1)
```

Returns 1854.5 where Amount = 1854.49.

```
Round({file.AMOUNT})
```

Returns 1854.00 where Amount = 1854.49.

```
Round({file.AMOUNT})
```

Returns 1855.00 where Amount = 1854.51.

```
Round({file.WEIGHT} / 100)
```

Returns 4 where Weight = 424.

```
Round({file.WEIGHT} / 100)
```

Returns 5 where Weight = 451.

```
Round((A * B) / C)
```

Returns 11 where A = 25, B = 3, and C = 7.

```
Round(file.AMOUNT,1)
```

Returns 1854.5 where Amount = 1854.51.

```
Round({file.WAGE} * {file.HOURS WORKED}, 2)
```

Returns $146.63 where Wage = $5.75 and Hours worked = 25.5.

Comments

Rounding is also a feature available as a formatting option for numerical values in fields.

Be aware that using field formatting features may affect how values are used in formulas. See ToNumber (numeric), ToNumber (string), ToNumber (Boolean) and the To currency operator.

Related topics

Truncate (x), Truncate (x, #places)

Fix

Int (number)

Formula 8

Truncate (x), Truncate (x, #places)

Truncate and Fix are equivalent functions. However, Truncate is preferred in Crystal syntax whereas Fix is preferred in Basic syntax.

Arguments

- x is a fractional Number to be truncated.
- #places is a whole number indicating how many decimal places are to remain after the value is truncated. (This argument is optional.)

Returns

Fractional Number

Action

Truncate(x) returns a whole number by truncating the number at the decimal point. If the #places argument is specified, the number is truncated to the decimal place indicated and the function returns a fractional Number. If #places is negative, the number is rounded to the first ten, hundred, and so on.

Typical uses

Use this function whenever the characters to the right of the decimal are not needed for a report or calculation.

Examples

The following examples are applicable to both Basic and Crystal syntax, although Fix is preferred in the latter:

```
Truncate(1.23456)
```

Returns 1.

```
Truncate(1.499)
```

Returns 1.

```
Truncate(1.599)
```

Returns 1.

```
Truncate(1.999)
```

Returns 1.

```
Truncate(12346.33, 1)
```

Returns 12346.3.

```
Truncate(12345.33, )
```

Returns 12300.00.

If you have 147 golf balls in stock and want to know how many dozen are available for sale, your calculation is 147/12=12.25 12.25 (truncated) = 12 dozen available for sale. If you sell balls only by the dozen, the .25 dozen you truncated is unimportant.

```
Truncate({file.BALL INVENTORY} / 12)
```

Returns 12 where Ball inventory = 147 (147 / 12 = 12.25, 12.25 truncated = 12).

```
Truncate({file.BALL INVENTORY} / 12)
```

Returns 12 where Ball inventory = 155 (155/12 = 12.92, 12.92 truncated = 12).

```
Truncate({file.BALL INVENTORY} / 12)
```

Returns 13 where Ball inventory = 157 (157/12 = 13.08, 13.08 truncated = 13).

Comments

- This is not a rounding function; Truncate simply deletes all characters to the right of the decimal point.

  See the Round (x), Round (x, #places) for an explanation on rounding.
- Truncate (n) and Int (number) are synonymous except when n (number) is negative, in which case Truncate returns the first integer greater than or equal to n, and Int returns the first integer smaller than or equal to n. For example,

```
Truncate (0.2)
```

Returns 0.

```
Int (0.2)
```

Returns 1.

- Truncating is also a feature available as a formatting option for numerical values in fields. Be aware that using field formatting features may affect how values are used in formulas. For more information on converting to Numbers and Currency, see the ToNumber (numeric), ToNumber (string), ToNumber (Boolean) and To currency operator.

Related topics

Int (number)

Formula 13

Fix

Fix and Truncate (x), Truncate (x, #places) are equivalent functions. However, Fix is preferred in Basic syntax whereas Truncate is preferred in Crystal syntax.

Overloads

- Fix (number)
- Fix (number, #places)

Arguments

- number is the Number value to be truncated; it can be positive, 0 or negative.
- #places is an optional Number indicating the number of decimal places to be truncated to. If omitted, 0 is assumed.

Returns

Whole number value which can be positive, 0 or negative.

Action

Fix truncates a number to the specified number of decimal places and returns it. If #places is omitted, 0 is assumed.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
Fix (123.678)
```

Returns 123.

```
Fix (23.678)
```

Returns 23.

```
Fix (123.678,1)
```

Returns 123.6.

```
Fix (123.678,2)
```

Returns 123.67.

Comments

- This function with a single parameter, Fix (number), is designed to work like the Visual Basic function of the same name.
- Fix (n) and Int (number) work the same except when n is negative, in which case Fix returns the first integer greater than or equal to n, and Int returns the first integer smaller than or equal to n. For example,

```
Fix (0.2)
```

Returns 0.

```
Int (0.2)
```

Returns 1.

Remainder (num, denom)

Basic and Crystal syntax.

Arguments

- Numerator is a fractional value.
- Denominator is a fractional value.

Returns

Fractional Value

Action

Remainder returns the remainder after the numerator (dividend) has been divided by the denominator (divisor). In a typical division situation, the program expresses a quotient as a whole number (if any) and up to six decimal places. When using Remainder, however, the program performs the division internally, determines the whole number quotient and the remainder, and returns only the remainder.

Typical uses

You can use this function in making conversions (feet to miles, units to grosses, etc.) You can also use it to select every nth item out of an array.

Examples

The following examples are applicable to Basic and Crystal syntax:

```
Remainder(12,5)
```

Returns 2.

```
Remainder(16,5)
```

Returns 1.

Rem Basic syntax

If Remainder ({file.EXAM#}, 7) = 0 Then

formula = "******"

End If

//Crystal syntax

If Remainder({file.EXAM#}, 7) = 0 Then

"******"

Else

""

This flags every 7th exam for grading by a second party.

```
ToText(Truncate({file.DAYS}/7)) + " week(s), " + ToText(Remainder
({file.DAYS},7)) + " day(s)"
```

Returns "9 week(s), 1 day(s)". Converts days to weeks and days. For example, if the field had a value of 64 days, the formula would return the value of "9 week(s), 1 day(s)."

Related topics

Modulus (x Mod y)

Formula 13

Formula 17

Sin (number)

Basic and Crystal syntax.

Argument
Number value that represents an angle in radians

Returns
Number value between and 1

Action
Sin returns a number specifying the sine of an angle given in radians. It takes a right-angle triangle, and returns the length of the side opposite to the specified angle divided by the length of the hypotenuse.

Examples
The following examples are applicable to both Basic and Crystal syntax:

```
Sin (1)
```

Returns 0.8415 (rounded to 4 decimals). This is the sine of 1 radian.

```
Sin (30 * crPi / 180)
```

Returns 0.5. This is the sine of 30 degrees. Before taking the sine, the angle is converted to radians by multiplying by crPi / 180.

Comments

- This function is designed to work like the Visual Basic function of the same name.
- The range of values returned by Sin is between and 1.

Cos (number)

Basic and Crystal syntax.

Argument
Number value which is an angle in number of radians

Returns
Number value between and 1

Action
Cos returns a Number specifying the cosine of an angle given in radians. It takes a right-angle triangle, and returns the length of the side adjacent to the specified angle divided by the length of the hypotenuse.

Examples
The following examples are applicable to both Basic and Crystal syntax:

```
Cos (1)
```

Returns 0.5403 (rounded to 4 decimals). This is the cosine of 1 radian.

```
Cos (60 * crPi / 180)
```

Returns 0.5. This is the cosine of 60 degrees. Before taking the cosine, the angle is converted to radians by multiplying by crPi / 180.

Comments

- This function is designed to work like the Visual Basic function of the same name.
- The range of values returned by Cos is between and 1.

Tan (number)

Basic and Crystal syntax.

Argument
Number value that represents an angle in radians.

Returns
Number value

Action
Tan returns a Number specifying the tangent of an angle given in radians. It takes a right-angle triangle, and returns the length of the side opposite the specified angle divided by the length of the side adjacent to the angle.

Examples
The following examples are applicable to both Basic and Crystal syntax:

```
Tan (1)
```

Returns 1.5574 (rounded to 4 decimals). This is the tangent of 1 radian.

```
Tan (45 * crPi / 180)
```

Returns 1. This is the tangent of 45 degrees. Before taking the tangent, the angle is converted to radians by multiplying by crPi / 180.

Comment
This function is designed to work like the Visual Basic function of the same name.

Atn (number)

Basic and Crystal syntax.

Argument
Number value

Returns
Number value which is an angle specified in number of radians

Action

Atn returns a Number specifying the arctangent of the given Number argument. In other words, it returns the angle whose tangent is the given Number argument.

Example

The following example is applicable to both Basic and Crystal syntax:

```
Atn (1)
```

Returns an angle of 0.7854 radians (rounded to 4 radians). To convert this angle to degrees, multiply by 180 / crPi. For instance, Atn (1) * 180 / crPi is 45 degrees.

Comments

- This function is designed to work like the Visual Basic function of the same name.
- The range of values returned by Atn is between i/2 and pi/2 radians.

Pi

Basic and Crystal syntax.

Returns the Mathematical value *pi*, which is 3.14 (if rounded to 2 decimal places).

Sqr (number)

Basic and Crystal syntax.

Argument

Number value greater than or equal to 0.

Returns

Number value

Action

Sqr returns the square root of a given number.

Example

The following examples is applicable to both Basic and Crystal syntax:

```
Sqr (100)
```

Returns 10.

Comment

This function is designed to work like the Visual Basic function of the same name.

Exp (number)

Basic and Crystal syntax.

Argument

Number value that specifies a power.

Returns
Number value.

Action
Exp returns a Number specifying *e* (the base of natural logarithms) raised to a power. The value of *e* is approximately 2.718282.

Example
The following example is applicable to both Basic and Crystal syntax:

```
Exp (1.5)
```

Returns the exponentiation of 1.5, which is approximately 4.48169.

Comments

- This function is designed to work like the Visual Basic function of the same name.
- Numeric overflow occurs if the given number argument is larger than approximately 705.

Log (number)

Basic and Crystal syntax.

Argument
Number value

Returns
Number value

Action
Log returns a Number specifying the natural logarithm of a given number. The natural logarithm is the logarithm to the base *e*, where *e* is approximately 2.718282.

Example
The following examples is applicable to both Basic and Crystal syntax:

```
Log (1.5)
```

Returns 0.4055 (rounded to 4 decimals).

Comments
This function is designed to work like the Visual Basic function of the same name.

Rnd

Basic and Crystal syntax.

Overloads

- Rnd ()
- Rnd (seed)

Argument
seed is an optional Number value argument.

Returns
A Number value.

Action
Rnd returns a random number greater than or equal to 0 and less than 1. If seed is equal to 0, Rnd returns the random number that was returned from the previous call to Rnd. If seed is not supplied *or is greater than 0*, then Rnd returns the next random number in the internally generated sequence of random numbers. If seed *is less than 0*, then Rnd uses this value of seed to start a new random number sequence and returns the first value in this sequence.

Typical uses
Used when your formula requires a randomly generated number such as for statistical calculations or for selecting records at random to limit the data in a report.

Comments

- This function is similar to the Visual Basic function of the same name.
- You can call Rnd without ever starting a new random number sequence by specifying a negative seed argument. If you do so, an internal seed will be generated using the system clock.
- The reason to start a new random number sequence by calling Rnd with a negative seed argument, and then making subsequent calls to Rnd without an argument (or with a positive argument) is so that the report will look exactly the same every time it is previewed. In other words, it allows you to make use of random numbers, but get a reproducible result.

Summary functions

The Summary functions are all used to summarize field data in several different ways. Because of their unique summarization features, they have been grouped together as summary functions. The function name SummaryFunction takes the place of an actual function name.

Summary functions can be designed to perform operations on group data.

Sum (fld), Sum (fld, fld), Sum (fld, condFld), Sum (fld, condFld, cond), Sum (x)

SummaryFunction (fld)

SummaryFunction (fld, condFld)

SummaryFunction (fld, condFld, cond)

Average (fld), Average (fld, condFld), Average (fld, condFld, cond), Average (x)

StdDev (fld), StdDev (fld, condFld), StdDev (fld, condFld, cond), StdDev (x)

PopulationStdDev (fld), PopulationStdDev (fld, condFld), PopulationStdDev (fld, condFld, cond), PopulationStdDev (x)

Variance (fld), Variance (fld, condFld), Variance (fld, condFld, cond), Variance (x)

Conditions for summary functions

Here is a list of the conditions for each summary function:

Boolean conditions	• any change • change to Yes • change to No • every Yes • every No • next is Yes • next is No

Date conditions	• daily • weekly • biweekly • semimonthly • monthly • quarterly • semiannually • annually
Time conditions	• by second • by minute • by hour • by AMPM

Sum (fld), Sum (fld, fld), Sum (fld, condFld), Sum (fld, condFld, cond), Sum (x)

Basic and Crystal syntax.

Arguments

- fld is any valid database or formula field that can be evaluated by the function.
- condFld is a field used to group the values in fld by.
- cond is a String indicating the type of grouping for condFld. You only specify this argument when condFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.
- x is an array of values that can be evaluated by the function being used.

Returns

Fractional Number

Action

Enables you to add the values that appear in your report. For example:

- If a sales report includes a field that shows the amount of each order, you can compute the sum of all the orders that appear on the report (a grand total sum). For information on this kind of summary, see SummaryFunction (fld).
- If you break orders into groups (for example, orders grouped by the state that they come from), you can compute the sum of the order amounts per group (in this case, per state). For information on this kind of summary, see SummaryFunction (fld, condFld).
- If you break orders into date or Boolean groups (for example, orders grouped by the month in which they were placed), you can compute the sum of the order amounts per group based on a specific change in the date or Boolean field (in this case, per month). For information on this kind of summary, see SummaryFunction (fld, condFld, cond).

- If you specify a set of individual values, you can also compute the sum of the values in the set. For information on this kind of summary, see Array summary functions (x).

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
Sum({file.QTY})
```

Calculates the sum of all values in the QTY field.

```
Sum({orders.ORDER ID}, {orders.CUSTOMER ID})
```

Sums (totals) the orders in each group of orders in the Amount field. The orders are separated into groups whenever the value in the Customer ID field changes.

```
Sum({orders.AMOUNT}, {orders.DATE}, "monthly") % Sum({orders.AMOUNT
})
```

Groups values in the Amount field by month, and calculates the sum of the values for each month group as a percentage of the sum of the values for the entire report.

```
Sum([{file.AMOUNT}, {file.PRICE}, {file.COST}])
```

Sum of values in the Amount, Price, and Cost fields.

Note: Using this function in a formula forces the formula to be evaluated at print time. For more information on evaluation time considerations, see the Evaluation Time functions.

SummaryFunction (fld)

Basic and Crystal syntax.

Arguments

fld is any valid database or formula field that can be evaluated by the function.

Returns

Fractional Number.

Action

SummaryFunction (fld) summarizes the values in the specified field for the entire report. It generates, in effect, a "grand total" summary.

Typical uses

Use these functions any time you need to summarize all the values that appear in a given field in a report, or when you need to use a summary in a calculation or comparison.

Examples

The following examples are applicable to Crystal syntax:

```
SummaryFunction({orders.ORDER AMOUNT})
```

Summarizes all values in the Amount field.

If {orders.ORDER AMOUNT} > SummaryFunction({orders.ORDER AMOUNT}) Then

"Flag"

Else

""

Compares each value in the Amount field to the summary value of all values in the Amount field. If the value is greater than the summary value, it flags it with the word "Flag"; if it is equal to or less then the summary value, it prints nothing.

```
Average({orders.ORDER AMOUNT})
```

Calculates the average of all values in the Amount field.

If Count({orders.ORDER ID}) >= 100 Then

"Congratulations on meeting your quota!"

Else

""

Prints the congratulatory message if the number of orders is 100 or more, and prints nothing if the number of orders is less than 100.

```
Sum({file.QTY})
```

Calculates the sum of all values in the QTY field.

Comments

Not all arithmetic group functions are available for use with all data types.

SummaryFunction (fld, condFld)

Basic and Crystal syntax.

Arguments

- fld is any valid database or formula field that can be evaluated by the function.
- condFld is any valid database or formula field used to group the values in fld.

Returns

Fractional Number

Action

SummaryFunction (fld, condFld) summarizes each group of values that is generated when the specified summary condition is met.

Typical uses

Use these functions whenever you want to duplicate, in a formula, a summary:

- that summarizes the values in a group
- that uses a String, Number, Date, Time, DateTime, or Currency field as the sort and group by (trigger) field.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
SummaryFunction({orders.ORDER AMOUNT}, {customer.CUSTOMER ID})
```

Summarizes the orders in each group of customer orders. The orders are separated into groups whenever the value in the Customer field changes.

```
Count({orders.AMOUNT}, {orders.CUSTOMER ID})
```

This formula counts the number of orders in each group of orders in the Amount field (the total orders for each month). The orders are separated into groups whenever the value in the Customer ID field changes.

```
StdDev({orders.AMOUNT}, {orders.CUSTOMER ID})
```

Calculates the standard deviation for the orders in each group of orders in the Orders field. The orders are separated into groups whenever the value in the Customer field changes.

```
Sum({orders.ORDER ID}, {orders.CUSTOMER ID})
```

Sums (totals) the orders in each group of orders in the Amount field. The orders are separated into groups whenever the value in the Customer ID field changes.

The following example is applicable to Crystal syntax:

```
SummaryFunction({orders.ORDER AMOUNT}, {customer.REGION}) %
SummaryFunction({orders.ORDER AMOUNT})
```

Groups values in the Amount field by region, summarizes the values for each region group, and shows those values as a percentage of the summary value for the entire report.

Comments

In order to use this function to insert a group field in a formula, you must have already entered a group field in your report with identical parameters: same field, same sort and group by (trigger) field, same action (average, count, etc.).

SummaryFunction (fld, condFld, cond)

Basic and Crystal syntax.

Arguments

- fld is any valid database or formula field that can be evaluated by the function.
- condFld is any valid date or Boolean field used to group the values in fld by.
- cond is a text string indicating the condition of condFld that controls grouping. Valid strings for this argument depend on whether condFld contains date or Boolean values. For information about the available conditions, see Conditions for summary functions.

Returns

Fractional Number

Action

SummaryFunction (fld, condFld, cond) summarizes each group of values that is generated when the specified summary condition is met. These functions work just like SummaryFunction (fld, condFld), but, because they use a date or Boolean field as a sort and group by (trigger) field, they require a condition in addition to the other arguments.

Typical uses

Use these functions whenever you want to duplicate, in a formula, a summary:

- that summarizes the values in a group
- that uses a date or Boolean field as the sort and group by field.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
SummaryFunction({orders.ORDER ID}, {orders.ORDER DATE}, "monthly")
```

Summarizes the orders in each group of orders in the Orders field. The orders are separated into groups whenever the value in the Date field changes to a new month.

```
Count({orders.ORDER AMOUNT}, {orders.ORDER DATE}, "monthly")
```

Counts the number of orders in each group of orders in the Amount field (the total orders for each month). The orders are separated into groups whenever the value in the Date field changes to a new month.

```
DistinctCount({orders.CUSTOMER ID}, {orders.ORDER DATE}, "weekly")
```

Counts the number of different customers contacted, follow up contacts are excluded. Customers are separated into groups whenever the value in the Date field changes to a new week.

The following example is applicable to Crystal syntax:

```
(SummaryFunction({orders.ORDER AMOUNT}, {orders.ORDER AMOUNT},
"monthly")) % (SummaryFunction({order.ORDER AMOUNT}))
```

Groups values in the Amount field by month, summarizes the values for each month group, and shows those values as a percentage of the summary value for the entire report.

Comments

In order to use these functions to insert a group field in a formula, you must have already entered a group field in your report with identical parameters: same field, same sort and group by field, same date or Boolean condition, same action (average, count, etc.).

Average (fld), Average (fld, condFld), Average (fld, condFld, cond), Average (x)

Basic and Crystal syntax.

Arguments

- fld is any valid database or formula field that can be evaluated by the function.
- condFld is a field used to group the values in fld by.
- cond is a String indicating the type of grouping for condFld. You only specify this argument when condFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.
- x is an array of values that can be evaluated by the function being used.

Returns

Fractional Number

Action

Enables you to average the values that appear in your report. For example:

- If a sales report includes a field that shows the amount of each order, you can compute the average of all the orders that appear on the report (a grand total average). For information on this kind of summary, see SummaryFunction (fld).
- If you break orders into groups (for example, orders grouped by the state that they come from), you can compute the average order per group (in this case, per state). For information on this kind of summary, see SummaryFunction (fld, condFld).
- If you break orders into date or Boolean groups (for example, orders grouped by the month in which they were placed), you can compute the average order per group based on a particular change in the date or Boolean field (in this case, per month). For information on this kind of summary, see SummaryFunction (fld, condFld, cond).
- If you specify an array of individual values, you can also compute the average value in the set. For information on this kind of summary, see Array summary functions (x).

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
Average({orders.ORDER AMOUNT})
```

Calculates the average of all values in the Amount field.

```
Average ({orders.ORDER AMOUNT}, {customer.CUSTOMER ID})
```

Finds the average amount of a sale made to each customer.

The following example is applicable to Crystal syntax:

```
Average ([10,12,32,48])
```

Calculates the average of the values in an array of constants.

Note: Using this function in a formula forces the formula to be evaluated at print time. For more information on evaluation time considerations, see Evaluation Time functions.

StdDev (fld), StdDev (fld, condFld), StdDev (fld, condFld, cond), StdDev (x)

Basic and Crystal syntax.

Arguments

- fld is any valid database or formula field that can be evaluated by the function.
- condFld is a field used to group the values in fld by.
- cond is a String indicating the type of grouping for condFld. You only specify this argument when condFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.
- x is an array of values that can be evaluated by the function being used.

Returns

Fractional Number

Action

Enables you to find the standard deviation of a set of values in your report. For example:

- You can calculate the grand total standard deviation for all values in a field. For information about this kind of summary, see SummaryFunction (fld).
- You can calculate the standard deviation for all values within a group (for example, sales grouped by the state that they come from). For information on this kind of summary, see SummaryFunction (fld, condFld).
- You can calculate the standard deviation for all values within a group in which grouping is controlled by changes in a date or Boolean field (for example, sales grouped by the month in which they were made). For information on this kind of summary, see SummaryFunction (fld, condFld, cond).
- If you specify a set of individual values, you can compute the standard deviation of the values in the set. For information on this kind of summary, see Array summary functions (x).

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
StdDev({file.RESULTS})
```

Calculates the standard deviation of all values in the Result field.

```
StdDev({orders.AMOUNT}, {orders.CUSTOMER ID})
```

Calculates the standard deviation for the orders in each group of orders in the Orders field. The orders are separated into groups whenever the value in the Customer field changes.

```
StdDev({file.RESULTS}, {file.DATE}, "daily")
```

Calculates the variation for each group of laboratory test results in the Results field. The test results are separated into groups whenever the value in the Date field changes to a new day.

```
StdDev({file.SCORES}, {file.NATURALIZED}, "any change")
```

Groups values in the Scores field based on whether or not the test taker is a naturalized citizen, and calculates the standard deviation for each group of scores.

The following example is applicable to Crystal syntax:

```
StdDev([({file.QTY1} * {file.PRICE1}),({file.QTY2} * {file.PRICE2}),
({file.QTY3} * {file.PRICE3}), ({file.QTY4} * {file.PRICE4})])
```

Returns 36.60 where Qty1 = 2, Price1 = 10.00, Qty2 = 2, Price2 = 2.00, Qty3 = 10, Price3 = 3.00, and Qty4 = 8, Price4 = 11.00.

Comments

Standard deviation is calculated using the following technique:

- It calculates the average (mean) value for the items in the sample.
- It subtracts the average value from the value of each item.

- It squares the difference for each item.
- It adds the squared differences for all of the items in the sample.
- It divides the sum by one less than the number of items in the sample (N - 1). The result is the variance. Compare this step to the PopulationStdDev (fld), PopulationStdDev (fld, condFld), PopulationStdDev (fld, condFld, cond), PopulationStdDev (x).
- It calculates the square root of the variance to arrive at the standard deviation.

Note: Using this function in a formula forces the formula to be evaluated at print time.

For more information on evaluation time considerations, see Evaluation Time functions.

PopulationStdDev (fld), PopulationStdDev (fld, condFld), PopulationStdDev (fld, condFld, cond), PopulationStdDev (x)

Basic and Crystal syntax.

Arguments

- fld is any valid database or formula field that can be evaluated by the function.
- condFld is a field used to group the values in fld by.
- cond is a String indicating the type of grouping for condFld. You only specify this argument when condFld is a Date, Time, DateTime or Boolean field.

  For more information on the valid strings for this argument, see Conditions for summary functions.

- x is an array of values that can be evaluated by the function being used.

Returns

Fractional Number

Action

Enables you to find the population standard deviation of a set of values in your report. For example:

- You can calculate the grand total population standard deviation for all values in a field. For information on this kind of summary, see SummaryFunction (fld).
- You can calculate the population standard deviation for all values within a group (for example, sales grouped by the state that they come from). For information on this kind of summary, see SummaryFunction (fld, condFld).
- You can calculate the population standard deviation for all values within a group in which grouping is controlled by changes in a date or Boolean field (for example, sales grouped by the month in which they were made). For information on this kind of summary, see SummaryFunction (fld, condFld, cond).
- If you specify a set of individual values, you can compute the population standard deviation of the values in the set. For information on this kind of summary, see Array summary functions (x).

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
PopulationStdDev({file.SCORES})
```

Calculates the population standard deviation of all values in the Scores field for the entire report.

```
PopulationStdDev({orders. AMOUNT}, {orders.CUSTOMER ID})
```

Calculates the population standard deviation for the orders in each group of orders in the Amount field. The orders are separated into groups whenever the value in the Customer ID field changes.

```
PopulationStdDev({file.RESULTS}, {file.DATE}, "daily")
```

Calculates the variation for each group of laboratory test results in the Results field. The test results are separated into groups whenever the value in the Date field changes to a new day.

The following example is applicable to Crystal syntax:

```
PopulationStdDev([{file.QTY1}, {file.QTY2}, {file.QTY3},
{file.QTY4}])
```

Returns 3.57 where Qty1 = 2, Qty2 = 2, Qty3 = 10, and Qty4 = 8.

Comments

Population standard deviation is calculated using the following technique:

- It calculates the average (mean) value for the items in the population.
- It subtracts the average value from the value of each item.
- It squares the difference for each item.
- It adds the squared differences for all of the items in the population.
- It divides the sum by the number of items in the population (N). The result is the population variance. Compare this step to the StdDev (fld), StdDev (fld, condFld), StdDev (fld, condFld, cond), StdDev (x).
- It calculates the square root of the population variance to arrive at the population standard deviation.

Note: Using this function in a formula forces the formula to be evaluated at print time.

For more information on evaluation time considerations, see the Evaluation Time functions.

Variance (fld), Variance (fld, condFld), Variance (fld, condFld, cond), Variance (x)

Basic and Crystal syntax.

Arguments

- fld is any valid database or formula field that can be evaluated by the function.
- condFld is a field used to group the values in fld by.

- cond is a String indicating the type of grouping for condFld. You only specify this argument when condFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.
- x is an array of values that can be evaluated by the function being used.

Returns

Fractional Number

Action

Enables you to find the variance of a set of values in your report. For example:

- You can calculate the grand total variance for all values in a field. For information about this kind of summary, see SummaryFunction (fld).
- You can calculate the variance for all values within a group (for example, sales grouped by the state that they come from). For information on this kind of summary, see SummaryFunction (fld, condFld).
- You can calculate the variance for all values within a group in which grouping is controlled by changes in a date or Boolean field (for example, sales grouped by the month in which they were made). For information on this kind of summary, see SummaryFunction (fld, condFld, cond).
- If you specify a set of individual values, you can compute the variance of the values in the set. For information on this kind of summary, see Array summary functions (x).

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
Variance({file.AMOUNT})
```

Calculates the variance of all values in the Amount field.

```
Variance({file.RESULTS}, {file.DATE},"daily")
```

Calculates the variation for each group of laboratory test results in the Results field. The test results are separated into groups whenever the value in the Date field changes to a new day.

The following examples are applicable to Crystal syntax:

```
Variance({file.SCORES}, {file.TEACHERS}) % Variance({file.TEACHERS})
```

Groups values in the Scores field by teacher, and calculates the variance for each teacher group as a percentage of the variance of the values for the entire report (for all teachers listed).

```
Variance([{file.QTY1}, {file.QTY2}, {file.QTY3}, {file.QTY4}])
```

Returns 17.00 where Qty1 = 2, Qty2 = 2, Qty3 = 10, and Qty4 = 8.

Comments

Variance is calculated using the following technique:

- It calculates the average (mean) value for the items in the sample.
- It subtracts the average value from the value of each item.
- It squares the difference for each item.
- It adds the squared differences for all of the items in the sample.

- It divides the sum by one less than the number of items in the sample (N - 1). The result is the variance. Compare this final step to the PopulationVariance (fld), PopulationVariance (fld, condFld), PopulationVariance (fld, condFld, cond), PopulationVariance (x).

Note: Using this function in a formula forces the formula to be evaluated at print time.

For more information on evaluation time considerations, see Evaluation Time functions.

PopulationVariance (fld), PopulationVariance (fld, condFld), PopulationVariance (fld, condFld, cond), PopulationVariance (x)

Basic and Crystal syntax.

Arguments

- fld is any valid database or formula field that can be evaluated by the function.
- condFld is a field used to group the values in fld by.
- cond is a String indicating the type of grouping for condFld. You only specify this argument when condFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.
- x is an array of values that can be evaluated by the function being used.

Returns

Fractional Number

Action

Enables you to find the population variance of a set of values in your report. For example:

- You can calculate the grand total population variance for all values in a field. For information on this kind of summary, see SummaryFunction (fld).
- You can calculate the population variance for all values within a group (for example, sales grouped by the state that they come from). For information on this kind of summary, see SummaryFunction (fld, condFld).
- You can calculate the population variance for all values within a group in which grouping is controlled by changes in a date or Boolean field (for example, sales grouped by the month in which they were made). For information on this kind of summary, seeSummaryFunction (fld, condFld, cond).
- If you specify a set of individual values, you can compute the population variance of the values in the set. For information on this kind of summary, see Array summary functions (x).

Examples

The following examples are applicable to both Basic and Crystal syntax:

For examples of each PopulationVariance function, click on the corresponding summary function.

```
PopulationVariance({file.CENSUS})
```

Calculates the variance of all values in the Census field.

```
PopulationVariance({file.RESULTS}, {file.PROCEDURE})
```

Calculates the population variation for each group of laboratory test results in the Results field. The test results are separated into groups whenever the value in the Procedure field changes.

```
PopulationVariance({file.SCORES}, {file.NATURALIZED}, "any change")
```

Groups values in the Scores field based on whether or not the test taker is a naturalized citizen, and calculates the population variance for each group of scores.

The following examples are applicable to Crystal syntax:

```
PopulationVariance({file.SCORES}, {file.TEACHERS}) %
PopulationVariance({file.TEACHERS})
```

Groups values in the Scores field by teacher, and calculates the population variance for each teacher group as a percentage of the population variance of the values for the entire report (for all teachers listed).

```
PopulationVariance([2,4,6,8,10])
```

Returns 8.

Comments

Population variance is calculated using the following technique:

- It calculates the average (mean) value for the items in the population.
- It subtracts the average value from the value of each item.
- It squares the difference for each item.
- It adds the squared differences for all of the items in the population.
- It divides the sum by the number of items in the population (N). The result is the population variance. Compare this final step to the Variance (fld), Variance (fld, condFld), Variance (fld, condFld, cond), Variance (x).

Note: Using this function in a formula forces the formula to be evaluated at print time.

For more information on evaluation time considerations, see Evaluation Time functions.

Maximum (fld), Maximum (fld, condFld), Maximum (fld, condFld, cond), Maximum (x)

Basic and Crystal syntax.

Arguments

- fld is any valid database or formula field that can be evaluated by the function.
- condFld is a field used to group the values in fld by.
- cond is a String indicating the type of grouping for condFld. You only specify this argument when condFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.
- x is an array of values that can be evaluated by the function being used.

Returns

Fractional Number

Action

Enables you to find the minimum value that appears in a set of values. For example:

- If a sales report includes a field that shows the amount of each order, you can find the highest order amount of all the orders that appear on the report (a grand total maximum amount). For information on finding this kind of minimum, see SummaryFunction (fld).
- If you break orders into groups (for example, orders grouped by the state that they come from), you can find the highest order amount per group (in this case, per state). For information on finding this kind of maximum, see SummaryFunction (fld, condFld).
- If you break orders into date or Boolean groups (for example, orders grouped by the month in which they were placed), you can find the highest order amount per group based on a specific change in the date or Boolean field (in this case, per month). For information on finding this kind of maximum, see SummaryFunction (fld, condFld, cond).
- If you specify a set of individual values, you can find the highest value in the set. For information on finding this kind of maximum, see Array summary functions (x).

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
Maximum({file.QTY})
```

Returns the highest value in the QTY field.

```
Maximum({orders.AMOUNT}, {orders.CUSTOMER ID})
```

Identifies the largest order in each group of orders in the Amount field (the largest order for each month). The orders are separated into groups whenever the value in the Customer ID field changes.

```
Maximum({orders.AMOUNT}, {orders.ORDER DATE}, "monthly")
```

Identifies the largest order in each group of orders in the Amount field (the largest order for each month). The orders are separated into groups whenever the value in the Date field changes to a new month.

The following example is applicable to Crystal syntax:

Maximum also allows you to set a floor on a calculation. For example: Maximum([{file.BALANCE}, 500]) sets a floor of 500 on the calculation. The expression will always return the balance unless the balance drops below 500. Then it will return the floor amount of 500. Thus:

```
Maximum([{file.PROFITS}, 500]) = {file.PROFITS}
```

Where Profits > 500.

Note: Using this function in a formula forces the formula to be evaluated at print time.

For more information on evaluation time considerations, see Evaluation Time functions.

Minimum (fld), Minimum (fld, condFld), Minimum (fld, condFld, cond), Minimum (x)

Basic and Crystal syntax.

Arguments

- fld is any valid database or formula field that can be evaluated by the function.
- condFld is a field used to group the values in fld by.
- cond is a String indicating the type of grouping for condFld. You only specify this argument when condFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.
- x is an array of values that can be evaluated by the function being used.

Returns

Fractional Number

Action

Enables you to find the minimum value that appears in a set of values. For example:

- If a sales report includes a field that shows the amount of each order, you can find the lowest order amount of all the orders that appear on the report (a grand total minimum amount). For information on finding this kind of minimum, see SummaryFunction (fld).
- If you break orders into groups (for example, orders grouped by the state that they come from), you can find the lowest order amount per group (in this case, per state). For information on finding this kind of minimum, see SummaryFunction (fld, condFld).
- If you break orders into date or Boolean groups (for example, orders grouped by the month in which they were placed), you can find the lowest order amount per group based on a specific change in the date or Boolean field (in this case, per month). For information on finding this kind of minimum, see SummaryFunction (fld, condFld, cond).
- If you specify a set of individual values, you can find the lowest value in the set. For information on finding this kind of minimum, see Array summary functions (x).

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
Minimum({file.QTY})
```

Returns the lowest value in the QTY field.

```
Minimum({orders.AMOUNT}, {orders.CUSTOMER ID})
```

Identifies the smallest order in each group of orders in the Amount field (the smallest order for each month). The orders are separated into groups whenever the value in the Customer ID field changes.

```
Minimum({orders.AMOUNT}, {orders.ORDER DATE}, "monthly")
```

Identifies the smallest order in each group of orders in the Amount field (the smallest order for each month). The orders are separated into groups whenever the value in the Date field changes to a new month.

The following example is applicable to Crystal syntax:

Minimum also allows you to set a ceiling on a calculation. For example: Minimum ([{file.COMMISSION}, 2500]) returns the commission accrued up to a cap or ceiling of 2500. Once accrued commission passes the $2500 mark, this expression returns 2500. Thus:

```
Minimum([{file.COMMISSION}, 2500])
```

Returns 1575 where commission = 1575.

Note: Using this function in a formula forces the formula to be evaluated at print time.

For more information on evaluation time considerations, see Evaluation Time functions.

Count (fld), Count (fld, condFld), Count (fld, condFld, cond), Count (x)

Basic and Crystal syntax.

Arguments

- fld is any valid database or formula field that can be evaluated by the function.
- condFld is a field used to group the values in fld by.
- cond is a String indicating the type of grouping for condFld. You only specify this argument when condFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.
- x is an array of values that can be evaluated by the function being used.

Returns

Fractional Number

Action

Enables you to count the values that appear in your report (for a specified field). For example:

- If a sales report includes all orders made and the amount of each order, you can compute the total number of orders that appear on the report (a grand total count). For information on this kind of counting, see SummaryFunction (fld).
- If you break orders into groups (for example, orders grouped by the state that they come from), you can compute the number of orders per group (in this case, per state). For information on this kind of counting, see SummaryFunction (fld, condFld).
- If you break orders into date or Boolean groups (for example, orders grouped by the month in which they were placed), you can compute the number of orders per group based on a particular change in the date or Boolean field (in this case, per month). For information on this kind of counting, see SummaryFunction (fld, condFld, cond).
- If you specify a set of individual values, you can compute the number of values in the set. For information on this kind of counting, see Array summary functions (x).

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
Count({orders.AMOUNT}, {orders.CUSTOMER ID})
```

This formula counts the number of orders in each group of orders in the Amount field (the total orders for each month). The orders are separated into groups whenever the value in the Customer ID field changes.

```
Count({orders.ORDER AMOUNT}, {orders.ORDER DATE}, "monthly")
```

Counts the number of orders in each group of orders in the Amount field (the total orders for each month). The orders are separated into groups whenever the value in the Date field changes to a new month.

The following examples are applicable to Crystal syntax:

If Count({orders.ORDER ID}) >= 100 Then

"Congratulations on meeting your quota!"

Else

""

Prints the congratulatory message if the number of orders is 100 or more, and prints nothing if the number of orders is less than 100.

```
Count([1,2,3,4,5])
```

Returns 5. Counts the total number of values in the array.

Note: Using this function in a formula forces the formula to be evaluated at print time.

For more information on evaluation time considerations, see Evaluation Time functions.

DistinctCount (fld), DistinctCount (fld, condFld), DistinctCount (fld, condFld, cond), DistinctCount (x)

Basic and Crystal syntax.

Arguments

- fld is any valid database or formula field that can be evaluated by the function.
- condFld is a field used to group the values in fld by.
- cond is a String indicating the type of grouping for condFld. You only specify this argument when condFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.
- x is an array of values that can be evaluated by the function being used.

Returns

Fractional Number

Action

Enables you to get a distinct count of the values that appear in your report. For example:

- If a sales report includes all orders made by customers, you can compute the total number of distinct customers that appear in the report (a grand total distinct count), excluding any duplicate records. If a customer made more than one order, the duplicate occurrences of that customer are ignored. For information on this kind of distinct count, see SummaryFunction (fld).
- If you break orders into groups (for example, orders grouped by the state that they come from), you can compute the number of distinct customers per group (in this case, per state). Any customers that made more than one order and appear more than once in a group are only counted once. For information on this kind of distinct count, see SummaryFunction (fld, condFld).
- If you break orders into date or Boolean groups (for example, orders grouped by the month in which they were placed), you can compute the number of distinct customers in each group based on a particular change in the date or Boolean field (in this case, the number of customers that placed orders each month). If a customer placed more than one order within the month, duplicate instances of that customer are ignored. For information on this kind of distinct count, see SummaryFunction (fld, condFld, cond).
- If you specify a set of individual values, you can compute the number of distinct values in the set. Duplicate values in the set are only counted once. For information on this kind of distinct count, see Array summary functions (x).

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
DistinctCount({customer.REGION})
```

Counts the number of different States represented in the Region field, ignores any repetitions.

```
DistinctCount({customer.CITY}, {customer.REGION})
```

Calculates the number of different cities in each State group.

```
DistinctCount({orders.CUSTOMER ID}, {orders.ORDER DATE}, "weekly")
```

Counts the number of different customers contacted, follow up contacts are excluded. Customers are separated into groups whenever the value in the Date field changes to a new week.

The following example is applicable to Crystal syntax:

```
DistinctCount([1,3,5,3,2,5])
```

Returns 4. Counts the number of distinct values in the array. Duplicate values are ignored.

Note: Using this function in a formula forces the formula to be evaluated at print time.

For more information on evaluation time considerations, see Evaluation Time functions.

Comments

DistinctCount counts the number of distinct values in a field. Duplicate values are ignored.

Correlation (fld, fld), Correlation (fld, fld, condFld), Correlation (fld, fld, condFld, cond)

Basic and Crystal syntax.

Arguments

- fld is any numeric field.
- condFld is a field used to group the values in fld by.
- cond is a String indicating the type of grouping for condFld. You only specify this argument when condFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.

Returns

Fractional Number

Action

Enables you to calculate the correlation of the specified fields (that is, the degree to which the fields vary in the same manner). For example:

- You can calculate the correlation of two fields, for all the records in the report.
- You can calculate the correlation of two fields for all values within a group (for example, sales grouped by the state that they come from). For information on this kind of summary, see SummaryFunction (fld, condFld).
- You can calculate the correlation of two fields for all the values within a group in which grouping is controlled by changes in a date or Boolean field (for example, sales grouped by the month in which they were made). For information on this kind of summary, see SummaryFunction (fld, condFld, cond).

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
Correlation({Customer.CUSTOMER ID}, {Customer.CREDIT ID})
```

Calculates the correlation between the Customer ID and Credit ID fields.

```
Correlation({orders.CUSTOMER ID}, {orders.CREDIT ID},
{Customer.REGION})
```

Calculates the correlation between the Customer ID and Credit ID fields, for each region.

```
Correlation({orders.CUSTOMER ID}, {orders.CREDIT ID},
{Customer.REGION}, "monthly")
```

Calculates the correlation between the Customer ID and Credit ID fields for all the values within each Region group, for each month.

Note: Using this function in a formula forces the formula to be evaluated at print time.

For more information on evaluation time considerations, see Evaluation Time functions.

Comments

The correlation will always be a number between and 1, unless the correlation is undefined, in which case the function is assigned a null value (0). If the correlation is 0, the fields are uncorrelated.

Covariance (fld, fld), Covariance (fld, fld, condFld), Covariance (fld, fld, condFld, cond)

Basic and Crystal syntax.

Arguments

- fld is any numeric field that can be evaluated by the function.
- condFld is a field used to group the values in fld by.
- cond is a String indicating the type of grouping for condFld. You only specify this argument when condFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.

Returns

Fractional Number

Action

Covariance is the measure of the linear relation between paired variables (that is, the tendency of two fields to vary together). Fields are covariant when they vary according to a specific mathematical relationship. The circumference of a circle and the radius of a circle are covariant.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
Covariance({table.FIELD1}, {table.FIELD2})
```

Calculates the covariance of the two fields and returns the covariance as a fractional number.

Note: Using this function in a formula forces the formula to be evaluated at print time.

For more information on evaluation time considerations, see Evaluation Time functions.

Comments

Covariance is calculated using the following technique:

- It calculates the average (mean) value for the items in each sample.
- It subtracts the average value from the value of each item, in both samples.
- It multiplies the differences for each pair of items.

WeightedAverage (fld, fld), WeightedAverage (fld, fld, condFld), WeightedAverage (fld, fld, condFld, cond)

Basic and Crystal syntax.

Arguments

- fld is any valid numeric database or formula field that can be evaluated by the function.
- condFld is a field used to group the values in fld by.

- cond is a String indicating the type of grouping for condFld. You only specify this argument when condFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.

Returns

Fractional Number

Action

Enables you to calculate the weighted average of the specified fields. When you calculate a weighted average, you are actually calculating the average of one field and then using the values in another field to "weigh" the contribution of each value in the first field to the average. In a normal average, all the weights are equal to 1.

For example:

- You can calculate the weighted average of two fields. For information about this kind of summary, see SummaryFunction (fld).
- You can calculate the weighted average of two fields, for all the values within a group (for example, sales grouped by the state that they come from). For information on this kind of summary, see SummaryFunction (fld, condFld).
- You can calculate the weighted average of two fields, for all the values within a group in which grouping is controlled by changes in a date or Boolean field (for example, sales grouped by the month in which they were made). For information on this kind of summary, see SummaryFunction (fld, condFld, cond).

Examples

The following example is applicable to both Basic and Crystal syntax:

```
WeightedAverage({table.FIELD1}, {table.FIELD2})
```

Returns 3.5, where the two fields have four values each. Field1 has values of 3, 2, 5, and 1. Field2 has values of 0, 1, 1, and 0. WeightedAverage calculates as follows: (3*0 + 2*1 + 5*1 + 1*0)/`sum{table.FIELD2}`) = 3.5.

Note: Using this function in a formula forces the formula to be evaluated at print time. For more information on evaluation time considerations, see the Evaluation Time functions.

Median (fld), Median (fld, condFld), Median (fld, condFld, cond)

Basic and Crystal syntax.

Arguments

- fld is any valid database or formula field that can be evaluated by the function.
- condFld is a field used to group the values in fld by.
- cond is a String indicating the type of grouping for condFld. You only specify this argument when condFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.

Returns

Fractional Number

Action

Calculates the median of the given numeric fields. The median is the middle value in a sequence of numeric values (or the average of the two middle values in an even-numbered sequence of values).

For example:

- You can calculate the median of all the values in a field. For information on this kind of summary, see SummaryFunction (fld).
- You can calculate the median of all the values in the field, within a group (for example, sales grouped by the state that they come from). For information on this kind of summary, see SummaryFunction (fld, condFld).
- You can calculate the median of all the values within a group in which grouping is controlled by changes in a date or Boolean field (for example, sales grouped by the month in which they were made). For information on this kind of summary, see SummaryFunction (fld, condFld, cond).

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
Median({Customer.LAST YEAR'S SALES})
```

Returns $29087, where the values of the Last Year's Sales field are $27014, $28000, $29087, $34500, and $48260.

```
Median({Customer.LAST YEAR'S SALES}, {Customer.REGION)
```

Groups the Last Year's Sales field by region and returns the median of the Last Year's Sales field per region.

```
Median({orders.ORDER AMOUNT}, {Customer.REGION}, "monthly")
```

Groups values in the Amount field by region, then returns the median of the order amount for each region per month.

Note: Using this function in a formula forces the formula to be evaluated at print time.

For more information on evaluation time considerations, see Evaluation Time functions.

PthPercentile (P, fld), PthPercentile (P, fld, condFld), PthPercentile (P, fld, condFld, cond)

Basic and Crystal syntax.

Arguments

- P is any integer between 0 and 100.
- fld is any Number or Currency field that can be evaluated by the function.
- condFld is a field used to group the values in fld by.

- cond is a String indicating the type of grouping for condFld. You only specify this argument when condFld is a Date, Time, DateTime or Boolean field.

  For more information on the valid strings for this argument, see Conditions for summary functions.

Returns
Number or Currency value.

Action
Calculates the value for a specified percentile (P) in a Number or Currency field.

Examples
The following examples are applicable to both Basic and Crystal syntax:

```
PthPercentile(20, {Customer.LAST YEAR'S SALES})
```

Returns $2302 as the value at the 20th percentile (if 20% of {Customer.LAST YEAR'S SALES are less than $2302

```
PthPercentile(P, {Customer.LAST YEAR'S SALES}, {Customer.REGION})
```

Groups the Last Year's Sales field by region and returns the value at percentile P in the Last Year's Sales field per region.

```
PthPercentile(P,{orders.ORDER AMOUNT}, {Customer.REGION}, "monthly")
```

Groups values in the Amount field by region, then returns the value at percentile P in the Amount field for each region, per month.

Comments
P = 50 (the 50th Percentile) is the same value returned by the Median (fld), Median (fld, condFld), Median (fld, condFld, cond) function (the median value).

Note: Using this function in a formula forces the formula to be evaluated at print time.

For more information on evaluation time considerations, see Evaluation Time functions.

NthLargest (N, fld), NthLargest (N, fld, condFld), NthLargest (N, fld, condFld, cond)

Basic and Crystal syntax.

Arguments

- N is any integer from 1 to 100 (inclusive).
- fld is any valid database or formula field that can be evaluated by the function.
- condFld is a field used to group the values in fld by.
- cond is a String indicating the type of grouping for condFld. You only specify this argument when condFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.

Returns

Fractional Number

Action

Determines the Nth largest value in a given field, either for the entire report or for each instance of the (condFld) group.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
NthLargest(1, {Customer.CUSTOMER ID}
```

Returns 50, where the Customer ID field contains numerical values ranging from 12 to 50.

```
NthLargest(5, {Customer.CUSTOMER NAME}, {Customer.REGION})
```

Returns the fifth largest value in the Customer Name field, per region.

```
NthLargest(1, {orders.ORDER AMOUNT}, {Customer.REGION}, "monthly")
```

Groups values in the Amount field by region, then returns the largest value in the Amount field for each region, per month.

Note: Using this function in a formula forces the formula to be evaluated at print time.

For more information on evaluation time considerations, see Evaluation Time functions.

NthSmallest (N, fld), NthSmallest (N, fld, condFld), NthSmallest (N, fld, condFld, cond)

Basic and Crystal syntax.

Arguments

- N is any integer from 1 to 100 (inclusive).
- fld is any valid database or formula field that can be evaluated by the function.
- condFld is a field used to group the values in fld by.
- cond is a String indicating the type of grouping for condFld. You only specify this argument when condFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.

Returns

Fractional Number

Action

Determines the Nth smallest value in a given field, either for the entire report or for each instance of the (condFld) group.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
NthSmallest (1, {Customer.CUSTOMER ID}
```

Returns 12, where the Customer ID field contains numerical values ranging from 12 to 50.

```
NthSmallest(1, {Customer.CUSTOMER NAME}, {Customer.REGION})
```

Returns the least value in the Customer Name field, per region.

```
NthSmallest({orders.ORDER AMOUNT}, {Customer.REGION}, "monthly")
```

Groups values in the Amount field by region, then returns the least value in the Amount field for each region, per month.

Note: Using this function in a formula forces the formula to be evaluated at print time.

For more information on evaluation time considerations, see Evaluation Time functions.

NthMostFrequent (N, fld), NthMostFrequent (N, fld, condFld), NthMostFrequent (N, fld, condFld, cond)

Basic and Crystal syntax.

Arguments

- N is any integer from 1 to 100 (inclusive).
- fld is any valid database or formula field that can be evaluated by the function.
- condFld is a field used to group the values in fld by.
- cond is a String indicating the type of grouping for condFld. You only specify this argument when condFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.
- Returns Fractional Number

Action

Determines the Nth most frequent value in a given field, either for the entire report or for each instance of the (condFld) group. If no values in the field appear more than once, the function will return the minimum value, by default.

Typical Uses

You can use this function to spotlight exceptionally high or exceptionally low values in a particular field.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
NthMostFrequent(1, {Customer.LAST YEAR'S SALES})
```

Returns $29087, where the value of $29087 appears most frequently in the Last Year's Sales field.

```
NthMostFrequent(2, {Customer.LAST YEAR'S SALES})
```

Returns $34700, where $34700 is the value that appears with the second highest frequency in the Last Year's Sales field.

```
NthMostFrequent(1, {Customer.LAST YEAR'S SALES}, {Customer.REGION})
```

Returns $48000 for one instance of the Customer.Region group, where $48000 is the value that appears most frequently in that group; returns $34000 for another instance of the group, where $34000 is the most frequent value for that occurrence of the group; and returns $9000 for a third instance of the group, where none of the values in this occurrence of the group are repeated, and $9000 is the lowest value.

Comments

When more than one value appears with the same frequency, the least value is considered the most frequent. For example, if

```
NthMostFrequent(1, {Customer.LAST YEAR'S SALES})
```

returns a value of $29087, and a value greater than $29087 (for example, $3500) appears with the same frequency, NthMostFrequent with N = 1 will still return $29087, since $29087 is the lesser value.

Note: Using this function in a formula forces the formula to be evaluated at print time.

For more information on evaluation time considerations, see Evaluation Time functions.

Mode (fld), Mode (fld, condFld), Mode (fld, condFld, cond)

Basic and Crystal syntax.

Arguments

- fld is any valid database or formula field that can be evaluated by the function.
- condFld is a field used to group the values in fld by.
- cond is a String indicating the type of grouping for condFld. You only specify this argument when condFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.

Returns

Fractional Number

Action

Identifies the most frequently occurring value.

For example:

- You can calculate the mode of all the values in a field. For information on this kind of summary, see SummaryFunction (fld).
- You can calculate the mode of all the values in the field, within a group (for example, sales grouped by the state that they come from). For information on this kind of summary, see SummaryFunction (fld, condFld).
- You can calculate the mode of all the values within a group in which grouping is controlled by changes in a date or Boolean field (for example, sales grouped by the month in which they were made). For information on this kind of summary, see SummaryFunction (fld, condFld, cond).

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
Mode({Customer.CUSTOMER NAME})
```

Returns the mode (the most frequently occurring value) for the Customer Name field.

```
Mode({Customer.CUSTOMER NAME}, {Customer.REGION})
```

Groups values in the Customer Name field by region, and then returns the mode for the Customer Name field, per region.

```
Mode({Customer.CUSTOMER NAME}, {Customer.REGION}, "monthly")
```

Groups values in the Customer Name field by region, then returns the mode of the Customer Name field for each region, per month.

Comments

Mode works like NthMost Frequent when N = 1.

Note: Using this function in a formula forces the formula to be evaluated at print time.

For more information on evaluation time considerations, see Evaluation Time functions.

PercentOfSum

Basic and Crystal syntax.

Overloads

- PercentOfSum (fld, condFld)
- PercentOfSum (fld, condFld, cond)
- PercentOfSum (fld, innerCondFld, outerCondFld)
- PercentOfSum (fld, innerCondFld, innerCond, outerCondFld)
- PercentOfSum (fld, innerCondFld, outerCondFld, outerCond)
- PercentOfSum (fld, innerCondFld, innerCond, outerCondFld, outerCond)

Arguments

- fld is a Number or Currency field that can be evaluated by the function.
- condFld is a field used to group the values in fld by.
- cond is a String indicating the type of grouping for condFld. You only specify this argument when condFld is a Date, Time, DateTime or Boolean field.

  For more information on the valid strings for this argument, see Conditions for summary functions.
- innerCondFld is a field used to group the values in fld by.
- innerCond is a String indicating the type of grouping for innerCondFld. You only specify this argument when innerCondFld is a Date, Time, DateTime or Boolean field.
- outerCondFld is a field used to group the values in fld by.

- outerCond is a String indicating the type of grouping for outerCondFld. You only specify this argument when outerCondFld is a Date, Time, DateTime or Boolean field.

  For more information on the valid strings for this argument, see Conditions for summary functions.

Returns

Number value.

Action

PercentOfSum (fld, condFld) has the same effect as: 100 * Sum (fld, condFld) / Sum (fld). It expresses the sum of the values of the field fld for the group determined by condFld as a percentage of the grand total sum.

PercentOfSum (fld, condFld, cond) has the same effect as: 100 * Sum (fld, condFld, cond) / Sum (fld). The String argument cond expresses some additional information to precisely specify the grouping. For example, if condFld is a Date field, is the grouping "daily" or "weekly" or "monthly" etc.

PercentOfSum (fld, innerCondFld, outerCondFld) has the same effect as: 100 * Sum (fld, innerCondFld) / Sum (fld, outerCondFld). It expresses the sum of the values of the field fld for the group determined by innerCondFld as a percentage of the sum for the group determined by outerCondFld.

PercentOfSum (fld, innerCondFld, innerCond, outerCondFld) has the same effect as: 100 * Sum (fld, innerCondFld, innerCond) / Sum (fld, outerCondFld).

PercentOfSum (fld, innerCondFld, outerCondFld, outerCond) has the same effect as: 100 * Sum (fld, innerCondFld) / Sum (fld, outerCondFld, outerCond).

PercentOfSum (fld, innerCondFld, innerCond, outerCondFld, outerCond) has the same effect as: 100 * Sum (fld, innerCondFld, innerCond) / Sum (fld, outerCondFld, outerCond).

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
PercentOfSum ({Orders.Order Amount}, {Orders.Order Date}, "annually")
```

Returns the total value of the orders that were ordered in a given year expressed as a percent of the total value of all orders.

```
PercentOfSum ({Orders.Order Amount}, {Orders.Ship Via})
```

Returns the total value of the orders shipped via a given shipping company expressed as a percent of the total value of all orders.

```
PercentOfSum ({Product.Price (SRP)}, {Product.Size}, {Product.Product
Class})
```

Returns the total price of the products of a given size and class expressed as a percent of the total price of the products of the same given class.

Comment

This function and other percentage functions are intended to support Percentage Summary fields. The groupings supplied as arguments must actually exist in the report for the summary function call to succeed.

PercentOfAverage

Basic and Crystal syntax.

Overloads

- PercentOfAverage (fld, condFld)
- PercentOfAverage (fld, condFld, cond)
- PercentOfAverage (fld, innerCondFld, outerCondFld)
- PercentOfAverage (fld, innerCondFld, innerCond, outerCondFld)
- PercentOfAverage (fld, innerCondFld, outerCondFld, outerCond)
- PercentOfAverage (fld, innerCondFld, innerCond, outerCondFld, outerCond)

Arguments

- fld is a Number or Currency field that can be evaluated by the function.
- condFld is a field used to group the values in fld by.
- cond is a String indicating the type of grouping for condFld. You only specify this argument when condFld is a Date, Time, DateTime or Boolean field.
- innerCondFld is a field used to group the values in fld by.
- innerCond is a String indicating the type of grouping for innerCondFld. You only specify this argument when innerCondFld is a Date, Time, DateTime or Boolean field.
- outerCondFld is a field used to group the values in fld by.
- outerCond is a String indicating the type of grouping for outerCondFld. You only specify this argument when outerCondFld is a Date, Time, DateTime or Boolean field.

For more information on the valid strings for these arguments, see Conditions for summary functions.

Returns

Number value.

Action

PercentOfAverage (fld, condFld) has the same effect as: 100 * Average (fld, condFld) / Average (fld). It expresses the average of the values of the field fld for the group determined by condFld as a percentage of the average of all the values of fld.

PercentOfAverage (fld, condFld, cond) has the same effect as: 100 * Average (fld, condFld, cond) / Average (fld). The String argument cond expresses some additional information to precisely specify the grouping. For example, if condFld is a Date field, is the grouping "daily" or "weekly" or "monthly" etc.

PercentOfAverage (fld, innerCondFld, outerCondFld) has the same effect as: 100 * Average (fld, innerCondFld) / Average (fld, outerCondFld). It expresses the average of the values of the field fld for the group determined by innerCondFld as a percentage of the average for the group determined by outerCondFld.

PercentOfAverage (fld, innerCondFld, innerCond, outerCondFld) has the same effect as: 100 * Average (fld, innerCondFld, innerCond) / Average (fld, outerCondFld).

PercentOfAverage (fld, innerCondFld, outerCondFld, outerCond) has the same effect as: 100 * Average (fld, innerCondFld) / Average (fld, outerCondFld, outerCond).

PercentOfAverage (fld, innerCondFld, innerCond, outerCondFld, outerCond) has the same effect as: 100 * Average (fld, innerCondFld, innerCond) / Average (fld, outerCondFld, outerCond)

Comment

This function and other percentage functions are intended to support Percentage Summary fields. The groupings supplied as arguments must actually exist in the report for the summary function call to succeed.

PercentOfMaximum

Basic and Crystal syntax

Overloads

- PercentOfMaximum (fld, condFld)
- PercentOfMaximum (fld, condFld, cond)
- PercentOfMaximum (fld, innerCondFld, outerCondFld)
- PercentOfMaximum (fld, innerCondFld, innerCond, outerCondFld)
- PercentOfMaximum (fld, innerCondFld, outerCondFld, outerCond)
- PercentOfMaximum (fld, innerCondFld, innerCond, outerCondFld, outerCond)

Arguments

- fld is a Number or Currency field that can be evaluated by the function.
- condFld is a field used to group the values in fld by.
- cond is a String indicating the type of grouping for condFld. You only specify this argument when condFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.
- innerCondFld is a field used to group the values in fld by.
- innerCond is a String indicating the type of grouping for innerCondFld. You only specify this argument when innerCondFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.
- outerCondFld is a field used to group the values in fld by.
- outerCond is a String indicating the type of grouping for outerCondFld. You only specify this argument when outerCondFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see C Conditions for summary functions.

Returns

Number value.

Action

PercentOfMaximum (fld, condFld) has the same effect as: 100 * Maximum (fld, condFld) / Maximum (fld). It expresses the maximum of the values of the field fld for the group determined by condFld as a percentage of the maximum of all the values of fld.

PercentOfMaximum (fld, condFld, cond) has the same effect as: 100 * Maximum (fld, condFld, cond) / Maximum (fld). The String argument cond expresses some additional information to precisely specify the grouping. For example, if condFld is a Date field, is the grouping "daily" or "weekly" or "monthly" etc.

PercentOfMaximum (fld, innerCondFld, outerCondFld) has the same effect as: 100 * Maximum (fld, innerCondFld) / Maximum (fld, outerCondFld). It expresses the maximum of the values of the field fld for the group determined by innerCondFld as a percentage of the maximum for the group determined by outerCondFld.

PercentOfMaximum (fld, innerCondFld, innerCond, outerCondFld) has the same effect as: 100 * Maximum (fld, innerCondFld, innerCond) / Maximum (fld, outerCondFld)

PercentOfMaximum (fld, innerCondFld, outerCondFld, outerCond) has the same effect as: 100 * Maximum (fld, innerCondFld) / Maximum (fld, outerCondFld, outerCond).

PercentOfMaximum (fld, innerCondFld, innerCond, outerCondFld, outerCond) has the same effect as: 100 * Maximum (fld, innerCondFld, innerCond) / Maximum (fld, outerCondFld, outerCond).

Comment

This function and other percentage functions are intended to support Percentage Summary fields. The groupings supplied as arguments must actually exist in the report for the summary function call to succeed.

PercentOfMinimum

Basic and Crystal syntax.

Overloads

- PercentOfMinimum (fld, condFld)
- PercentOfMinimum (fld, condFld, cond)
- PercentOfMinimum (fld, innerCondFld, outerCondFld)
- PercentOfMinimum (fld, innerCondFld, innerCond, outerCondFld)
- PercentOfMinimum (fld, innerCondFld, outerCondFld, outerCond)
- PercentOfMinimum (fld, innerCondFld, innerCond, outerCondFld, outerCond)

Arguments

- fld is a Number or Currency field that can be evaluated by the function.
- condFld is a field used to group the values in fld by.
- cond is a String indicating the type of grouping for condFld. You only specify this argument when condFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.
- innerCondFld is a field used to group the values in fld by.

- innerCond is a String indicating the type of grouping for innerCondFld. You only specify this argument when innerCondFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.
- outerCondFld is a field used to group the values in fld by.
- outerCond is a String indicating the type of grouping for outerCondFld. You only specify this argument when outerCondFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.

Returns

Number value.

Action

PercentOfMinimum (fld, condFld) has the same effect as: 100 * Minimum (fld, condFld) / Minimum (fld). It expresses the minimum of the values of the field fld for the group determined by condFld as a percentage of the minimum of all the values of fld.

PercentOfMinimum (fld, condFld, cond) has the same effect as: 100 * Minimum (fld, condFld, cond) / Minimum (fld). The String argument cond expresses some additional information to precisely specify the grouping. For example, if condFld is a Date field, is the grouping "daily" or "weekly" or "monthly" etc.

PercentOfMinimum (fld, innerCondFld, outerCondFld) has the same effect as: 100 * Minimum (fld, innerCondFld) / Minimum (fld, outerCondFld). It expresses the minimum of the values of the field fld for the group determined by innerCondFld as a percentage of the minimum for the group determined by outerCondFld.

PercentOfMinimum (fld, innerCondFld, innerCond, outerCondFld) has the same effect as: 100 * Minimum (fld, innerCondFld, innerCond) / Minimum (fld, outerCondFld).

PercentOfMinimum (fld, innerCondFld, outerCondFld, outerCond) has the same effect as: 100 * Minimum (fld, innerCondFld) / Minimum (fld, outerCondFld, outerCond).

PercentOfMinimum (fld, innerCondFld, innerCond, outerCondFld, outerCond) has the same effect as: 100 * Minimum (fld, innerCondFld, innerCond) / Minimum (fld, outerCondFld, outerCond).

Comment

This function and other percentage functions are intended to support Percentage Summary fields. The groupings supplied as arguments must actually exist in the report for the summary function call to succeed.

PercentOfCount

Basic and Crystal syntax.

Overloads

- PercentOfCount (fld, condFld)
- PercentOfCount (fld, condFld, cond)
- PercentOfCount (fld, innerCondFld, outerCondFld)
- PercentOfCount (fld, innerCondFld, innerCond, outerCondFld)

- PercentOfCount (fld, innerCondFld, outerCondFld, outerCond)
- PercentOfCount (fld, innerCondFld, innerCond, outerCondFld, outerCond)

Arguments

- fld is a Number, Currency, String, Boolean, Date, Time or DateTime field that can be evaluated by the function.
- condFld is a field used to group the values in fld by.
- cond is a String indicating the type of grouping for condFld. You only specify this argument when condFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.
- innerCondFld is a field used to group the values in fld by.
- innerCond is a String indicating the type of grouping for innerCondFld. You only specify this argument when innerCondFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.
- outerCondFld is a field used to group the values in fld by.
- outerCond is a String indicating the type of grouping for outerCondFld. You only specify this argument when outerCondFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.

Returns

Number value.

Action

PercentOfCount (fld, condFld) has the same effect as: 100 * Count (fld, condFld) / Count (fld). It expresses the count of the values of the field fld for the group determined by condFld as a percentage of the count of all the values of fld.

PercentOfCount (fld, condFld, cond) has the same effect as: 100 * Count (fld, condFld, cond) / Count (fld). The String argument cond expresses some additional information to precisely specify the grouping. For example, if condFld is a Date field, is the grouping "daily" or "weekly" or "monthly" etc.

PercentOfCount (fld, innerCondFld, outerCondFld) has the same effect as: 100 * Count (fld, innerCondFld) / Count (fld, outerCondFld). It expresses the count of the values of the field fld for the group determined by innerCondFld as a percentage of the count for the group determined by outerCondFld.

PercentOfCount (fld, innerCondFld, innerCond, outerCondFld) has the same effect as: 100 * Count (fld, innerCondFld, innerCond) / Count (fld, outerCondFld).

PercentOfCount (fld, innerCondFld, outerCondFld, outerCond) has the same effect as: 100 * Count (fld, innerCondFld) / Count (fld, outerCondFld, outerCond).

PercentOfCount (fld, innerCondFld, innerCond, outerCondFld, outerCond) has the same effect as: 100 * Count (fld, innerCondFld, innerCond) / Count (fld, outerCondFld, outerCond).

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
PercentOfCount ({Orders.Order ID}, {Orders.Order Date}, "annually")
```

Returns the total number of orders that were ordered in a given year expressed as a percent of the total number of all orders.

```
PercentOfCount ({Orders.Order ID}, {Orders.Ship Via})
```

Returns the total number of orders shipped via a given shipping company expressed as a percent of the total number of all orders.

```
PercentOfCount ({Product.Product ID}, {Product.Size},
{Product.Product Class})
```

Returns the total number of products of a given size and class expressed as a percent of the total number of products of the same given class.

Comment

This function and other percentage functions are intended to support Percentage Summary fields. The groupings supplied as arguments must actually exist in the report for the summary function call to succeed.

PercentOfDistinctCount

Basic and Crystal syntax.

Overloads

- PercentOfDistinctCount (fld, condFld)
- PercentOfDistinctCount (fld, condFld, cond)
- PercentOfDistinctCount (fld, innerCondFld, outerCondFld)
- PercentOfDistinctCount (fld, innerCondFld, innerCond, outerCondFld)
- PercentOfDistinctCount (fld, innerCondFld, outerCondFld, outerCond)
- PercentOfDistinctCount (fld, innerCondFld, innerCond, outerCondFld, outerCond)

Arguments

- fld is a Number, Currency, String, Boolean, Date, Time or DateTime field that can be evaluated by the function.
- condFld is a field used to group the values in fld by.
- cond is a String indicating the type of grouping for condFld. You only specify this argument when condFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.
- innerCondFld is a field used to group the values in fld by.
- innerCond is a String indicating the type of grouping for innerCondFld. You only specify this argument when innerCondFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.
- outerCondFld is a field used to group the values in fld by.
- outerCond is a String indicating the type of grouping for outerCondFld. You only specify this argument when outerCondFld is a Date, Time, DateTime or Boolean field. For more information on the valid strings for this argument, see Conditions for summary functions.

Returns

Number value.

Action

PercentOfDistinctCount (fld, condFld) has the same effect as: 100 * DistinctCount (fld, condFld) / DistinctCount (fld). It expresses the distinct count of the values of the field fld for the group determined by condFld as a percentage of the distinct count of all the values of fld.

PercentOfDistinctCount (fld, condFld, cond) has the same effect as: 100 * DistinctCount (fld, condFld, cond) / DistinctCount (fld). The String argument cond expresses some additional information to precisely specify the grouping example: if condFld is a Date field, is the grouping "daily" or "weekly" or "monthly" etc.

PercentOfDistinctCount (fld, innerCondFld, outerCondFld) has the same effect as: 100 * DistinctCount (fld, innerCondFld) / DistinctCount (fld, outerCondFld). It expresses the distinct count of the values of the field fld for the group determined by innerCondFld as a percentage of the distinct count for the group determined by outerCondFld.

PercentOfDistinctCount (fld, innerCondFld, innerCond, outerCondFld) has the same effect as: 100 * DistinctCount (fld, innerCondFld, innerCond) / DistinctCount (fld, outerCondFld).

PercentOfDistinctCount (fld, innerCondFld, outerCondFld, outerCond) has the same effect as: 100 * DistinctCount (fld, innerCondFld) / DistinctCount (fld, outerCondFld, outerCond)

PercentOfDistinctCount (fld, innerCondFld, innerCond, outerCondFld, outerCond) has the same effect as: 100 * DistinctCount (fld, innerCondFld, innerCond) / DistinctCount (fld, outerCondFld, outerCond).

Comment

This function and other percentage functions are intended to support Percentage Summary fields. The groupings supplied as arguments must actually exist in the report for the summary function call to succeed.

Financial functions

Click the function of interest from the following list for further information:

DDB

FV

IPmt

IRR

MIRR (values, financeRate, reinvestRate)

NPer

NPV (rate, values)

Pmt

PPmt

PV

Rate

SLN (cost, salvage, life)

SYD (cost, salvage, life, period)

DDB

Basic and Crystal syntax.

Overloads

- DDB (cost, salvage, life, period)
- DDB (cost, salvage, life, period, factor)

Arguments

- cost is a Number or Currency that specifies the initial cost of the asset. The value is nonnegative and greater than or equal to salvage.
- salvage is a Number or Currency that specifies the value of the asset at the end of its useful life. The value is nonnegative.
- life is a positive Number that specifies the length of the useful life of the asset.
- period is a Number that specifies the period for which asset depreciation is calculated. The value is positive and less than or equal to life. The arguments life and period must have the same units.
- factor is an optional positive Number that specifies the rate at which the balance declines. If omitted, 2 (double-declining method) is assumed.

Returns

Number value

Action

DDB returns a Number specifying the depreciation of an asset for a specific time period, using the double-declining balance method or some other method as specified by the factor argument.

Examples

The following examples are applicable to both Basic and Crystal syntax:

Suppose a company purchases a fleet of cars for $500,000. The cars have a lifetime of 10 years and a salvage value of $60,000. They are depreciated using the double-declining method.

```
DDB (500000, 60000, 10, 1)
```

Returns 100000. The first year's depreciation is $100,000.

```
DDB (500000, 60000, 10, 4)
```

Returns 51200. The fourth year's depreciation is $51,200.

```
DDB (500000, 60000, 10, 10)
```

Returns 7108.864. The final year's depreciation is $7,108.86.

Comment

This function is designed to work like the Visual Basic function of the same name.

FV

Basic and Crystal syntax.

Overloads

- FV (rate, nPeriods, payment)
- FV (rate, nPeriods, payment, presentValue)
- FV (rate, nPeriods, payment, presentValue, type)

Arguments

- rate is a Number that specifies the interest rate per period.
- nPeriods is a positive Number that specifies the total number of payment periods in the annuity. The units used for specifying rate and nPeriods must consistent. For example, if nPeriods is the number of periods in months, then rate is a monthly interest rate.
- payment is a Number or Currency that specifies the payment to be made each period.
- presentValue is an optional Number or Currency that specifies the present value of a series of future payments.
- Type is an optional Number that specifies when payments are due. Specify 0 if payments are due at the end of the payment period, and 1 if payments are due at the beginning of the period. If omitted, 0 is assumed.

Returns

Number value

Action

FV returns a Number specifying the future value of an annuity based on periodic, fixed payments and a fixed interest rate.

Examples

The following examples are applicable to both Basic and Crystal syntax:

Suppose that you put $1000 per month into a retirement savings plan that pays 6 percent annual interest, compounded monthly. How much will the account be worth after 20 years?

```
FV (0.06 / 12, 20 * 12, 000)
```

Returns 462041 (rounded to the nearest integer). So your account will have $462,041. The payment (000) is negative since you are paying out the money to the plan.

The above example assumes that you made your payments into the plan at the end of the month. Thus after the first month, your plan would have only $1000 in it, since there was no time for interest to accrue. Suppose that instead you make your payments at the start of the month:

```
FV (0.06 / 12, 20 * 12, 000, 0, 1)
```

Returns 464351 (rounded to the nearest integer). So your account will have $464,351. You will save $2,310 more by depositing at the beginning of the month.

Now suppose that in addition to making payments at the start of the month, you start your plan with an initial deposit of $20,000.

```
FV (0.06 / 12, 20 * 12, 000, 0000, 1)
```

Returns 530555 (rounded to the nearest integer). Your account will have $530,555 after 20 years.

You can also use the FV function to calculate the future value of a lump sum deposit. For example, if you deposit $20,000 in to a plan that pays 6 percent annual interest compounded monthly for 20 years:

```
FV (0.06 / 12, 20 * 12, 0, 0000)
```

Returns 66204 (rounded to the nearest integer). Thus you would have $66,204 in your account. This is also equal to the difference of the previous 2 examples ($530,555 - $464,351).

Comment

This function is designed to work like the Visual Basic function of the same name.

IPmt

Basic and Crystal syntax.

Overloads

- IPmt (rate, period, nPeriods, presentValue)
- IPmt (rate, period, nPeriods, presentValue, futureValue)
- IPmt (rate, period, nPeriods, presentValue, futureValue, type)

Arguments

- rate is a Number that specifies the interest rate per period.
- period is a Number that specifies the payment period in the range 1 through nPeriods.
- nPeriods is a positive Number that specifies the total number of payment periods in the annuity. The units used for specifying rate, period and nPeriods must be consistent. For example, if nPeriods is the number of periods in months, then rate is a monthly interest rate and period specifies a month.
- presentValue is a Number or Currency that specifies the present value, or value today, of a series of future payments or receipts.
- futureValue is an optional Number or Currency that specifies the future value or cash balance you want after you've made the final payment. If omitted, 0 is assumed.
- type is an optional Number that specifies when payments are due. Specify 0 if payments are due at the end of the payment period, and 1 if payments are due at the beginning of the period. If omitted, 0 is assumed.

Returns

Number value

Action

IPmt returns a Number specifying the interest payment for a given period of an annuity based on periodic, fixed payments and a fixed interest rate.

Examples

The following example is applicable to both Basic and Crystal syntax:

Suppose that you want to take out a $250,000 loan payable monthly over 15 years at an annual interest rate of 7 percent. The following formula returns the amount of interest that you pay in your first loan payment. Note that the monthly interest rate is 0.07 / 12 and the number of months of the loan is 15 * 12.

```
IPmt (0.07 / 12, 1, 15 * 12, 250000)
```

Returns the Number value 458.33 (rounded to 2 decimals). The value is negative because it represents a payment out from you whereas the loan amount of $250,000 is positive because it represents a payment in to you.

The following formula returns the amount of interest that you pay in your 121st payment (after 10 years of payments):

```
IPmt (0.07 / 12, 10*12 + 1, 15 * 12, 250000)
```

Returns 61.98 (rounded to 2 decimals). You've made progress on the loan and so less of your monthly payment is for paying interest.

Comment

This function is designed to work like the Visual Basic function of the same name.

IRR

Basic and Crystal syntax.

Overloads

- IRR (values)
- IRR (values, guess)

Arguments

- values is a Number or Currency type array that specifies cash flow values. The array must contain at least one negative value (a payment) and one positive value (a receipt). The cash flows must occur at regular intervals such as monthly or yearly.
- guess is an optional Number value that is estimated to be returned by IRR. If omitted, guess is 0.1 (10 percent).

Returns

Number value

Action

IRR returns a Number specifying the internal rate of return for a series of periodic cash flows (payments and receipts).

Example

The following example is applicable to both Basic and Crystal syntax:

Suppose that you can choose one of two offers: $20,000 now or guaranteed payments of $5,000 after 1 year, $10,000 after 2 years and $15,000 after 3 years. What is the better offer? One way to quantify this is to calculate the internal rate of return. If you take the second offer, you can't take the first so that is like experiencing an initial payment of $20,000 followed by the receipts:

```
Rem Basic syntax

formula = IRR (Array(0000, 5000, 10000, 15000))


//Crystal syntax

IRR ([0000, 5000, 10000, 15000])
```

Returns 0.194 (rounded to 3 decimals) or 19.4 percent interest. So all other things being equal, if you think that 19.4 percent is a good rate of return, you would prefer the second offer.

Comments

- This function is designed to work like the Visual Basic function of the same name.
- The NPV and IRR functions are related since NPV (IRR (values), values) = 0. That is, the internal rate of return of a sequence of cash flows is the interest rate for which that sequence of cash flows has a net present value of 0.
- There is no direct formula for the IRR function and so Crystal Reports calculates the value by iteration. The process depends on the initial guess for the internal rate of return. If the program reports an error, try changing the value of the guess argument to be closer to what you expect the internal rate of return should be.

MIRR (values, financeRate, reinvestRate)

Basic and Crystal syntax.

Arguments

- values is a Number or Currency type array that specifies cash flow values. The array must contain at least one negative value (a payment) and one positive value (a receipt). The cash flows must occur at regular intervals such as monthly or yearly.
- financeRate is a Number that specifies the interest rate paid as the cost of financing.
- reinvestRate is a Number that specifies the interest rate received on gains from cash reinvestment.

Returns

Number value

Action

MIRR returns a Number specifying the modified internal rate of return for a series of periodic cash flows (payments and receipts).

Example

The following example is applicable to both Basic and Crystal syntax:

Suppose that you run a business that makes equipment investments, which result in a loss in your first and fourth years. Your expected annual returns are: -$50,000, $40,000, $65,000, -$60,000, $50,000, $70,000. Your losses are financed at 10 percent while you reinvest your earnings in an account at 6 percent. The modified internal rate of return is:

Rem Basic syntax

formula = MIRR(Array(0000, 40000, 65000, 0000, 50000, 70000), 0.10, 0.06)

//Crystal syntax

MIRR([0000, 40000, 65000, 0000, 50000, 70000], 0.10, 0.06)

Returns 0.214 (rounded to 3 decimals) or 21.4 percent.

Comment

This function is designed to work like the Visual Basic function of the same name.

NPer

Basic and Crystal syntax.

Overloads

- NPer (rate, payment, presentValue)
- NPer (rate, payment, presentValue, futureValue)
- NPer (rate, payment, presentValue, futureValue, type)

Arguments

- rate is a Number that specifies the interest rate per period.
- payment is a Number or Currency that specifies the payment to be made each period. Payments usually contain principal and interest that doesn't change over the life of the annuity.
- presentValue is a Number or Currency that specifies the present value, or value today, of a series of future payments or receipts.
- futureValue is an optional Number or Currency that specifies the future value or cash balance you want after you've made the final payment. If omitted, 0 is assumed.
- type is an optional Number that specifies when payments are due. Specify 0 if payments are due at the end of the payment period, and 1 if payments are due at the beginning of the period. If omitted, 0 is assumed.

Returns

Number value

Action

NPer returns a Number specifying the number of periods for an annuity based on periodic, fixed payments and a fixed interest rate.

Examples

The following examples are applicable to both Basic and Crystal syntax:

Suppose that you want a $200,000 loan to buy a house. The interest rate is 7 percent and you can afford to pay $2500 per month. How long of a mortgage do you need?

```
NPer (0.07/12, 500, 200000)
```

Returns 108.08 (rounded to 2 decimals) months. That is 108.08 / 12 = 9 years.

Rather than seeking a loan, suppose that you want to save up $200,000 and then buy the house. You can get a 7 percent interest rate compounded monthly from your savings plan and want to save $2500 per month. How long will you need to save to have the money?

```
NPer (0.07/12, 500, 0, 200000)
```

Returns 65.85 (rounded to 2 decimals) months. That is about 5.5 years.

Comment

This function is designed to work like the Visual Basic function of the same name.

NPV (rate, values)

Basic and Crystal syntax.

Arguments

- rate is a Number that specifies the discount rate over the length of the period, expressed as a decimal.
- values is a Number or Currency type array that specifies cash flow values. Negative values represent payments and positive values receipts. The cash flows must occur at regular intervals such as monthly or yearly.

Returns

Number value

Action

NPV returns a Number specifying the net present value of an investment based on a series of periodic cash flows (payments and receipts) and a discount rate.

Example

The following example is applicable to both Basic and Crystal syntax:

Suppose that someone offers to pays you $1000 after 1 year, $2000 after 2 years, $1500 after 3 years and $1200 after 4 years. If the discount rate (the time value of money) is 5 percent, the value of this offer to you today is:

Rem Basic syntax

formula = NPV (0.05, Array (1000, 2000, 1500, 1200))

//Crystal syntax

NPV (0.05, [1000, 2000, 1500, 1200])

The formula returns 5049.44 (rounded to 2 decimals). So this scheme is worth $5,049.44 to you today. This is less than the sum of the payments, which is $5700, since you have to wait for this money.

Comments

- This function is designed to work like the Visual Basic function of the same name.
- The NPV and IRR functions are related since NPV (IRR (values), values) = 0. That is, the internal rate of return of a sequence of cash flows is the discount rate for which that sequence of cash flows has a net present value of 0.

Pmt

Basic and Crystal syntax.

Overloads

- Pmt (rate, nPeriods, presentValue)
- Pmt (rate, nPeriods, presentValue, futureValue)
- Pmt (rate, nPeriods, presentValue, futureValue, type)

Arguments

- rate is a Number that specifies the interest rate per period.
- nPeriods is a positive Number that specifies the total number of payment periods in the annuity. The units used for specifying rate and nPeriods must consistent. For example, if nPeriods is the number of periods in months, then rate is a monthly interest rate.
- presentValue is a Number or Currency that specifies the present value or principal. That is, the amount that a series of payments in the future is worth now.
- futureValue is an optional Number or Currency that specifies the future value or cash balance you want after you've made the final payment. If omitted, 0 is assumed.
- type is an optional Number that specifies when payments are due. Specify 0 if payments are due at the end of the payment period, and 1 if payments are due at the beginning of the period. If omitted, 0 is assumed.

Returns

Number value

Action

Pmt returns a Number specifying the payment for an annuity based on periodic, fixed payments and a fixed interest rate.

Examples

The following examples are applicable to both Basic and Crystal syntax:

Suppose that you want to take out a $250,000 loan payable monthly over 15 years at an annual interest rate of 7 percent. The following formula returns your monthly loan payment. Note that the monthly interest rate is 0.07 / 12 and the number of months of the loan is 15 * 12.

```
Pmt (0.07 / 12, 15 * 12, 250000)
```

Returns the Number value 247.07 (rounded to 2 decimals). The value is negative because it represents a payment out from you whereas the loan amount of $250,000 is positive because it represents a payment in to you.

Now suppose that the payments are made at the beginning of the month instead of the end (as is the default). Your monthly loan payment is calculated as:

```
Pmt (0.07 / 12, 15 * 12, 250000, 0, 1)
```

Returns 234.04 (rounded to 2 decimals). Note that your monthly payment is about $13 less each month than in the previous example where payments are made at the end of the month.

Now suppose that you know that you'll receive $100,000 in 15 years so there is no need to fully pay off the loan; only to reduce the amount owed to $100,000 after 15 years. Note that the future value is negative since after 15 years you need to pay out $100,000 to clear the loan. Your monthly loan payment is calculated as:

```
Pmt (0.07 / 12, 15 * 12, 250000, 00000)
```

Returns 931.58 (rounded to 2 decimals).

Comment

- This function is designed to work like the Visual Basic function of the same name.
- To find the total amount paid out over the whole loan, multiply the payment per period (the value returned by Pmt) by the total number of periods (nPeriods).

PPmt

Basic and Crystal syntax.

Overloads

- PPmt (rate, period, nPeriods, presentValue)
- PPmt (rate, period, nPeriods, presentValue, futureValue)
- PPmt (rate, period, nPeriods, presentValue, futureValue, type)

Arguments

- rate is a Number that specifies the interest rate per period.
- period is a Number that specifies the payment period in the range 1 through nPeriods.
- nPeriods is a positive Number that specifies the total number of payment periods in the annuity. The units used for specifying rate, period and nPeriods must be consistent. For example, if nPeriods is the number of periods in months, then rate is a monthly interest rate and period specifies a month.
- presentValue is a Number or Currency that specifies the present value, or value today, of a series of future payments or receipts.

- futureValue is an optional Number or Currency that specifies the future value or cash balance you want after you've made the final payment. If omitted, 0 is assumed.
- type is an optional Number that specifies when payments are due. Specify 0 if payments are due at the end of the payment period, and 1 if payments are due at the beginning of the period. If omitted, 0 is assumed.

Returns

Number value

Action

PPmt returns a Number specifying the principal payment for a given period of an annuity based on periodic, fixed payments and a fixed interest rate.

Examples

The following examples are applicable to both Basic and Crystal syntax:

Suppose that you want to take out a $250,000 loan payable monthly over 15 years at an annual interest rate of 7 percent. The following formula returns the amount of principal that you pay in your first loan payment. Note that the monthly interest rate is 0.07 / 12 and the number of months of the loan is 15 * 12.

```
PPmt (0.07 / 12, 1, 15 * 12, 250000)
```

Returns the Number value 88.74 (rounded to 2 decimals). The value is negative because it represents a payment out from you whereas the loan amount of $250,000 is positive because it represents a payment in to you.

The following formula returns the amount of principal that you pay in your 121st payment (after 10 years of payments):

```
PPmt (0.07 / 12, 10*12 + 1, 15 * 12, 250000)
```

Returns 585.09 (rounded to 2 decimals). You've made progress on the loan and are paying off more of the principal per payment. This is because less interest can accrue each month since more of the loan is paid off so that your fixed monthly payment is applied more to the principal.

The following Basic syntax formula returns the amount of the loan that has been paid off after 10 years by summing up the principal payments made for the first 120 monthly payment:

Rem Basic syntax

Dim total, i

total = 0

For i = 1 To 10 * 12

total = total + PPmt (0.07 / 12, i, 15 * 12, 250000)

Next i

formula = total

Returns 36518.45 (rounded to 2 decimals). So you have paid off $136,518.45 of the loan and still owe $113,481.55 after the 120th payment.

Another way to calculate the amount you still owe after 10 years is to use 2 financial functions:

Rem Basic syntax

Dim payment

payment = Pmt (0.07 / 12, 15 * 12, 250000)

formula = FV (0.07 / 12, 10 * 12, payment, 250000)

Returns 13481.55 (rounded to 2 decimals). So you still owe $113, 481.55 after 10 years of payments.

Comment

This function is designed to work like the Visual Basic function of the same name.

PV

Basic and Crystal syntax.

Overloads

- PV (rate, nPeriods, payment)
- PV (rate, nPeriods, payment, futureValue)
- PV (rate, nPeriods, payment, futureValue, type)

Arguments

- rate is a Number that specifies the interest rate per period.
- nPeriods is a positive Number that specifies the total number of payment periods in the annuity. The units used for specifying rate and nPeriods must consistent. For example, if nPeriods is the number of periods in months, then rate is a monthly interest rate.
- payment is a Number or Currency that specifies payment to be made each period.
- futureValue is an optional Number or Currency that specifies the future value or cash balance you want after you've made the final payment. If omitted, 0 is assumed.
- type is an optional Number that specifies when payments are due. Specify 0 if payments are due at the end of the payment period, and 1 if payments are due at the beginning of the period. If omitted, 0 is assumed.

Returns

Number value

Action

PV returns a Number specifying the present value of an annuity based on periodic, fixed payments to be paid in the future and a fixed interest rate.

Example

The following example is applicable to both Basic and Crystal syntax:

Suppose that you want to buy a condo and can make payments of $1100 twice a month (24 annual payments). If the mortgage rates are 7 percent, and you want to pay off the condo in 10 years, what is the maximum loan that you can take out?

```
PV (0.07 / 24, 10 * 24, 100)
```

Returns 189668 (rounded to the nearest dollar). Thus you can afford a loan of about $190,000. Notice that the payment argument is negative since you are paying out the money each month.

Comment

This function is designed to work like the Visual Basic function of the same name.

Rate

Basic and Crystal syntax.

Overloads

- Rate (nPeriods, payment, presentValue)
- Rate (nPeriods, payment, presentValue, futureValue)
- Rate (nPeriods, payment, presentValue, futureValue, type)
- Rate (nPeriods, payment, presentValue, futureValue, type, guess)

Arguments

- nPeriods is a positive Number that specifies the total number of payment periods in the annuity.
- payment is a Number or Currency that specifies payment to be made each period.
- presentValue is a Number or Currency that specifies the present value, or value today, of a series of future payments or receipts.
- futureValue is an optional Number or Currency that specifies the future value or cash balance you want after you've made the final payment. If omitted, 0 is assumed.
- type is an optional Number that specifies when payments are due. Specify 0 if payments are due at the end of the payment period, and 1 if payments are due at the beginning of the period. If omitted, 0 is assumed.
- guess is an optional Number value that is estimated to be returned by Rate. If omitted, guess is 0.1 (10 percent).

Returns

Number value

Action

Rate returns a Number specifying the interest rate per period for an annuity. The units of the returned value are consistent with the units of nPeriods. For example, if nPeriods is in months, then the rate returned will be a monthly interest rate.

Examples

The following examples are applicable to both Basic and Crystal syntax:

An electronics store offers to finance a $1500 television for $70 per month, over 2 years, with no money down. Is this a good deal? A first step to determine this is to figure out what interest rate the store is charging:

```
Rate (2 * 12, 0, 1500)
```

Returns 0.00927 (rounded to 5 decimals). Notice that nPeriods is 24 months, payment (0) is negative since you are paying out the monthly payments, and the present value (1500) is positive since at the start of the loan, you effectively received $1500 (the value of the television). The interest rate returned is a monthly interest rate, since nPeriods was in months.

This next expression calculates the interest rate but expressed as a yearly interest rate and as a percent.

```
Rate (2 * 12, 0, 1500) * 12 * 100
```

Returns 11.1 (rounded to 1 decimal). Thus, the store is charging an effective annual interest rate of 11.1 percent.

Comment

- This function is designed to work like the Visual Basic function of the same name.
- There is no direct formula for the Rate function and so Crystal Reports calculates the value by iteration. The process depends on the initial guess for the rate. If the program reports an error, try changing the value of the guess argument to be closer to what you expect the interest rate should be.

SLN (cost, salvage, life)

Arguments

- cost is a Number or Currency that specifies the initial cost of the asset.
- salvage is a Number or Currency that specifies the value of the asset at the end of its useful life.
- life is a Number that specifies the length of the useful life of the asset. It must not equal 0.

Returns

Number value

Action

SLN returns a Number specifying the straight-line depreciation of an asset for a single period.

Example

The following examples is applicable to both Basic and Crystal syntax:

Suppose a company purchases a fleet of cars for $500,000. The cars have a lifetime of 10 years and a salvage value of $60,000. The depreciation per year is:

```
SLN (500000, 60000, 10)
```

Returns 44000. Thus the depreciation per year is $44,000.

Comment

This function is designed to work like the Visual Basic function of the same name.

SYD (cost, salvage, life, period)

Basic and Crystal syntax.

Arguments

- cost is a Number or Currency that specifies the initial cost of the asset.
- salvage is a Number or Currency that specifies the value of the asset at the end of its useful life.
- life is a positive Number that specifies the length of the useful life of the asset.
- period is a Number that specifies the period for which asset depreciation is calculated. The value is positive and less than or equal to life. The arguments life and period must have the same units.

Returns

Number value

Action

SYD returns a Number specifying the sum-of-years' digits depreciation of an asset for a single period.

Examples

The following examples are applicable to both Basic and Crystal syntax:

Suppose a company purchases a fleet of cars for $500,000. The cars have a lifetime of 10 years and a salvage value of $60,000. They are depreciated as follows:

```
SYD (500000, 60000, 10, 1)
```

Returns 80000. The first year's depreciation is $80,000.

```
SYD (500000, 60000, 10, 4)
```

Returns 56000. The fourth year's depreciation is $56,000.

```
SYD (500000, 60000, 10, 10)
```

Returns 8000. The final year's depreciation is $8,000.

Comment

This function is designed to work like the Visual Basic function of the same name.

String functions

String functions are used for the evaluation, manipulation and conversion of text strings.

Len (str)

Length (str)

Trim (str)

LTrim (str)

TrimLeft (str)

RTrim (str)

TrimRight (str)

Len (str)

Len and Length (str) are equivalent functions. However, Len is preferred in Basic syntax whereas Length is preferred in Crystal syntax.

Argument

str is a text String value

Returns

Whole number

Action

Len returns a Number indicating the length of the given string.

Note: Text strings must be enclosed in quotation marks. Any blank spaces are included as part of the character count.

Typical use

Use this function any time you have a manipulation, comparison, or calculation that is dependent on the length of a text string.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
Len("Account")
```

Returns 7.

```
Len("Text String")
```

Returns 11. The blank space between "Text" and "String" is counted as a character.

```
Len(" Center ")
```

Returns 10. There are 2 spaces on either side of the word "Center".

```
Len(ToText({orders.ORDER AMOUNT}))
```

Returns 9, where {orders.ORDER AMOUNT} is 14233.08, and ToText ({orders.ORDER AMOUNT}) returns the String "14,233.08". Here, the comma and period have been specified in the Number format for the formula field as the thousands and decimal separators respectively.

```
Len(ToText({orders.ORDER AMOUNT}, 2, "", "."))
```

Returns 8 where {orders.ORDER AMOUNT} = 14233.08, and ToText ({orders.ORDER AMOUNT}, 2, "", ".") returns the String "14233.08". Here, the null character and period have been specified in the Number format for the formula field as the thousands and decimal separators respectively.

```
Len({customer.CUSTOMER NAME})
```

The length of the text string stored as a value in {customer.CUSTOMER NAME}.

The following examples are applicable to Basic syntax:

```
{customer.CUSTOMER NAME} (Len({customer.CUSTOMER NAME}))
```

Returns the third character from the right in the Customer Name. This can also be used to extract a substring from a String that always starts at a fixed position from the end of the String.

In the following example, an address line contains the province name and postal code. Full province names are of different lengths; however, a postal code has a fixed number of characters: the first character of the postal code is always the seventh from the last character in the String. The formula determines the length of the address line, subtracts 7 from it, and uses the result to identify the first character of the postal code in the address line.

Dim addressLine As String

Dim addressLen As Number

addressLine = "British Columbia V6X 3W2"

addressLen = Len (addressLine)

formula = addressLine (addressLen+1 To addressLen)

Returns the String "V6X 3W2".

Comments

This function is designed to work like the Visual Basic function of the same name.

Length (str)

Length and Len (str) are equivalent functions. However, Length is preferred in Crystal syntax whereas Len is preferred in Basic syntax.

Arguments

str is a text value.

Returns

Whole Number

Action

Length returns the number of characters in a text string that you enter into the formula, or in a text string stored as a value in a data field.

Note: Text strings must be enclosed in either double or single quotation marks (" " or ` '). Any blank spaces are included as part of the character count.

Typical uses

Use this function any time you have a manipulation, comparison, or calculation that is dependent on the length of a text string.

Examples

The following examples are applicable to both Basic and Crystal syntax, although Len is preferred in the latter:

```
Length("Account")
```

Returns 7.

```
Length("Text_String")
```

Returns 11.

```
Length("__Center__")
```

Returns 10.

```
Length(ToText({orders.ORDER AMOUNT}))
```

Returns 9, where {orders.ORDER AMOUNT} is 14233.08, and ToText ({orders.ORDER AMOUNT}) returns the String "14,233.08". Here, the comma and period have been specified in the Number format for the formula field as the thousands and decimal separators respectively,

```
Length(ToText({orders.ORDER AMOUNT}, 2, "", "."))
```

Returns 8 where {orders.ORDER AMOUNT} = 14233.08, and ToText ({orders.ORDER AMOUNT}, 2, "", ".") returns the String "14233.08". Here, the null character and period have been specified in the Number format for the formula field as the thousands and decimal separators respectively.

```
Length("BOB")
```

Returns 3.

```
Length("SMITH")
```

Returns 5.

```
Length("BOB SMITH")
```

Returns 9. (The blank space between BOB and SMITH is counted as a character.)

```
Length({customer.CUSTOMER NAME})
```

The length of the text string stored as a value in {customer.CUSTOMER NAME}.

The following examples are applicable to Crystal syntax.

```
{customer.CUSTOMER NAME}[Length({customer.CUSTOMER NAME})]
```

Returns the third character from the right in the Customer Name. This can also be used to extract a substring from a String that always starts at a fixed position from the end of the String.

In the following example, an address line contains the province name and postal code. Full province names are of different lengths, but the first character of the postal code is always the seventh from the last character. The formula determines the length of the address line, subtracts 7 from it, and uses the result to identify the first character of the postal code in the address line.

Local StringVar addressLine;

Local NumberVar addrLength;

addressLine := "British Columbia V6X 3W2";

addrLength := Length (addressLine);

addressLine [addrLength+1 to addrLength]

Related topics
Formula 14

Trim (str)

Basic and Crystal syntax.

Arguments
str is a text string to be trimmed.

Returns
Text String

Action
The Trim function removes leading and trailing spaces from string arguments.

Typical uses
Use this function any time there are leading and/or trailing blanks in a text object that may interfere with an alignment of text strings, a character count, or with a calculation (if the string is eventually converted to a Number).

Examples
The following examples are applicable to both Basic and Crystal syntax:

```
Trim(" abcde ")
```

Returns "abcde".

Comments
This function is designed to work like the Visual Basic function of the same name.

LTrim (str)

LTrim and TrimLeft (str) are equivalent functions. However, LTrim is preferred in Basic syntax, whereas TrimLeft is preferred in Crystal syntax.

Argument
str is a text String value

Returns
String value.

Action
LTrim removes all spaces stored to the left of the given string and returns it.

Typical uses

- Use this function any time there are leading blanks in a text object that may interfere with an alignment of text strings, a character count, or with a calculation (if the string is eventually converted to a Number).
- Use whenever you combine (concatenate) justified text strings and want to have the proper spacing between strings.

Examples
The following examples are applicable to both Basic and Crystal syntax:

```
LTrim(" A1/4520/B12")
```

Returns "A1/4520/B12".

```
LTrim(" 200")
```

Returns "200", where the text string " 200" is right-justified with leading blanks.

You have two right-justified database fields, {file.FOOD1} and {file.FOOD2}. Each field can hold up to 15 characters. {file.FOOD1} contains the word bread and {file.FOOD2} contains the word butter. If you were to print these words, they would appear like this:

" bread"

" butter"

For each 15haracter field the database includes the field value, right-justified plus enough blank spaces to fill up the field. To use these words without the leading spaces (to create the expression bread and butter, for example) use the LTrim function in the following manner:

```
LTrim({file.FOOD1}) + " and " + LTrim({file.FOOD2})
```

Returns "bread and butter".

The spaces enclosed in the quotation marks before and after the word and assure that there is the correct spacing between the three words in the resulting sentence.

Comments

This function is designed to work like the Visual Basic function of the same name.

Related topics

Trim (str)

RTrim (str)

TrimLeft (str)

TrimLeft and LTrim (str) are equivalent functions. However, TrimLeft is preferred in Crystal syntax, whereas LTrim is preferred in Basic syntax.

Arguments

str is a string.

Returns

Text String

Action

TrimLeft removes all spaces to the left of a string or data field which is stored as a right-justified string in a database.

Typical uses

- Use this function any time there are leading blanks in a text object that may interfere with an alignment of text strings, a character count, or with a calculation (if the string is eventually converted to a Number).
- Use whenever you combine (concatenate) justified text strings and want to have the proper spacing between strings.

Examples

The following examples are applicable to both Basic and Crystal syntax, although LTrim is preferred in the latter:

```
TrimLeft(" A1/4520/B12")
```

Returns "A1/4520/B12".

```
TrimLeft(" 200")
```

Returns "200", where the text string "200" is right-justified with leading blanks.

You have two right-justified database fields, {file.FOOD1} and {file.FOOD2}. Each field can hold up to 15 characters. {file.FOOD1} contains the word bread and {file.FOOD2} contains the word butter. If you were to print these words, they would appear like this:

" bread"

" butter"

For each 15haracter field the database includes the field value, right-justified plus enough blank spaces to fill up the field. To use these words without the leading spaces (to create the expression bread and butter, for example) use the TrimLeft function in the following manner:

```
TrimLeft({file.FOOD1}) + " and " + TrimLeft({file.FOOD2})
```

Returns "bread and butter".

The spaces enclosed in the quotation marks before and after the word and assure that there is the correct spacing between the three words in the resulting sentence.

Related topics

Trim (str)

TrimRight (str)

Formula 14

RTrim (str)

RTrim and TrimRight (str) are equivalent functions. However, RTrim is preferred in Basic syntax, whereas TrimRight is preferred in Crystal syntax.

Argument

str is a text String value

Returns

String value.

Action

RTrim removes all spaces to the right of the given string and returns it.

Typical uses

- Use this function any time there are trailing blanks in a text object that may interfere with an alignment of text strings, a character count, or with a calculation (if the string is eventually converted to a Number).
- Use whenever you combine (concatenate) justified text strings and want to have the proper spacing between strings.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
RTrim("Al/4520/B12 ")
```

Returns "A1/4520/B12".

```
RTrim({file.REFERENCE})
```

Returns "Bal Fwd." where {file.REFERENCE} contains the left-justified text string "Bal Fwd. ".

Comments

This function is designed to work like the Visual Basic function of the same name.

Related topics

Trim (str)

LTrim (str)

TrimRight (str)

TrimRight and RTrim (str) are equivalent functions. However, TrimRight is preferred in Crystal syntax, whereas RTrim is preferred in Basic syntax.

Arguments

TrimRight(str)

Where str is a text string.

Returns

Text String

Action

TrimRight removes all spaces to the right of a string or data field which is stored left-justified in a database.

Typical uses

- Use this function any time there are trailing blanks in a text object that may interfere with an alignment of text strings, a character count, or with a calculation (if the string is eventually converted to a Number).
- Use whenever you combine (concatenate) justified text strings and want to have the proper spacing between strings.

Examples

The following examples are applicable to both Basic and Crystal syntax, although RTrim is preferred in the latter:

```
TrimRight("Al/4520/B12 ")
```

Returns "A1/4520/B12".

```
TrimRight({file.REFERENCE})
```

Returns "Bal Fwd." where the text string "Bal Fwd. " is left-justified in the field.

Related topics

Trim (str)

TrimLeft (str)

UCase (str)

UCase and UpperCase (str) are equivalent functions. However, UCase is preferred in Basic syntax, whereas UpperCase is preferred in Crystal syntax.

Argument

str is a text String value

Returns

String value.

Action

UCase converts the given string to all upper case and returns it.

Typical use

A good use of this function is when a field contains both uppercase and lowercase letters and you want to convert all letters to uppercase for consistency.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
UCase("Description")
```

Returns "DESCRIPTION".

```
UCase({customer.FIRST NAME})
```

Returns "RONALD", where {customer.FIRST NAME} contains "Ronald".

```
UCase("Ronald Black")
```

Returns "RONALD BLACK".

```
UCase("abc12345")
```

Returns "ABC12345".

```
UCase("BrEaD " + "AND " + "bUtTeR")
```

Returns "BREAD AND BUTTER".

Comments

This function is designed to work like the Visual Basic function of the same name.

Related topics

LCase (str)

UpperCase (str)

UpperCase and UCase (str) are equivalent functions. UpperCase is preferred in Crystal syntax, whereas UCase is preferred in Basic syntax.

Arguments

str is a text string.

Returns

Text String

Action

UpperCase returns the text string or text value in the data field in upper case (capital letters).

Typical uses

A good use of this function is when a field contains both uppercase and lowercase letters and you want to convert all letters to uppercase for consistency.

Examples

The following examples are applicable to both Basic and Crystal syntax, although UCase is preferred in the latter:

```
UpperCase("Description")
```

Returns "DESCRIPTION".

```
UpperCase({customer.FIRST NAME})
```

Returns "RONALD", where {customer.FIRST NAME} contains "Ronald".

```
UpperCase("Ronald Black")
```

Returns "RONALD BLACK".

```
UpperCase("abc12345")
```

Returns "ABC12345".

```
UpperCase("BrEaD " + "AND " + "bUtTeR" )
```

Returns "BREAD AND BUTTER".

Related topics

LowerCase (str)

LCase (str)

LCase and LowerCase (str) are equivalent functions. However, LCase is preferred in Basic syntax, whereas LowerCase is preferred in Crystal syntax.

Argument

str is a text value

Returns

String value.

Action

LCase converts the given string to all lower case and returns it.

Typical use

A good use of this function is when a field contains both uppercase and lowercase letters and you want to convert all letters to lowercase for consistency.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
LCase ("Description")
```

Returns "description".

```
LCase ({customer.FIRST NAME})
```

Returns "ronald", where {customer.FIRST NAME} is "Ronald".

```
LCase ("Ronald Black")
```

Returns "ronald black".

```
LCase ("ABC12345")
```

Returns "abc12345".

```
LCase ("BrEaD " + "AND " + "bUtTeR")
```

Returns "bread and butter".

Comments

- This function is designed to work like the Visual Basic function of the same name.
- Numbers that are part of the text are not affected by the LCase function.

Related topics

UCase (str)

LowerCase (str)

LowerCase and LCase (str) are equivalent functions. However, LowerCase is preferred in Crystal syntax, whereas LCase is preferred in Basic syntax.

Arguments

x is a text value.

Returns

Text String

Action

LowerCase returns the given text value in all lower case letters.

Typical uses

A good use of this function is when a field contains both uppercase and lowercase letters and you want to convert all letters to lowercase for consistency.

Examples

The following examples are applicable to both Basic and Crystal syntax, although LCase is preferred in the latter:

```
LowerCase("Description")
```

Returns "description".

```
LowerCase({customer.FIRST NAME})
```

Returns "ronald", where {customer.FIRST NAME} is "Ronald".

```
LowerCase("Ronald Black")
```

Returns "ronald black".

```
LowerCase("ABC12345")
```

Returns "abc12345".

```
LowerCase("BrEaD " + "AND " + "bUtTeR")
```

Returns "bread and butter".

Comments

Numbers that are part of the text are not affected by the LowerCase function.

Related topics

UpperCase (str)

StrReverse (inputString)

Basic and Crystal syntax.

Argument

inputString is a String whose characters are to be reversed.

Returns

String value.

Action

StrReverse returns a String in which the character order of inputString is reversed. If inputString is a zero-length string (""), a zero-length string is returned.

Examples

The following example is applicable to both Basic and Crystal syntax:

```
StrReverse("abc")
```

Returns "cba".

Comments

This function is designed to work like the Visual Basic function of the same name.

IsNumeric (str)

IsNumeric and NumericText (str) are equivalent functions. However, IsNumeric is preferred in Basic syntax, whereas NumericText is preferred in Crystal syntax.

Argument

str is a String value to be tested.

Returns

Boolean value (True or False).

Action

IsNumeric returns True if the String argument can be converted to a Number and False otherwise.

Typical use

If you store numbers (like weight) in a text String object, use IsNumeric to check the value of each record to make sure it is OK to convert using ToNumber or CDbl.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
IsNumeric({file.REFERENCE})
```

Returns FALSE where {file.REFERENCE} = "ABCDEFG".

```
IsNumeric({file.IDNUM})
```

Returns TRUE where {file.IDNUM} = "12345".

```
IsNumeric({file.IDNUM})
```

Returns TRUE where {file.IDNUM} = "12345443".

```
IsNumeric({file.IDNUM})
```

Returns FALSE where {file.IDNUM} = "1234543".

```
IsNumeric ({file.IDNUM} (1 to 5))
```

Returns TRUE where {file.IDNUM} = "1234543".

```
IsNumeric ({file.IDNUM} (6))
```

Returns FALSE where {file.IDNUM} = "1234543".

```
IsNumeric ({file.IDNUM} (7 to 9))
```

Returns TRUE where {file.IDNUM} = "1234543".

```
IsNumeric({file.IDNUM})
```

Returns FALSE where {file.IDNUM} = "12345T".

Note: You can use this function in combination with ToNumber (numeric), ToNumber (string), ToNumber (Boolean) or CStr to test for a Number in the Reference field, then return the string as a Number or return 0 if the text string is not a Number.

The following example is applicable to Basic syntax:

If IsNumeric({file.IDNUM}) Then

formula = ToNumber({file.IDNUM})

Else

formula = 0

End If

NumericText (str)

NumericText and IsNumeric (str) are equivalent functions. However, NumericText is preferred in Crystal syntax, whereas IsNumeric is preferred in Basic syntax.

Arguments

str is a text string being tested for numeric text.

Returns

Boolean Value

Action

NumericText tests to see if the content of a text object is a Number.

- If the entire content of the field is a Number - or if the characters extracted via the Subscript for arrays and strings are entirely a Number - the function returns a TRUE value.
- If any part of the content of the field - or of the characters extracted - are not a Number, the function returns the value FALSE.

Typical uses

If you store numbers (like weight) in a text object, use NumericText to check the value of each record to make sure it is OK to convert using ToNumber (numeric), ToNumber (string), ToNumber (Boolean).

Examples

The following examples are applicable to both Basic and Crystal syntax, although IsNumeric is preferred in the latter:

```
NumericText({file.REFERENCE})
```

Returns FALSE where {file.REFERENCE} = "ABCDEFG".

```
NumericText({file.IDNUM})
```

Returns TRUE where {file.IDNUM} = "12345".

```
NumericText({file.IDNUM})
```

Returns TRUE where {file.IDNUM} = "12345443".

```
NumericText({file.IDNUM})
```

Returns FALSE where {file.IDNUM} = "1234543".

```
NumericText ({file.IDNUM} [1 to 5])
```

Returns TRUE where {file.IDNUM} = "1234543".

```
NumericText ({file.IDNUM} [6])
```

Returns FALSE where {file.IDNUM} = "1234543".

```
NumericText ({file.IDNUM} [7 to 9])
```

Returns TRUE where {file.IDNUM} = "1234543".

```
NumericText({file.IDNUM})
```

Returns FALSE where {file.IDNUM} = "12345T".

Note: You can use this function in combination with ToNumber (numeric), ToNumber (string), ToNumber (Boolean) to test for a Number in the Reference field, then print the string as a Number or print 0 if the text string is not a Number.

The following example is applicable to Crystal syntax:

If NumericText({file.REFERENCE}) Then

ToNumber({file.REFERENCE})

Else

0;

Related topics
Formula 9

ToNumber (numeric), ToNumber (string), ToNumber (Boolean)

Basic and Crystal syntax.

ToNumber and CDbl are equivalent functions.

Arguments

- numeric is a Number or a Currency type value.
- string is a text string that holds numeric text.
- Boolean is a Boolean value that you want to treat as a number.

Returns

Fractional Number

Action

ToNumber converts a Number, Currency, text string, or Boolean value to a Number.

In a database, some numbers are stored in numeric fields, as Numbers, and some are stored in character fields, as text. You make the determination which fields are to be numeric and which are to be text when you set up the database in the first place. Numbers on which you might wish to perform arithmetic (item cost, quantity ordered, etc.) are typically stored in numeric fields; numbers on which you do not expect to perform arithmetic (customer number, telephone number, etc.) are typically stored in text objects.

ToNumber allows you to convert a Number stored as text to a Number on which you can perform arithmetic.

The Boolean argument enables you to treat SQL_BIT data values as if they were numbers.

Typical uses

You might use this function, for example, if your item numbers contain coded product information and you want to use that information in calculations. Typically, you would be using this function to convert a Currency or String value that contains all numeric characters to a Number type value.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
ToNumber({orders.ORDER AMOUNT})
```

Returns a Number type value that the Currency field {orders.ORDER AMOUNT} contains.

```
ToNumber("123.45")
```

Returns 123.45.

```
ToNumber({file.REFERENCE})
```

Returns 200.00 where "200" is the text string in the {file.REFERENCE} field.

```
ToNumber({file.ACCTNO}/2)
```

Returns 22144 where {file.ACCTNO} = 44288.

```
ToNumber({customers.POSTAL CODE}) < 33333
```

Returns TRUE where {customer.POSTAL CODE} is "21385".

Comments

When using the ToNumber function, you should first test the value with the NumericText function. NumericText returns a value of TRUE only if the value in the string can be correctly converted to a Number. Otherwise, if you try to convert a value to a Number that is not a Number, the formula will produce an error in your report.

The following example is a common use of the NumericText and ToNumber functions together, in Crystal syntax:

If NumericText ({file.FIELD}) Then

ToNumber ({file.FIELD})

Else

0

Related topics

Formula 2

Formula 3

Formula 9

Formula 17

ToWords (x), ToWords (x, #places)

Basic and Crystal syntax.

Arguments

- x is a fractional Number to be converted into words.
- #places is a whole number indicating the number of decimal places to be converted. (This argument is optional.)

Returns

Text String

Action

You can use this function to convert a Number or Currency field value or the result of a numeric calculation to words so it can be used as text. The ability to adjust the number of decimal places can be useful when the number is the result of a calculation that may produce more decimal places than you want.

Typical uses

You can use this function to spell out the dollar amount for each check if you are using computer checks.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
ToWords(12345)
```

Returns twelve thousand three hundred forty-five and xx/100.

```
ToWords(12345.6749,2)
```

Returns twelve thousand three hundred forty-five and 67/100.

```
ToWords(12345.4999,0)
```

Returns twelve thousand three hundred forty-five.

```
ToWords(12345.5000,0)
```

Returns twelve thousand three hundred forty six.

```
ToWords(2345)
```

Returns negative twelve thousand three hundred forty-five and xx/100.

```
ToWords(12.3499)
```

Returns twelve and 35/100.

```
ToWords({orders.ORDER AMOUNT})
```

Returns two thousand sixteen and 84/100 where Amount = 2016.84.

```
ToWords((({file.QTY1} + {file.QTY2} + {file.QTY3}) * {file.PRICE}) *
1.075)
```

Returns one hundred two and 13/100.

```
ToWords((({file.QTY1} + {file.QTY2} + {file.QTY3}) * {file.PRICE}) *
1.075,0)
```

Returns one hundred two where QTY1 = 1, QTY2 = 82, QTY3 = 12, and Price = 1.00 (sums three quantities, multiplies them times the price and adds 7.5% sales tax). In this case the numeric answer is 102.125 which is then rounded to 102.13 (the standard two decimal places) before putting into words.

Comments

- The function treats the number as a whole, rather than as a series of individual numbers. That is, 123 is treated as the number one hundred twenty-three rather than the individual digits one, two, and three.
- Negative numbers begin with the word "negative."
- Currency field values and Number field values are treated in the same way and produce identical results.
- Since the spelled out value will be much longer than the Number value, you will need to increase the size of the field box to accommodate the new field length.

ReplicateString (str, #copies)

Basic and Crystal syntax.

Arguments

- str is the text string to be replicated.
- #copies is a whole number indicating the number of times str is to be replicated.

Returns

Text String

Action

Replicates the string in str the number of times specified by #copies.

Typical uses

You can use this function to insert a line of characters any time they are needed. Some typical uses are:

- to flag critical data on your report
- to build simple bar graphs

- to split your report into visible sections
- to highlight totals, subtotals, and other summary data.

Examples

The following examples are applicable to Crystal syntax:

If {file.SALES} < {file.QUOTA} Then

ReplicateString("*",10)

Else

""

Prints the character * ten times as a flag.

```
{file.NAME}+ " " + ReplicateString("*", {file.SCORE})
```

Prints a simple bar graph showing test results. The formula prints an asterisk for each point in a test score (the value in the file.SCORE field).

The formula prints an asterisk for each point in a test score (the value in the {file.SCORE} field). For example, given the following data:

Name	Score
Smith	23
Jones	17
Brown	21
Johnson	13
Reynolds	25

The formula will produce the following results:

Name	@Score
Smith	***********************
Jones	******************
Brown	**********************
Johnson	***************
Reynolds	*****************************

Space (x)

Basic and Crystal syntax.

Arguments

x is a whole number indicating the number of spaces.

Returns
Text String (one or more spaces)

Action
The Space function returns a specified number of spaces.

Typical uses
Use the Space function to easily add a specific number of spaces to a text string.

Examples
The following examples are applicable to both Basic and Crystal syntax:

```
Space(2)
```

Returns " ".

Comments
This function is designed to work like the Visual Basic function of the same name.

InStr (str1, str2), InStr (start, str1, str2)

Basic and Crystal syntax.

Arguments

- start is the character in str1 where the search is to begin. This is a 1 based index. (This argument is optional.)
- str1 is the text string to be searched.
- str2 is the text string being sought.

Returns
Whole Number

Action
The InStr function returns the position of the first occurrence of one string within another. This position is a 1 based index of the characters in str1. If str2 is not found in str1, the InStr function returns 0. The start argument sets the starting position for the search.

Typical uses
Use this function to determine if one string contains another.

Examples
The following examples are applicable to both Basic and Crystal syntax:

```
InStr("abcdefg", "bcd")
```

Returns 2.

```
InStr(3, "abcdefg", "cde")
```

Returns 3.

Comments

- This function is designed to work like the Visual Basic function of the same name.
- There are two versions of this function. The first does not use the start argument, the second does. If the start argument is not used, InStr searches all of str1 to try to find str2. If the start argument is used, InStr begins searching at the character in str2 indicated by the start argument.

InStrRev

Basic and Crystal syntax.

Overloads

- InStrRev (inputString, findString)
- InStrRev (inputString, findString, startPosition)
- InStrRev (inputString, findString, startPosition, compare)

Arguments

- inputString is a String expression being searched.
- findString is a String expression being searched for.
- startPosition is an optional numeric expression that sets the starting position for each search. If omitted, is used, which means that the search begins at the last character position.
- compare is an optional numeric value indicating the kind of comparison to use when evaluating substrings:
- 0 performs a comparison that is case-sensitive
- 1 performs a comparison that is case-insensitive

If omitted, a case-sensitive comparison is performed.

Returns

Number value that indicates the position of the matching string in the string to be matched. The first character of the inputString (from the beginning) is 1.

Action

InStrRev returns the position of the first occurrence of one string within another. The matching proceeds in the reverse direction from the specified start position.

Examples

The following examples are applicable to Basic and Crystal syntax:

```
InStrRev ("abcdefgbchijk", "bc")
```

Returns 8, where matching starts from the very end of the string in the reverse direction.

```
InStrRev ("abcdefgbchijk", "bc", )
```

Returns 8, where matching starts from the very end of the string in the reverse direction.

```
InStrRev ("abcdefgbchijk", "bc", 2)
```

Returns 0, i.e., no match is found. Matching starts from the "b" in "ab" in the reverse direction.

```
InStrRev ("abcdefgbchijk", "bc", 3)
```

Returns 2. Matching starts from the "c" in "abc" in the reverse direction.

```
InStrRev ("abcdefgbchijk", "bc", 10)
```

Returns 8. Matching starts from the 'h' in "abcdefgbch" in the reverse direction, and the first occurrence of "bc" is found at position 8 from the beginning of the input string.

```
InStrRev ("abcdefgbchijk", "BC", , 1)
```

Returns 8. Matching starts from the very end of the string in the reverse direction, and the first occurrence of "BC", independent of case since case-insensitive comparison is specified, is found at position 8 from the beginning of the input string.

```
InStrRev ("aBCdefgbchijk", "BC", , 0)
```

Returns 2. Matching starts from the very end of the string in the reverse direction, and the first occurrence of "BC" is found at position 2 from the beginning of the input string.

Comment

This function is designed to work like the Visual Basic function of the same name.

StrCmp (str1, str2), StrCmp (str1, str2, compare)

Basic and Crystal syntax.

Arguments

- str1 is the first text string to be compared.
- str2 is the second text string to be compared.
- compare is an optional Number value indicating the kind of string comparison to use:
- 0 performs a comparison that is case-sensitive
- 1 performs a comparison that is case-insensitive

If omitted, a case-sensitive comparison is performed.

Returns

Whole Number where

- = Less Than
- 0 = Equal To
- 1 = Greater Than

Action

The StrCmp function compares two strings.

Typical Uses

Use this function to determine if two string values are identical.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
StrCmp("abcd", "aa")
```

Returns = 1.

```
StrCmp("abcd", "aa", 1)
```

Returns = 1.

```
StrCmp ("abcd", "ac")
```

Returns = .

```
StrCmp ("aa", "aa")
```

Returns = 0.

Comments

This function is designed to work like the Visual Basic function of the same name.

Mid (str, start), Mid (str, start, length)

Basic and Crystal syntax.

Arguments

- str is the text string from which one or more characters is being extracted.
- start is a whole number indicating the position of the first character to extract.
- length is a whole number indicating the number of characters to be extracted from the text string. This argument is optional. If a value is not specified, the rest of the string from the starting position is extracted.

Returns

Text String

Action

The Mid function returns a specified number of characters from a string. The second argument is the character position where the part to be taken begins. The third argument is the length of the string you want to be taken out. If the third argument is not specified, everything from the start position to the end of the string is extracted.

Typical uses

Use this function anytime you need to extract a set of characters from somewhere in the middle of a text string.

Examples

The following example is applicable to both Basic and Crystal syntax:

```
Mid("abcdef", 3, 2)
```

Returns "cd".

Comments
This function if designed to work like the Visual Basic function of the same name.

Left (str, length)

Basic and Crystal syntax.

Arguments

- str is a text string value.
- length is a whole number indicating the number of characters to be extracted from str.

Returns
Text String

Action
Extracts the specified number of characters from the left side of the string.

Typical uses
Use this function to obtain a certain number of characters from the left end of a text string. For instance, you could use the Left function to obtain just the area code from the values in a field containing phone numbers.

Examples
The following examples are applicable to Crystal syntax:

```
Left("abcdefg", 4)
```

Returns "abcd".

```
Left({customer.FAX}, 3)
```

Retrieves the area code of each fax number when the first three characters of the Fax field contains the area code.

Comments
This function is designed to work like the Visual Basic function of the same name.

Right (str, length)

Basic and Crystal syntax.

Arguments

- str is a text string.
- Length is a whole number indicating the number of characters to be extracted from str.

Returns
Text String

Action
Extracts the given number of text characters from the right side of the specified string.

Typical uses
Use this function to obtain just the right part of a string. For instance, use it to obtain the last four digits of social security numbers stored in a field.

Examples
The following examples are applicable to both Basic and Crystal syntax:

```
Right("abcdefg", 3)
```

Returns "efg".

```
Right({table.SSNUM}, 4)
```

Returns the last 4 digits of the social security number as a String.

Comments
This function is designed to work like the Visual Basic function of the same name.

Val (str)

Basic and Crystal syntax.

Arguments

- str is a text string.

Returns
Fractional Number

Action
The Val (str) function reads a string containing Numbers (example: an address, phone number, or social security number) and converts them to a decimal value. Val stops reading the string when it finds the first character in the string that it finds that it can not recognize as a number or as a space.

Typical uses
Use this function to extract only the numeric portion of a string that contains both numbers and text. For instance, you can extract only the street number from an address, or only the numeric portion of an ID that contains letters and numbers.

Examples
The following example is applicable to both Basic and Crystal syntax:

```
Val("2234 100th Street")
```

Returns 2234100.

Comments
This function is designed to work like the Visual Basic function of the same name.

Chr (x)

Basic and Crystal syntax.

Arguments
x is a whole number, specifically, it is any ASCII value.

Returns
Text String (one character long)

Action
The Chr (x) function returns the single character text string associated with the ASCII value passed as x.

Typical uses
Use this function any time you need to obtain the character associated with a particular ASCII value. This function is most useful for application developers using the Report Engine.

Examples
The following example is applicable to both Basic and Crystal syntax:

```
Chr(65)
```

Returns "A".

Comments
This function is designed to work like the Visual Basic function of the same name.

Asc (str)

Basic and Crystal syntax.

Arguments
str is a text string.

Returns
Whole Number (the ASCII value for the first character in the string).

Action
The Asc (str) function returns the ASCII code of the first character of the string.

Typical uses
Use this function anytime you need to obtain the ASCII value of a character. This function is most useful for application developers using the Report Engine.

Examples
The following example is applicable to both Basic and Crystal syntax:

```
Asc("A")
```

Returns 65.

Comments
This function is designed to work like the Visual Basic function of the same name.

Filter

Basic and Crystal syntax.

Overloads

- Filter (inputStrings, searchString)
- Filter (inputStrings, searchString, include)
- Filter (inputStrings, searchString, include, compare)

Arguments

- InputStrings is an array of strings to be searched.
- searchString is a string to search for.
- include is an optional Boolean value indicating whether to return substrings that include or exclude searchString. If include is True, Filter returns the subset of the array that contains searchString as a substring. If include is False, Filter returns the subset of the array that does not contain searchString as a substring. If omitted, the value True is assumed.
- compare is an optional Number value indicating the kind of string comparison to use:
- 0 performs a comparison that is case-sensitive
- 1 performs a comparison that is case-insensitive

If omitted, a case-sensitive comparison is performed.

Returns
Array of String values.

Action
Filter searches an array of strings for a specified string, and returns the strings in an array.

Examples
The following examples are applicable to Basic and Crystal syntax.

The examples assume that inputStrings is a String array consisting of the 5 elements: "pineapple", "Apple", "APPLE", "grape", and "orange".

```
Filter (inputStrings, "ap")
```

Returns an array containing the 2 strings "pineapple" and "grape".

```
Filter (inputStrings, "Ap", True)
```

Returns an array containing the 1 string "Apple".

```
Filter (inputStrings, "ap", True, 1)
```

Returns an array containing the 4 strings "pineapple", "Apple", "APPLE" and "grape". Matching to the search string uses case-insensitive comparison.

```
Filter (inputStrings, "ap", False)
```

Returns an array containing the 3 strings "Apple", "APPLE" and "orange".

```
Filter (inputStrings, "cd")
```

Returns an array containing 1 string, the empty string "". The returned array does not contain any matching strings but contains 1 element since the formula language does not support empty arrays.

Comments

This function is designed to work like the Visual Basic function of the same name.

Replace

Basic and Crystal syntax.

Overloads

- Replace (inputString, findString, replaceString)
- Replace (inputString, findString, replaceString, startPosition)
- Replace (inputString, findString, replaceString, startPosition, count)
- Replace (inputString, findString, replaceString, startPosition, count, compare)

Arguments

- inputString is a String containing substring to replace.
- findString is a substring being searched for.
- replaceString is the replacement substring.
- startPosition is an optional Number indicating the position within inputString where substring search is to begin. If omitted, 1 is assumed.
- count is an optional Number of substring substitutions to perform. If omitted, the default value is , which means make all possible substitutions.
- compare is an optional Number indicating the kind of comparison to use when evaluating substrings:
- 0 performs a comparison that is case-sensitive
- 1 performs a comparison that is case-insensitive

If omitted, a case-sensitive comparison is performed.

Returns

String value.

Action

Replace returns a String in which a specified substring has been replaced with another substring a specified number of times. As an option, you can also specify where in the String to begin replacing, and return a String starting from that position.

Typical use

To systematically replace in a String one substring with another.

Examples

The following examples are applicable to both Basic and Crystal syntax. They assume that inputString is the text String "Monday is my day off. Can we meet next Monday?".

```
Replace (inputString, "Monday", "Wednesday")
```

Returns "Wednesday is my day off. Can we meet next Wednesday?".

```
Replace (inputString, "Monday", "Wednesday", 23)
```

Returns "Can we meet next Wednesday?" The replacement begins at the 23rd character from the beginning of the original inputString. The returned string is this portion of the string with the substitutions made.

```
Replace (inputString, "Monday", "Wednesday", 1, 1)
```

Returns "Wednesday is my day off. Can we meet next Monday?" There is only 1 replacement beginning at the 1st character from the beginning of the original inputString. The returned string is thus the original string with the single substitution made.

```
Replace (inputString, "monday", "Wednesday", 1, , 0)
```

Returns the same inputString with no replacements since no match was found for "monday" with a case-sensitive comparison.

```
Replace (inputString, "monday", "Wednesday", 1, , 1)
```

Returns the inputString with both occurrences of "Monday" replaced by "Wednesday". That is, "Wednesday is my day off. Can we meet next Wednesday?".

Comments

- This function is designed to work like the Visual Basic function of the same name.
- The returned value of the Replace function is a String, with the specified substring replacements made, that begins at the position specified by startPosition and ends at the end of the inputString string. It is not a copy of the original string from start to finish.

Join

Basic and Crystal syntax.

Overloads

- Join (list)
- Join (list, delimiter)

Arguments

- list is a String array containing substrings to be joined.
- delimiter is an optional String used to separate the substrings in the returned string. If omitted, the space character (" ") is used. If delimiter is a zero-length string (""), all items in the list are concatenated with no delimiters.

Returns

String value.

Action
Join returns a String created by joining a number of substrings contained in an array.

Typical use
To convert elements stored in a String array to a single String.

Examples
The following examples are applicable to Basic and Crystal syntax.

The examples assume that list is a String array consisting of the 3 elements: "Chocolate", "Vanilla" and "Strawberry".

```
Join (list)
```

Returns the String "Chocolate Vanilla Strawberry".

```
Join (list, "***")
```

Returns the String "Chocolate***Vanilla***Strawberry".

Comments
This function is designed to work like the Visual Basic function of the same name.

Related topics
Split

Split

Basic and Crystal syntax.

Overloads

- Split (inputString)
- Split (inputString, delimiter)
- Split (inputString, delimiter, count)
- Split (inputString, delimiter, count, compare)

Arguments

- inputString is a String expression containing substrings and delimiters.
- delimiter is an optional String character used to identify substring limits. If omitted, the space character (" ") is assumed to be the delimiter. If delimiter is a zero-length string, a single-element array containing the entire inputString string is returned.
- count is an optional number value of substrings to be returned. The value indicates that all substrings are returned. If omitted, is assumed.
- compare is an optional Number indicating the kind of comparison to use when evaluating the delimiter string:
- 0 performs a comparison that is case-sensitive
- 1 performs a comparison that is case-insensitive

If omitted, a case-sensitive comparison is performed.

Note: Unlike in Visual Basic, in SCR if you omit an optional argument, you must omit all the following arguments. For example, if you do not specify a delimiter, you cannot specify count nor compare.

Returns

Array of String values.

Action

Split takes a String that contains a number of substrings, breaks it up into a specified number of substrings and returns an array containing the substrings.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
Split ("Chocolate Strawberry Pineapple")
```

Returns an array that contains 3 elements, "Chocolate", "Strawberry" and "Pineapple".

```
Split ("Chocolate//Strawberry//Pineapple", "//")
```

Returns an array that contains 3 elements, "Chocolate", "Strawberry" and "Pineapple".

```
Split ("Chocolate//Strawberry//Pineapple", "//", 2)
```

Returns an array that contains 2 elements, "Chocolate" and "Strawberry//Pineapple". The last element in the array is a concatenation of the 2nd substring and the remaining substring.

```
Split ("Chocolate and Strawberry and Pineapple", " And ", , 0)
```

Returns an array that contains 1 element, "Chocolate and Strawberry and Pineapple". The delimiter " And " cannot be matched.

```
Split ("Chocolate and Strawberry and Pineapple", " And ", , 1)
```

Returns an array that contains 3 elements, "Chocolate", "Strawberry" and "Pineapple". The delimiter " And " is matched with " and " regardless of the case.

Comments

- This function is designed to work like the Visual Basic function of the same name.
- If count, c, is less than the total number of substrings in inputString, then at most c substrings will be returned as elements in the resultant array. The last element in the array being a concatenation of the ch substring and all the remaining substrings.

Related topics

Join

ToText

Basic and Crystal syntax.

ToText and CStr are equivalent functions.

Overloads

- ToText (x)
- ToText (x, y)
- ToText (x, y, z)
- ToText (x, y, z, w)
- ToText (x, y, z, w, q)

Arguments

Converting . . .	Description
Boolean values	x is a Boolean value that is converted to a String, either "True" or "False".
Number and Currency values	• x is a Number or Currency value to be converted into a text string; it can be a whole or fractional value. • y is a whole number indicating the number of decimal places to carry the value in x to (This argument is optional.). • z is a single character text string indicating the character to be used to separate thousands in x. Default is the character specified in your International or Regional settings control panel. (This argument is optional.) • w is a single character text string indicating the character to be used as a decimal separator in x. Default is the character specified in your International or Regional settings control panel. (This argument is optional.)
Number and Currency values (formatting)	• x is a Number or Currency value to be converted into a text string; it can be a whole or fractional value. • y is a text string used to indicate the format for displaying the value in x. See Format Strings for information on creating a format string. • z is a whole number indicating the number of decimal places to carry the value in x to. (This argument is optional.) • w is a single character text string indicating the character to be used to separate thousands in x. Default is the character specified in your International or Regional settings control panel. (This argument is optional.) • q is a single character text string indicating the character to be used as a decimal separator in x. The default is the character specified in your International or Regional settings control panel. (This argument is optional.)
Date values	• x is a Date value to be converted into a text string. • y is a text string that defines how the value in x is to be formatted. See Format strings for Date, Time, and DateTime values for more information on creating this format string. (This argument is optional.)

Time values	• x is a Time value to be converted into a text string. • y is a text string that defines how the value in x is to be formatted. See Format strings for Date, Time, and DateTime values for more information on creating this format string. (This argument is optional.) • z is a text string to be used as a label for A.M. (morning) hours. (This argument is optional.) • w is a text string to be used as a label for P.M. (evening) hours. (This argument is optional.)
DateTime values	• x is a DateTime value to be converted into a text string. • y is a text string of characters that indicate how the resulting text string will be formatted. See Format strings for Date, Time, and DateTime values for more information on creating a format string. (This argument is optional.) • z is a text string to be used as a label for A.M. (morning) hours. (This argument is optional.) • w is a text string to be used as a label for P.M. (evening) hours. (This argument is optional.)

Returns

Text String

Action

The ToText function converts Numbers, Currency, Date, Time, and DateTime values to text strings.

Typical uses

Use this function to convert a Number, Currency, Date, Time, or DateTime value to a text string to appear as text in your report (form letters, comments, etc.).

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
ToText({orders.SHIPPED})
```

Returns True where the value in ({orders.SHIPPED}) is True.

```
ToText(123.45)
```

Returns "123.45".

```
ToText(12345.6749,2)
```

Returns "12345.67".

```
ToText(12345.6750,2)
```

Returns "12345.68".

```
ToText(12345.4999,0)
```

Returns "12345".

```
ToText(12345.5000,0)
```

Returns "12346".

```
ToText({file.AMT} * {file.QUANTITY})
```

Returns 44,890.20 where Amt = 24.45 and Quantity = 1836.

ToText is useful when you want to build a sentence by combining (concatenating) a converted Number or other value with other text strings:

```
"The base price of item # " + {file.ITEM NUMBER} + " is $" + ToText
({file.BASE PRICE}) + "."
```

Prints the sentence "The base price of item A1/4520/B12 is $50.00." where the Item number is A1/4520/B12 and the Base Price is 50.00, converted to text and formatted with two decimal places.

The following examples are applicable to Crystal syntax:

```
ToText(CDate(1996, 11, 1), "yy MMM dd, dddd")
```

Returns 96 Nov 01, Monday.

```
ToText(DateTime(1995,10,12,3,30,11),"HH:mm, yy MMMM ddd")
```

Returns 03:30, 95 October Mon.

```
ToText(Time(12, 10, 10), "HH*mm*ss tt", "amStr", "pmStr")
```

Returns 12*10*10 pmStr.

Comments

The overloads of ToText that take only one argument work like the Visual Basic function of the same name.

Converting Boolean values:

- The ToText function, when used with Boolean values, is most useful for combining (concatenating) a Boolean value with other text. Otherwise, a Boolean field can be formatted to appear as True or False in your report simply by changing the format on the Boolean Tab of the Format Editor.

Converting numbers and currency values:

- If the number of decimal places is specified, this function does not truncate the number when converted to text, but rounds it to the number of decimal places specified. See Round (x), Round (x, #places), for more information on the rounding procedure.

Converting Date, Time, and DateTime values:

- Any character, with the exception of the Date or Time format characters, can be used within the format string. For example, you may wish to use a slash to separate the different elements (month, day, year) of the date, as in "12/30/95", or you may wish to use a colon to separate the different elements (hours, minutes, seconds) of the time, as in "12:30:10".

- If you wish to use any of the above characters in the format string, they must appear in quotes. For example: ToText(CDateTime(1995,10,12,13,20,22), "MM/dd/yy hh 'h' mm 'min' ss 'sec' ", 'am', 'pm') = "10/12/95 1 h 20 min 22 sec pm"

Passing optional arguments:

- Many arguments for the ToText function have been specified as optional. However, you can only leave an argument blank if all the arguments that follow it are left blank as well. In other words, you can not leave the y and z arguments blank and provide an argument for w. It is possible, however, to leave one, two, or all of the optional arguments blank, as long as no arguments are supplied after the blank arguments. The following combinations are possible when supplying arguments to the ToText function:
- ToText (x)
- ToText (x, y)
- ToText (x, y, z)
- ToText (x, y, z, w)
- ToText (x, y, z, w, q)
- Using the "t" or "tt" format characters in a time format string provides default strings for indicating a.m. (morning) and p.m. (evening) hours. "t" produces just a single character, "a" or "p", while "tt" produces the entire string, "am" or "pm". You can pass your own custom strings for indicating am/pm strings. (See the arguments for converting Time and DateTime values with ToText in the Arguments section above.) If you do pass your own am/pm strings, the "t" and "tt" format characters will have the same effect on them (producing single character vs. multiple character strings). However, the "t" and "tt" format characters are optional and only necessary when default am/pm strings are needed.

Format Strings

The ToText and CStr functions can use format strings to control how the Number, Currency, Date, Time, or DateTime value passed to the x argument will appear when converted to a String. This section describes how to create format strings for the various data types.

Format strings for fractional Numbers and Currency values

The following characters are used to create format strings for fractional Numbers and Currency values:

Character	Description	Comments/Examples

#	Represents numerical digits or spaces. When the original number is converted, # characters are ignored unless the number of # characters in the format string extends beyond the number of significant digits in the original value. Then, for every additional # that appears, a space is added to the string.	If your original value is 125.34: • the format string "#" produces the string "125" (number of significant digits exceeds the number of # characters) • the format string "###.##" produces the string "125.34" (number of significant digits equal to number of # characters) • the format string "####.###" produces the string "125.34" (number of significant digits is less than the number of # characters).

0	Represents numerical digits or zeros (0). When the original number is converted, 0 characters are ignored unless the number of 0 characters in the format string extends beyond the number of significant digits in the original value. Then, for every additional 0 that appears, a zero is added to the string.	If your original value is 125.34: • the format string "0" produces the string "125" (number of significant digits exceeds the number of 0 characters) • the format string "000.00" produces the string "125.34" (number of significant digits equal to number of 0 characters) • the format string "0000.000" produces the string "0125.340" (number of significant digits is less than the number of 0 characters). # and 0 characters can be combined in format strings. If your original value is 125.34: • the format string "####.0000" produces the string "125.3400". • A comma (,) • A decimal point (.) indicates where a decimal separator should appear. The actual character that appears as a decimal separator in the resulting string can be changed.
comma (,)	Indicates where a thousands separator should appear. The actual character that appears as a thousands separator in the resulting string can be changed.	
decimal point (.)	Indicates where a decimal separator should appear. The actual character that appears as a decimal separator in the resulting string can be changed.	

How format string can affect values

The following table illustrates how format strings can affect different fractional Numbers and Currency values being converted. In this table, the underscore character (_) represents a space in the resulting string:

Original Value	Format string	Resulting String
16.13	#	16
16.13	0	16
16.13	##	16
16.13	00	16
16.13	###	_16
16.13	000	016
16.13	#.####	16.13_
16.13	0.0000	16.1300
16125.00	###,###.#	_16,125.0
16125.00	000,000.0	016,125.0
16125.00	000000.0	016125.0

Format strings for Date, Time, and DateTime values

The following characters are used to create format strings for Date, Time, and DateTime values:

Character	Comments
d	day of month without leading zero for single digits
dd	day of month with leading zero for single digits.
ddd	day of week as a three letter abbreviation
dddd	full name of day of week
M	month without leading zero for single digit
MM	month with leading zero for single digit
MMM	month as three letter abbreviation.
MMMM	full name of month
yy	last two digits of year
yyyy	full four digits of year
h	hours without leading zeros for single digits (12 hour)
hh	hours without leading zeros for single digits (12 hour).
H	hours without leading zeros for single digits (24 hour)
HH	hours with leading zeros for single digits (24 hour)
m	minutes without leading zeros for single digits
mm	minutes with leading zeros for single digits
s	seconds without leading zeros for single digits
ss	seconds with leading zeros for single digits
t, tt	single character or multiple character a.m./p.m. string

The following table illustrates how format strings can affect different Date, Time, and DateTime values being converted by the ToText or CStr function:

Date, Time, or DateTime value	**Format String**	**Resulting String**
CDate(1996, 11, 1)	yy MMM dd, dddd	96 Nov 01, Monday
CTime(12, 10, 10)	HH*mm*ss tt	12*10*10
CDateTime(1995, 10, 12, 3, 30, 1)	HH:mm, yy MMMM ddd	03:30, 96 October Mon
CTime(13, 20, 22)	hh "h" mm "min" ss "sec" tt	01 h 20 min 22 sec
CDate(1998, 3, 17)	M/dd/yy	3/17/98

Converting Boolean values

The ToText function, when used with Boolean values, is most useful for combining (concatenating) a Boolean value with other text. Otherwise, a Boolean field can be formatted to appear as True or False in your report simply by changing the format on the Boolean tab (Format Editor) of the Format Editor.

Converting fractional Numbers and Currency values

If the number of decimal places is specified, this function does not truncate the number when converted to text, but rounds it to the number of decimal places specified. See Round (x), Round (x, #places), for more information on the rounding procedure.

Converting Date, Time, and DateTime values

Any character, with the exception of the date or time format characters, can be used within the format string. For example, you may wish to use a slash to separate the different elements (month, day, year) of the date, as in "12/30/95", or you may wish to use a colon to separate the different elements (hours, minutes, seconds) of the time, as in "12:30:10".

If you wish to use any of the above characters in the format string, they must appear in quotes. For example:

```
ToText(DateTime(1995,10,12,13,20,22), "MM/dd/yy hh 'h' mm 'min' ss
'sec' ", 'am', 'pm') = "10/12/95 1 h 20 min 22 sec pm"
```

Passing optional arguments

- Many arguments for the ToText function have been specified as optional. However, you can only leave an argument blank if all the arguments that follow it are left blank as well. In other words, you can not leave the y and z arguments blank and provide an argument for w. It is possible, however, to leave one, two, or all of the optional arguments blank, as long as no arguments are supplied after the blank arguments.

  The following combinations are possible when supplying arguments to the ToText function:

- ToText(x)
- ToText (x, y)
- ToText (x, y, z)
- ToText (x, y, z, w)
- ToText (x, y, z, w, q)

- Using the "t" or "tt" format characters in a time format string provides default strings for indicating a.m. (morning) and p.m. (evening) hours. "t" produces just a single character, "a" or "p", while "tt" produces the entire string, "am" or "pm". You can pass your own custom strings for indicating am/pm strings. See the arguments for converting time and DateTime values with ToText in the Arguments section of ToText.)

  If you do pass your own am/pm strings, the "t" and "tt" format characters will have the same effect on them (producing single character vs. multiple character strings). However, the "t" and "tt" format characters are optional and only necessary when default am/pm strings are needed.

Related topics

CStr

Formula1

Formula 2

Formula 3

Formula 5

Formula 13

Date/Time functions

Date functions allow you to convert numbers to dates (which you can then format to display as you wish) and to convert dates to numbers.

CurrentDate

CurrentTime

CurrentDateTime

Date

Time

DateTime

DateValue

TimeValue

DateTimeValue

DateSerial (year, month, day)

TimeSerial (hour, minute, second)

IsDateTime

IsTime

IsDate

Year (x)

Month (x)

[Day (x)]
[WeekDay]
[DayOfWeek]
[Hour (x)]
[Minute (x)]
[Second (x)]
[MonthName]
[WeekdayName]
[Timer ( )]
[DateAdd (intervalType, nIntervals, startDateTime)]
[DateDiff]
[DatePart]
[First Day of Week/Year constants]

Additional Date/Time functions

[DateTo2000 (Date, Number)]
[DTSTo2000 (DateString, Number)]
[DTSToDate (DateTimeString)]
[DTSToSeconds (DateTimeString)]
[DTSToTimeString (DateTimeString)]

CurrentDate

Basic and Crystal syntax.

Returns

Date Value

Action

CurrentDate returns the current date on a report. The date is taken from your computer's internal clock.

Typical Uses

Use CurrentDate any time you want to stamp your report with the current date.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
CurrentDate
```

Returns 1998/07/04, if you begin printing your report on July 4, 1998.

```
CurrentDate
```

Returns 1996/12/11, if you begin printing your report on December 11, 1996.

Comments

- The date format is determined by the settings in the International or Regional Settings control panel.
- CurrentDate is identical to the Today function, which was included with previous version of the program. Today is provided for compatibility with reports created with earlier versions of the program. New reports should use the CurrentDate function.

CurrentTime

Basic and Crystal syntax.

Returns

Time Value

Action

CurrentTime returns the current time on a report. The time is taken from your computer's internal clock.

Typical Uses

Use CurrentTime any time you want to time stamp your report.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
CurrentTime
```

Returns 4:45:18 if you begin printing your report at 4:45:18.

```
CurrentTime
```

Returns 17:14:33 if you begin printing your report at 17:14:33.

Comments

- The time format is determined by the settings in the International or Regional Settings control panel.
- The CurrentTime function is identical to the Now function, which was included with previous versions of the program. Now is provided for compatibility with reports created with earlier versions of the program. New reports should use the CurrentTime function.

CurrentDateTime

Basic and Crystal syntax.

Returns

DateTime Value

Action

CurrentDateTime returns the current date and time on a report. The date and time is taken from your computer's internal clock.

Typical Uses

Use CurrentDateTime any time you want to stamp your report with the current date and time.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
CurrentDateTime
```

Returns 1998/07/04 4:45:18, if you begin printing your report at 4:45:18 on July 4, 1998.

```
CurrentDateTime
```

Returns 1996/12/11 17:14:33, if you begin printing your report at 17:14:33 on December 11, 1996.

Comments

- The date and time formats are determined by the settings in the International or Regional Settings control panel.
- CurrentDateTime produces the same result as the formula:

```
CDateTime(CurrentDate, CurrentTime)
```

Date

Crystal syntax.

The CDate and DateValue functions are equivalent to Date. However, Date can only be used in Crystal syntax since it is a type name in Basic syntax.

Overloads

- Date (number)
- Date (string)
- Date (dateTime)
- Date (YYYY, MM, DD)

Arguments

Date (number)	• number is a value representing the number of days starting from December 30, 1899. • It can be positive or negative, and is truncated if fractional.
Date (string)	• string is a text string representing a date, example: "September 20, 1999".
Date (dateTime)	dateTime is a DateTime value.

Date (YYYY, MM, DD)	• YYYY is a whole number representing a year, example: 1996. • MM is a whole number representing a month, example: 12 for December. • DD is a whole number representing a day of the month, example: 05.

Returns

A Date value.

Action

Date (number)	Returns a Date value given a number of days starting from December 30, 1899.
Date (string)	Returns a Date value given a string that represents a date from January 1, 100 through December 31, 9999.
Date (dateTime)	Returns the date portion of a given DateTime value.
Date (YYYY, MM, DD)	Returns a Date value given numeric arguments of the year, month, and day.

Examples

The following examples are applicable to Crystal syntax:

```
Date ("Dec 31, 1999")
```

Returns the Date value for Dec. 31, 1999.

```
Date (50)
```

Returns the Date value for February 18, 1900.

```
Date (#Oct. 20, 1999 12:02pm#)
```

Returns the Date value for October 20, 1999.

```
Date (1930, 7, 30)
```

Returns the Date value for July 30, 1930.

Comments

You can use the IsDate function to check if a String argument can be converted to a Date before doing the actual conversion. That way, if the conversion cannot be done, you can handle the situation appropriately.

Related topics

DateSerial (year, month, day)

Time

Crystal syntax.

The CTime and TimeValue functions are equivalent to Time. However, Time can only be used in Crystal syntax since it is a type name in Basic syntax.

Overloads

- Time (number)
- Time (string)
- Time (dateTime)
- Time (HH, MM, SS)

Arguments

Time (number)	• Returns a Time value given a number in units of 24 hours • number can be fractional or negative as well
Time (string)	Returns a Time value that represents the time, given a String expression specifying a time from 0:00:00 (12:00:00 A.M.) to 23:59:59 (11:59:59 P.M.), inclusive.
Time (dateTime)	DateTime is a DateTime value.
Time (hour, min, sec)	• Hour is a whole number representing an hour of the day. • Min is a whole number representing a minute. • Sec is a whole number representing a second.

Returns
Time Value

Action
Creates a Time value from the information provided.

Typical uses
Use this function any time you need to work with a Time value that is obtained either from a DateTime value or from the individual elements of the time (i.e. hour, minute, second).

Examples
The following examples are applicable to Crystal syntax:

```
Time(.2) and Time (1.2)
```

Both return 4:48:00 AM

```
Time(-.2)
```

Returns 7:12:00 PM

```
Time("3:05pm")
```

Returns 3:05:00 PM

```
Time(DateTime(1996, 1, 1, 12, 10, 10)
```

Returns 12:10:10.

```
Time(10, 20, 22)
```

Returns 10:20:22.

Related topics
TimeValue

DateTime

Crystal syntax.

The CDateTime and DateTimeValue functions are equivalent to DateTime. However, DateTime can only be used in Crystal syntax since it is a type name in Basic syntax.

Overloads

- DateTime (date)
- DateTime (number)
- DateTime (string)
- DateTime (date, time)
- DateTime (YYYY, MM, DD)
- DateTime (YYYY, MM, DD, HH, MM, SS)

Arguments

DateTime (date)	date is a Date value.
DateTime (date, time)	• date is a Date value. • time is a Time value.
DateTime (string)	string represents a date and time Example: "September 15, 1999, 10:45a.m."
DateTime (number)	number represents a number of days from December 30, 1899. Example: 20 represents 20 days from December, 1899, which is January 19, 1900.

DateTime (year, month, day) DateTime (year, month, day, hour, min, sec)	• year is a whole number representing a calendar year, example: 1996. • month is a whole number representing a month, example: 1 for January. • day is a whole number representing a day of the month, example: 10. • hour is a whole number representing an hour of the day, example: 12. • min is a whole number representing a minute, example: 59. • sec is a whole number representing seconds, example: 30.

Returns

DateTime value.

Action

DateTime (date)	Returns a DateTime value given a Date value, assigning 12:00:00 AM for the time portion for the returned DateTime value.
DateTime (date, time)	Returns a DateTime value given a Date and a Time value.
DateTime (number)	Returns a DateTime value given a number that specifies the number of days from December 30, 1899. Number can be positive or negative, and can be fractional.
DateTime (string)	Returns a DateTime value given a string that specifies a date and time; various formats of the string are supported.
DateTime (YYYY, MM, DD)	Returns a DateTime value given numeric arguments for the year, month and day. Assigns 12:00:00 AM for the time portion for the returned DateTime value.
DateTime (YYYY, MM, DD, HH, MM, SS)	Returns a DateTime value given numeric arguments for the year, month, day, hour, minute and second.

Examples

The following examples are applicable to Crystal syntax:

```
DateTime ("10/4/1999 10:20am")
```

Returns the DateTime value October 4, 1999 10:20:00 am.

```
DateTime (12.5)
```

Returns the DateTime value January 11, 1900 12:00:00 pm.

```
DateTime (.5)
```

Returns the DateTime value December 27, 1899, 12:00:00 pm.

```
DateTime (CDate ("Dec. 25, 1999"))
```

Returns the DateTime value December 25, 1999 12:00:00 am.

```
DateTime (CDate ("November 10, 1999"), CTime("12:20am"))
```

Returns the DateTime value November 10, 1999 12:20:00 am.

```
DateTime (1945, 8, 21, 0, 0, 0)
```

Returns the DateTime value August 21, 1945 12:00:00 am.

```
DateTime (1945, 8, 21, 10, 0, 0)
```

Returns the DateTime value August 21, 1945 10:00:00 am.

Comments

You can use the IsDateTime function to check if a String argument can be converted to a DateTime before doing the actual conversion. That way, if the conversion cannot be done, you can handle the situation appropriately.

Related topics

DateTimeValue

DateValue

Basic and Crystal syntax.

The CDate and Date functions are equivalent to DateValue. However, Date can only be used in Crystal syntax since it is a type name in Basic syntax.

Overloads

- DateValue (string)
- DateValue (number)
- DateValue (dateTime)
- DateValue (YYYY, MM, DD)

Arguments

DateValue (number)	• number is a value representing the number of days starting from December 30, 1899. • It can be positive or negative, and is truncated if fractional.
DateValue (string)	string is a text string representing a date, example: "September 20, 1999".
DateValue (dateTime)	dateTime is a DateTime value.
DateValue (YY,MM,DD)	• year is a whole number representing a year, example: 1996. • month is a whole number representing a month, example: 12 for December. • day is a whole number representing a day of the month, example: 05.

Returns

Date value.

Action

DateValue (string) returns a Date value given a string that represents a date from January 1, 100 through December 31, 9999.

DateValue (number) returns a Date value given a number of days starting from December 30, 1899. Number can be positive or negative, and truncated if fractional.

DateValue (dateTime) returns the date portion of a given DateTime value.

DateValue (YYYY, MM, DD) returns a Date value given numeric arguments of the year, month and day.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
DateValue ("Dec 31, 1999")
```

Returns the Date value for Dec. 31, 1999.

```
DateValue (50)
```

Returns the Date value for February 18, 1900.

```
DateValue (#Oct. 20, 1999 12:02pm#)
```

Returns the Date value for October 20, 1999.

```
DateValue (1930, 7, 30)
```

Returns the Date value for July 30, 1930.

Comments

You can use the IsDate function to check if a String argument can be converted to a Date before doing the actual conversion. That way, if the conversion cannot be done, you can handle the situation appropriately.

Related topics

DateSerial (year, month, day)

TimeValue

Basic and Crystal syntax.

The CTime and Time functions are equivalent to TimeValue. However, Time can only be used in Crystal syntax since it is a type name in Basic syntax.

Overloads

- TimeValue (number)
- TimeValue (string)
- TimeValue (dateTime)
- TimeValue (HH, MM, SS)

Arguments

TimeValue (number)	• Returns a Time value given a number in units of 24 hours number can be fractional or negative as well.
TimeValue (string)	Returns a Time value that represents the time, given a String expression specifying a time from 0:00:00 (12:00:00 A.M.) to 23:59:59 (11:59:59 P.M.), inclusive.
TimeValue (dateTime)	DateTime is a DateTime value.
TimeValue (hour, min, sec)	• Hour is a whole number representing an hour of the day. • Min is a whole number representing a minute. • Sec is a whole number representing a second.

Returns

Time value.

Action

TimeValue (number) returns a Time value given a number in units of 24 hours. The number can be negative or fractional as well.

TimeValue (string) returns a Time value that represents the time, given a String expression specifying a time from 0:00:00 (12:00:00 A.M.) to 23:59:59 (11:59:59 P.M.), inclusive.

TimeValue (dateTime) returns the time portion of a given DateTime value.

TimeValue (HH, MM, SS) returns a Time value given numeric arguments of the hour, minute and seconds.

Examples

The following examples are applicable to both Basic and Crystal syntax:

TimeValue (.2)
 TimeValue (1.2)

Both return the same value 4:48:00 a.m.

```
TimeValue (-.2)
```

Returns 7:12:00 pm

```
TimeValue ("Jan. 6, 1999 2:30pm")
```

Returns 2:30:00 pm

```
TimeValue (#Feb. 24, 1999 2:40pm#)
```

Returns 2:40:00 pm

```
TimeValue (18, 30, 30)
```

Returns 6:30:30 pm

Comments

You can use the IsTime function to check if a string argument can be converted to a Time before doing the actual conversion. That way, if the conversion cannot be done, you can handle the situation appropriately. See alsoTimeSerial (hour, minute, second).

DateTimeValue

Basic and Crystal syntax.

The CDateTime and DateTime functions are equivalent to DateTimeValue. However, DateTime can only be used in Crystal syntax since it is a type name in Basic syntax.

Overloads

- DateTimeValue (date)
- DateTimeValue (date, time)
- DateTimeValue (number)
- DateTimeValue (string)
- DateTimeValue (YYYY, MM, DD)
- DateTimeValue (YYYY, MM, DD, HH, MM, SS)

Arguments

DateTimeValue (date)	• date is a Date value.
DateTimeValue (date, time)	• date is a Date value. • time is a Time value.
DateTimeValue (number)	• number represents a number of days from December 30, 1899, example: 20 represents 20 days from December, 1899, which is January 19, 1900.
DateTimeValue (string)	• string represents a date and time, example: "September 15, 1999, 10:45a.m."
DateTimeValue (year, month, day) DateTimeValue (year, month, day, hour, min, sec)	• year is a whole number representing a calendar year, example: 1996. • month is a whole number representing a month, example: 1 for January. • day is a whole number representing a day of the month, example: 10. • hour is a whole number representing an hour of the day, example: 12. • min is a whole number representing a minute, example: 59. • sec is a whole number representing seconds, example: 30.

Returns

DateTime value.

Action

DateTimeValue (date) returns a DateTime value given a Date value, assigning 12:00:00 AM for the time portion for the returned DateTime value.

DateTimeValue (date, time) returns a DateTime value given a Date and a Time value.

DateTimeValue (number) returns a DateTime value given a number that specifies the number of days from December 30, 1899. Number can be positive or negative, and can be fractional.

DateTimeValue (string) returns a DateTime value given a string that specifies a date and time; various formats of the string are supported.

DateTimeValue (YYYY, MM, DD) returns a DateTime value given numeric arguments for the year, month and day. Assigns 12:00:00 AM for the time portion for the returned DateTime value.

DateTimeValue (YYYY, MM, DD, HH, MM, SS) returns a DateTime value given numeric arguments for the year, month, day, hour, minute and second.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
DateTimeValue ("10/4/1999 10:20am")
```

Returns the DateTime value October 4, 1999 10:20:00 am

```
DateTimeValue (12.5)
```

Returns the DateTime value January 11, 1900 12:00:00 pm

```
DateTimeValue (.5)
```

Returns the DateTime value December 27, 1899, 12:00:00 pm

```
DateTimeValue (CDate ("Dec. 25, 1999"))
```

Returns the DateTime value December 25, 1999 12:00:00 am

```
DateTimeValue (CDate ("November 10, 1999"), CTime("12:20am"))
```

Returns the DateTime value November 10, 1999 12:20:00 am

```
DateTimeValue (1945, 8, 21, 0, 0, 0)
```

Returns the DateTime value August 21, 1945 12:00:00 am

```
DateTimeValue (1945, 8, 21, 10, 0, 0)
```

Returns the DateTime value August 21, 1945 10:00:00 am

Comments

You can use the IsDateTime function to check if a String argument can be converted to a DateTime before doing the actual conversion. That way, if the conversion cannot be done, you can handle the situation appropriately.

DateSerial (year, month, day)

Basic and Crystal syntax.

Arguments

- year is a whole Number or numeric expression representing a year, example: 1996.
- month is a whole Number or numeric expression representing a month, example: 12 for December.
- day is a whole Number or numeric expression representing a day of the month, example: 5.

Returns

A Date value.

Action

DateSerial returns a Date value for the specified year, month and day. It also handles relative Date expressions.

Typical uses

DateSerial can be used instead of CDate or DateValue to create a Date value out of a year, month and day.

A useful feature of DateSerial is that the month argument does not have to be from 1 to 12 and the day argument does not have to be in the valid range for the number of days in the month. Such dates are interpreted as relative dates, and DateSerial will produce a valid Date value. This can be used to perform many computations with dates, without checking for special cases such as the end of the year, leap years and the number of days in a month. See the examples below for some typical applications.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
DateSerial (1999, 6, 15)
```

```
DateSerial (2000, 1 - 7, 15)
```

```
DateSerial (1999, 1, 166)
```

All three return the date June 15, 1999. The second says that 7 months before January 1, 2000 is June 15, 1999. The third says that the 166th day of 1999 is June 15, 1999.

```
DateSerial (1996 + 12, 2 + 13, 29 + 14)
```

Returns April 12, 2009. What it means is that 12 years, 13 months and 14 days from February 29, 1996 is April 12, 2009.

Suppose you want to calculate the last day of the month for the DateTime field {Orders.Order Date}. Notice that in the calculation, DateSerial(Year(d), Month(d) + 1, 1) is the first day of the month after {Orders.Order Date} and then subtracting one day gives the desired result:

Rem Basic syntax

Dim d

d = {Orders.Order Date}

formula = DateSerial(Year(d), Month(d) + 1, 1 - 1)

//Crystal syntax

Local DateTimeVar d := {Orders.Order Date};

DateSerial(Year(d), Month(d) + 1, 1 - 1)

It is often useful to combine DateSerial with other date functions. For example, if you want to calculate the date of the last Friday of the month for the field {Orders.Order Date} you can do the following.

The calculation proceeds by finding the last day of the month and then subtracting off a number of days to get to a Friday. Since some months have 5 Fridays (example: October 1999) and some have 4 Fridays (example: November 1999), this approach is easier than working from the first day of the month.

Rem Basic syntax

```
Dim d1, d2
d1 = {Orders.Order Date}
d2 = DateSerial(Year(d1), Month(d1) + 1, 1 - 1)
formula = d2 - (WeekDay(d2, crFriday) - 1)

//Crystal syntax
Local DateTimeVar d1 := {Orders.Order Date};
Local DateVar d2;
d2 := DateSerial(Year(d1), Month(d1) + 1, 1 - 1);
d2 - (DayOfWeek(d2, crFriday) - 1)
```

Returns the Date value March 27, 1998 if {Orders.Order Date} is March 18, 1998.

Here is an example that returns the date of the second Tuesday of the month 3 months before {Orders.Order Date}:

```
Rem Basic syntax
Dim d1, d2
d1 = {Orders.Order Date}
d2 = DateSerial(Year(d1), Month(d1) - 3, 1)
formula = d2 + (2 * 7 - (WeekDay(d2, crTuesday) - 1))

//Crystal syntax
Local DateTimeVar d1 := {Orders.Order Date};
Local DateVar d2 := DateSerial(Year(d1), Month(d1) - 3, 1);
d2 + (2 * 7 - (DayOfWeek(d2, crTuesday) - 1))
```

Returns the Date value December 9, 1997 if {Orders.Order Date} is March 18, 1998.

Comment

This function is designed to work like the Visual Basic function of the same name.

TimeSerial (hour, minute, second)

Basic and Crystal syntax.

Arguments

- hour is a Number or numeric expression specifying the hour.
- minute is a Number or numeric expression specifying the number of minutes.
- second is a Number or numeric expression specifying the number of seconds.

Returns

Time value.

Action

TimeSerial returns a Time value specifying the time for a specific hour, minute and second.

Typical uses

TimeSerial can be used instead of CTime or TimeValue to create a Time value out of an hour, minute and second.

A useful feature of TimeSerial is that the hour, minute and second arguments do not need to fall within normally valid ranges. That is, the hour does not need to be from 0 to 24 and the minute and second do not need to be from 0 to 59. Such times are interpreted as relative times, and TimeSerial will produce a valid Time value. This can be used to easily perform many computations with times.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
TimeSerial (18, 45, 0)
```

```
TimeSerial (22 - 3, 15 - 30, 0)
```

Both return the same time 6:45 PM. The second has the interpretation that 3 hours and 30 minutes before 10:15 PM is 6:45 PM.

```
TimeSerial (10 + 20, 30 + 55, 0)
```

Returns 7:25 AM. What it means is that 20 hours and 55 minutes from 10:30 AM is 7:25 AM (of the next day).

Comment

This function is similar to the Visual Basic function of the same name. There is a difference when the relative times span more than one day. For example, in Crystal Reports,

```
TimeSerial (1, 0 - 180, 0)
```

returns 10:00 PM. This has the interpretation that 180 minutes before 1:00 AM is 10:00 PM. However, the same function call in Visual Basic returns 2:00 AM. The difference occurs because 180 minutes before 1:00 AM moves the time to 10:00 PM *of the previous day* and Visual Basic then handles this differently.

IsDateTime

Basic and Crystal syntax.

Overloads

- IsDateTime (string)
- IsDateTime (number)

Arguments

- string is a String value or expression to be tested for being convertible to a DateTime value. Many forms are accepted.
- number is a Number value or expression to be tested for being convertible to a DateTime value. It can be positive, negative or fractional. It is interpreted as a number of days since December 30, 1899.

Returns

Boolean value (True or False).

Action

IsDateTime returns True if the given String or Number value can be converted to a valid DateTime and returns False otherwise. A valid DateTime is any date-time between January 1, 100 through December 31, 9999.

Typical use

If you want to convert a String value to a DateTime using the functions CDateTime or DateTimeValue, use the IsDateTime function first to check if the conversion will succeed. See the examples section of IsDate for an example of this type.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
IsDateTime ("September 30, 1999")
```

Returns True since the given string "September 30, 1999 is interpreted as the DateTime value September 30, 1999, 12:00:00am.

```
IsDateTime (00.2)
```

Returns True since the given number argument is interpreted as the DateTime value September 20, 1899, 7:12:00 pm.

Comment

This function is similar to the Visual Basic function IsDate.

IsTime

Basic and Crystal syntax.

Overloads

- IsTime (string)
- IsTime (number)

Arguments

- string is a String value or expression to be tested for being convertible to a Time value. Many forms are accepted

- number is a Number value or expression to be tested for being convertible to a Time value. It can be positive, negative or fractional. It is interpreted as units of 24 hours. What this means is that 0 is 12 midnight and 0.5 is 12 noon.

Returns

Boolean value (True or False).

Action

IsTime returns True if the given String or Number value can be converted to a valid Time and returns False otherwise.

Typical use

If you want to convert a String value to a Time using the functions CTime or TimeValue, use the IsTime function first to check if the conversion will succeed. See the examples section of IsDate for an example of this type.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
IsTime ("10:30 pm")
```

Returns True.

```
IsTime ("September 30,1999")
```

Returns True since the given String argument is interpreted as 12:00:00 am.

```
IsTime (3.2)
```

Also returns True since the given Number argument is interpreted as 3.2 units of 24 hours, which is 4:48:00am.

Comment

This function is similar to the Visual Basic function IsDate, except that it works specifically with Crystal Report's Time type which holds Time values only.

IsDate

Basic and Crystal syntax.

Overloads

- IsDate (string)
- IsDate (number)

Arguments

- string is a String value or expression to be tested for being convertible to a Date value. Many forms are accepted.
- number is a Number value or expression to be tested for being convertible to a Date value. It can be positive, negative or fractional. It is interpreted as a number of days since December 30, 1899.

Returns

Boolean value (True or False).

Action

IsDate returns True if the given String or Number value can be converted to a valid Date and returns False otherwise. A valid Date is any date between January 1, 100 through December 31, 9999.

Typical use

If you want to convert a String value to a Date using the functions CDate or DateValue, use the IsDate function first to check if the conversion will succeed. See below for an example.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
IsDate ("Jan 1, 1999")
```

Returns True.

```
IsDate (100)
```

Also returns True since the number 100 is interpreted as 100 days from Dec. 30, 1899, which is April 9, 1900.

```
IsDate ("Feb 29, 1999")
```

Returns False since 1999 is not a leap year and so the String cannot be converted to a Date.

Suppose an orders report is grouped monthly by order date. Here is a formula that returns the Date value that is extracted from the GroupName String field. If the report is a TopN report, the GroupName field can have a value "Others". This can not be converted into a Date, and so IsDate is used to prevent an error from being generated in the call to CDate:

Rem Basic syntax

Dim s As String

s = GroupName ({Orders.Order Date}, "monthly")

If IsDate(s) Then

formula = CDate(s)

Else

formula = CDate(0,0,0)

End If

//Crystal syntax

Local StringVar s := GroupName ({Orders.Order Date}, "monthly");

If IsDate(s) Then

CDate(s)

Else

CDate(0,0,0)

Returns the Date value May 1, 1998 if the GroupName field value is "May - 1998". Returns the null Date (a non-printing Date value) if the GroupName field value is "Others".

Comment

This function is similar to the Visual Basic function of the same name, except that it works specifically with Crystal Report's Date type which holds date values only.

Year (x)

Basic and Crystal syntax.

Arguments

x is a Date value or a DateTime value.

Returns

Whole Number

Action

Year extracts the year from a date and returns it as a Number.

Typical uses

Use this function any time you need to use a year, converted to a Number, in calculations or comparisons.

Examples

The following examples are applicable to Crystal syntax:

```
Year({file.LAST INVOICE ON})
```

Returns 1989, where the last invoice date was in 1989.

If Year({file.LAST UPGRADE}) < 1988 Then

"P.S. You are missing out on many of the benefits of our new, improved version."

Else

""

Prints a P.S. for those customers who last upgraded prior to 1988, and prints nothing for those customers who show a more recent upgrade.

```
Year(CDateTime(1996, 3, 3, 10, 33, 20))
```

Returns 1996.

Comments

This function is designed to work like the Visual Basic function of the same name.

Month (x)

Basic and Crystal syntax.

Arguments
x is a Date value or a dateTime value.

Returns
Whole Number

Action
Month extracts the month component of a date and converts it to a Number.

Typical uses
Use any time you need only the month component of a date. For example, if you are tracking payments that fall within a given year, you are interested only in the month they arrive; day and year information would be redundant. Also, if you need to use the number of the month in a numeric calculation (Month({file.OCTPMT}) - Month({file.SEPPMT}), for example) use the Month function to extract the month component of the date and convert it to a Number.

Examples
The following examples are applicable to Crystal syntax:

```
Month({file.INVOICE}) - Month({file.PAYMENT})
```

Calculates the number of months between the Invoice and the Payment.

If Month({file.LAST PURCHASE}) < 6 Then

"You will see some changes since you last shopped our store."

Else

"As you have certainly noticed, we have been making many changes in our store recently.";

Prints "You will see some changes since you last shopped our store." for those customers whose last purchase was in May or earlier, and prints "As you have certainly noticed, we have been making many changes in our store recently." for customers who have purchased more recently than May.

Comments
This function is designed to work like the Visual Basic function of the same name.

Day (x)

Basic and Crystal syntax.

Arguments
x is a Date value or a DateTime value.

Returns
Whole Number (the day of the month)

Action

Day extracts the day from a Date or DateTime value and returns a whole number.

Typical uses

Use any time you need only the day component of a Date or DateTime value. For example, if you are tracking payments that fall within a given month, you are interested only in the day they arrive; month and year information would be redundant. Also, if you need to use the day of the month in a numeric calculation (example: Day({file.OCTPMT}) - Day ({file.SEPPMT})), use the Day function to extract the day of the month and convert it to a Number.

Examples

The following examples are applicable to Crystal syntax:

```
Day({file.LAST INVOICE})
```

Returns 10 where the last invoice date was the 10th day of the month.

If Day({file.LAST PAYMENT}) < 15 Then

"Past Due"

Else

""

If the day of the last payment was less than 15, print "Past Due", otherwise print nothing.

```
Day(CDateTime(1996, 3, 3, 10, 33, 20))
```

Returns 3.

Comments

This function is designed to work like the Visual Basic function of the same name.

WeekDay

WeekDay and DayOfWeek are equivalent functions. However, WeekDay is preferred in Basic syntax and DayOfWeek is preferred in Crystal syntax.

Overloads

- WeekDay (date)
- WeekDay (date, firstDayOfWeek)

Arguments

- date is a Date or DateTime value for which a numeric designation for the day of the week is to be returned.
- firstDayOfWeek is an optional Number indicating the first day of the week. See DayOfWeek for more information. If not specified, Sunday is assumed.

Returns

Number value.

Action
WeekDay returns a number n which is the nth day of the week given a Date or DateTime value.

Typical use
WeekDay is often useful in performing date computations that take into account special days of the week such as weekends. See the DateSerial (year, month, day) examples section for some examples.

Examples
The following examples are applicable to both Basic and Crystal syntax:

```
WeekDay (#Oct. 4, 1999#)
```

Returns 2 since Oct. 3, 1999 is a Monday, and the first day of the week is assumed to be Sunday.

```
WeekDay (#Oct. 4, 1999#, crMonday)
```

Returns 1 since Oct. 3, 1999 is a Monday, and the first day of the week is specified to be Monday.

Comments
This function is designed to work like the Visual Basic function of the same name.

DayOfWeek

DayOfWeek and WeekDay are equivalent functions. However, WeekDay is preferred in Basic syntax and DayOfWeek is preferred in Crystal syntax.

Overloads

- DayOfWeek (date)
- DayOfWeek (date, firstDayOfWeek)

Arguments
date is a Date value or dateTime value.

Returns
Whole Number

Action
DayOfWeek determines the day of the week the given date falls on, and converts the day of the week to a Number (1 to 7). Optionally a numeric value for the first day of the week can be specified. If the first day of the week is not specified, Sunday is assumed.

Typical uses
Use this function any time you need to use the day of the week as a Number.

Examples
The following examples are applicable to Crystal syntax:

```
DayOfWeek(Date(1990,10,1))
```

Returns 2 where October 1, 1990 is a Monday.

If DayOfWeek({orders.ORDER DATE}) = 3 Then

"Sam"

Else

"Bill"

In determining whether Sam or Bill was on duty on September 8, 1990.

If DayOfWeek({orders.ORDER DATE}) = 7 Then

"Saturday"

Else

""

DayOfWeek(#Sept. 24, 1999#, crMonday)

Returns 5 where Sept. 24, 1999, is a Friday, and Monday is specified to be the first day of the week.

Comments

If you want to get the day of the week spelled out, in Crystal syntax, use this formula:

```
["Sun", "Mon", "Tues",...] [DayOfWeek(Date)]
```

Sets up an array (["Sun",...]) and uses the number of the day of the week (Sun = 1, Sat = 7) to select the desired date name from the array.

Related topics

Formula 10

Hour (x)

Basic and Crystal syntax.

Arguments

x is a Time value or DateTime value.

Returns

Whole Number

Action

The Hour function extracts the hour portion of a Time value and returns it as a whole number.

Typical uses

Use this function anytime you need to display just the hour portion of a Time value or to perform calculations based on hours.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
Hour(CTime(10, 20, 22))
```

Returns 10.

```
Hour(CTime(10, 00, 30)) - Hour(CTime(6, 00, 00))
```

Returns 4.

Comments

This function is designed to work like the Visual Basic function of the same name.

Minute (x)

Basic and Crystal syntax.

Arguments

x is a Time value.

Returns

Whole Number

Action

Retrieves just the minutes from a Time value.

Typical uses

Use this function whenever you need to work with the minutes specified in a Time value for performing time calculations.

Examples

The following examples are applicable to Crystal syntax:

```
Minute(CTime(10, 20, 22))
```

Returns 20.

TimeVar ShipTime := CTime({orders.SHIP DATE});

Minute(ShipTime)

Returns 30 if {orders.SHIP DATE} is 1998/12/05 8:30:15.

Comments

This function is designed to work like the Visual Basic function of the same name.

Second (x)

Basic and Crystal syntax.

Arguments

x is a Time value or DateTime value.

Returns
Whole Number

Action
Extracts the seconds portion of a specified Time value.

Typical uses
Use this function anytime you need to perform calculations with the seconds indicated in a Time value.

Examples
The following examples are applicable to both Basic and Crystal syntax:

```
Second(CTime(10, 20, 22))
```

Returns 22.

Comments
This function is designed to work like the Visual Basic function of the same name.

MonthName

Basic and Crystal syntax.

Overloads

- MonthName (month)
- MonthName (month, abbreviate)

Arguments

- month is a whole Number representing the month of the year, value between 1 and 12, with 1 being January.
- abbreviate is an optional Boolean value that indicates if the month name is to be abbreviated. If omitted, the default is False.

Returns
String value.

Action
MonthName returns a string name for the specified month.

Examples
The following examples are applicable to both Basic and Crystal syntax:

```
MonthName (4)
```

Returns the String value "April".

```
MonthName (4, True)
```

Returns "Apr".

Comment
This function is designed to work like the Visual Basic function of the same name.

WeekdayName

Basic and Crystal syntax.

Overloads

- WeekdayName (weekday)
- WeekdayName (weekday, abbreviate)
- WeekdayName (weekday, abbreviate, firstDayOfWeek)

Arguments

- weekday is the numeric designation for the day of the week. Numeric value of each day depends on the value of the firstDayOfWeek argument. If firstDayOfWeek is not specified, crSunday is assumed.
- abbreviate is an optional Boolean value that indicates if the weekday name is to be abbreviated. If omitted, the default is False.
- firstDayOfWeek is an optional Number indicating the first day of the week. See Day of Week constants.

Returns
String value.

Action
WeekdayName returns a String indicating the name of the specified day of the week.

Examples
The following examples are applicable to both Basic and Crystal syntax:

```
WeekdayName (3)
```

Returns "Tuesday" since Sunday is assumed to be the first day of the week.

```
WeekdayName (3, True, crMonday)
```

Returns "Wed" since abbreviation is selected and the first day of the week is specified to be Monday.

Comments
This function is designed to work like the Visual Basic function of the same name.

Timer ( )

Basic and Crystal syntax.

Returns
Number value.

Action

Timer returns the number of seconds elapsed since midnight.

Example

The following example is applicable to both Basic and Crystal syntax:

```
Timer
```

Returns 57,720 if the current time is 4:02 PM.

Comments

This function is designed to work like the Visual Basic function of the same name.

DateAdd (intervalType, nIntervals, startDateTime)

Basic and Crystal syntax.

Arguments

intervalType is a String expression specifying the interval of time to be added. Values can be:

IntervalType value	Description
yyyy	Year
q	Quarter (3onth period)
m	Month
y	Day of year
d	Day
w	Weekday
ww	Week (7ay period)
h	Hour
n	Minute
s	Second

- nIntervals is a Number or numeric expression specifying the number of intervals to be added. It can be positive (to get date-times in the future) or negative (to get date-times in the past).
- startDateTime is the DateTime value to which the intervals are to be added.

Returns

A DateTime value.

Action

DateAdd returns a DateTime value to which a specified number of time intervals have been added.

Typical use

DateAdd is used to add intervals of time to a DateTime. Its main feature is that the DateTime returned will always be valid. For example, DateAdd takes into account such factors as the number of days in a month and leap years. If you want to add or subtract days to a DateTime, you could use the addition and subtraction operators instead of DateAdd with the "d" parameter. However, DateAdd also handles other types of intervals such as adding months or hours to a DateTime.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
DateAdd("d", 2, #Sept 28, 1999#)
```

Returns the DateTime value for August 27, 1999.

```
DateAdd ("m", 1, #1/31/1996#)
```

Returns the DateTime value for February 29, 1996. Notice that DateAdd will not return the invalid value February 31, 1996.

```
DateAdd ("q", 17, #Sept 28, 1999#)
```

Returns the DateTime value for December 28, 2003.

```
DateAdd ("h", 00, #Sept 28, 1999#)
```

Returns the DateTime value for September 11, 1999 8:00 AM. In other words, this is the result of subtracting 400 hours from September 28, 1999 12:00 AM.

Comments

- This function is designed to work like the Visual Basic function of the same name.
- To add days to a date-time, you can use any of the interval type parameters "y", "d" or "w". They all have the same effect for DateAdd.
- DateAdd returns a DateTime value and not a Date value. However, you may need to convert this DateTime value to a Date value in certain situations (such as if you wanted to assign the value returned by DateAdd to a Date type variable). To convert to a Date value, use DateAdd in combination with the CDate type conversion function. For example, the following returns the Date value for October 6, 1997:

```
CDate(DateAdd("yyyy", , #October 6, 1999#))
```

DateDiff

Basic and Crystal syntax.

Overloads

- DateDiff (intervalType, startDateTime, endDateTime)
- DateDiff (intervalType, startDateTime, endDateTime, firstDayOfWeek)

Arguments

- intervalType is a String expression that is the interval of time you use to calculate the difference between startDateTime and endDateTime. Possible values can be:

Interval type value	**Description**
yyyy	Year
q	Quarter
m	Month
y	Day of year
d	Day (both "y" and "d" find the difference in days)
w	Number of weeks between startDateTime and endDateTime
ww	Number of firstDayOfWeek's between startDateTime and endDateTime
h	Hour
n	Minute
s	Second

- startDateTime is the first DateTime value used in calculating the difference.
- endDateTime is the second DateTime value used in calculating the difference.
- firstDayOfWeek is an optional constant specifying the first day of the week. If not specified, crSunday is assumed. See Day of Week constants.

Returns

A Number value.

Action

DateDiff returns a number of time intervals between two specified dates.

Examples

The following examples are applicable to both Basic and Crystal syntax:

Use DateDiff with the "d" or "y" interval type parameter to find the number of days between two dates:

```
DateDiff ("d", #10/7/1999#, #10/10/1999#)
```

Returns 3.

Use DateDiff with the "yyyy" interval type parameter to find the number of years difference between two dates. This use of DateDiff is the same as finding the difference between the year of endDateTime and the year of startDateTime.

```
DateDiff ("yyyy", #10/7/1999#, #2/10/2005#)
```

Returns 6.

```
DateDiff ("yyyy", #12/31/1999#, #1/1/2000#)
```

Returns 1 (a 1 year difference), even though there is only a 1 day difference between the dates.

```
DateDiff ("yyyy", #1/1/1999#, #12/31/1999#)
```

Returns 0 (a 0 year difference), even though there is a 364 day difference.

Suppose that for the above examples, the first date is the date that you bought a mutual fund, and the second date is the date you sold it. The mutual fund company must send you an annual report for every year in which you owned units in the fund. Thus, you would get 7, 2 and 1 annual reports respectively, in the above cases.

Use DateDiff with the "q" parameter to find the number of quarters (3 month periods) difference between two dates.

```
DateDiff ("q", #10/6/1999#, #5/20/2003#)
```

Returns 14.

```
DateDiff ("q", #3/31/1999#, #4/1/1999#)
```

Returns 1. The two dates are in adjacent quarters.

```
DateDiff ("q", #1/1/1999#, #3/31/1999#)
```

Returns 0. The two dates are in the same quarter.

Suppose the mutual fund company in the "yyyy" example mailed out quarterly reports. Then it would need to mail out 15, 2 and 1 quarterly reports respectively, in the above cases.

Use DateDiff with the "m" parameter to find the number of months difference between two dates.

```
DateDiff ("m", #3/15/1999#, #7/13/1999#)
```

Returns 4.

Use DateDiff with the "w" parameter to calculate the number of weeks between two dates. For example, if startDateTime is on a Tuesday, DateDiff counts the number of Tuesdays between startDateTime and endDateTime not including the initial Tuesday of startDateTime. Note however that it counts endDateTime if endDateTime is on a Tuesday.

```
DateDiff ("w", #10/19/1999#, #10/25/1999#)
```

Returns 0.

```
DateDiff ("w", #10/19/1999#, #10/26/1999#)
```

Returns 1.

Use DateDiff with the "ww" parameter to calculate the number of firstDayOfWeek's occurring between two dates. For the DateDiff function, the "ww" parameter is the only one that makes use of the firstDayOfWeek argument. It is ignored for all the other interval type parameters. For example, if firstDayOfWeek is crWednesday, it counts the number of Wednesday's between startDateTime and endDateTime. It does not count startDateTime even if startDateTime falls on a Wednesday, but it does count endDateTime if endDateTime falls on a Wednesday. For the examples, note that October 6, 13, 20 and 27 all fall on a Wednesday in 1999.

```
DateDiff ("ww", #10/5/1999#, #10/29/1999#, crWednesday)
```

Returns 4.

```
DateDiff ("ww", #10/6/1999#, #10/29/1999#, crWednesday)
```

Returns 3.

```
DateDiff ("ww", #10/5/1999#, #10/27/1999#, crWednesday)
```

Returns 4.

For example, suppose you want to calculate the number of days between the order date and ship date, excluding Saturdays and Sundays:

Rem Basic syntax

Dim d1, d2

d1 = {Orders.Order Date}

d2 = {Orders.Ship Date}

formula = DateDiff("d", d1, d2) - _

DateDiff("ww", d1, d2, crSaturday) - _

DateDiff("ww", d1, d2, crSunday)

//Crystal syntax

Local DateTimeVar d1 := {Orders.Order Date};

Local DateTimeVar d2 := {Orders.Ship Date};

DateDiff ("d", d1, d2) -

DateDiff ("ww", d1, d2, crSaturday) -

DateDiff ("ww", d1, d2, crSunday)

Comment

This function is designed to work like the Visual Basic function of the same name.

DatePart

Basic and Crystal syntax.

Overloads

- DatePart (intervalType, inputDateTime)
- DatePart (intervalType, inputDateTime, firstDayOfWeek)
- DatePart (intervalType, inputDateTime, firstDayOfWeek, firstWeekOfYear)

Arguments

- intervalType a String expression that specifies the part of a date to be returned. Possible values can be:

Interval type value	Description
yyyy	Extracts the year
q	Quarter (the result is 1, 2, 3 or 4
m	Month (the result is from 1 to 12)
y	Day of year (1 to 365 or 366 in a leap year)
d	Day part of the date (1 to 31)
	Year
w	Day of week (1 to 7 with the result depending on firstDayOfWeek)
ww	Week of year (1 to 53 with firstDayOfWeek and firstWeekOfYear determining the exact days of the first calendar week of the year)
h	Extracts the hour part of the given DateTime (0 to 23)
n	Minute part (0 to 59)
s	Second part (0 to 59)

- inputDateTime is the DateTime value whose part will be extracted.
- firstDayOfWeek is an optional constant used to specify the first day of the week. If not specified, crSunday is assumed. See Day of Week constants for more information.
- firstWeekOfYear is an optional constant specifying the first week of the year. If not specified, the first week is assumed to be the one in which Jan. 1 occurs (crFirstJan1). See First Week of Year constants.

Returns

A Number value.

Action

DatePart returns a Number that specifies a given part of a given date.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
DatePart ("d", #August 15, 1999#)
```

Returns 15.

```
DatePart ("m", #August 15, 1999#)
```

Returns 9.

```
DatePart ("n", #10:35 am#)
```

Returns 35.

```
DatePart ("q", #September 29, 1999#)
```

Returns 3 since September 29 is in the third quarter of the year.

```
DatePart ("ww", #September 14, 1997#)
```

Returns 38 since September 14, 1997 is in the 38th week of 1997.

Suppose that a company wants to stamp an id number on each of its shipments for quality control purposes. The id is composed of the order id, followed by a dash, followed by the week the order was shipped, followed by the last two digits of the year. Here is an example formula that produces these ids:

```
Rem Basic syntax

formula = CStr({Orders.Order ID}, 0, "") & "-" & _

CStr(DatePart("ww", {Orders.Ship Date}), 0) & _

CStr({Orders.Ship Date}, "yy")


//Crystal syntax

CStr({Orders.Order ID}, 0, "") & "-" &

CStr(DatePart("ww", {Orders.Ship Date}), 0) &

CStr({Orders.Ship Date}, "yy")
```

Returns the string "2082897" if {Orders.Order ID} is 2082 and {Orders.Ship Date} is September 14, 1997.

Comments

This function is designed to work like the Visual Basic function of the same name.

The DatePart function with "yyyy" intervalType argument is the same as the Year function. Similarly, the DatePart function with "m", "d", "w", "h", "n" and "s" intervalType argument is the same as the functions Month, Day, Weekday (or DayOfWeek), Hour, Minute and Second respectively. On the other hand, there is no easy alternative to using the DatePart function for the "q", "y" and "ww" intervalType arguments.

The firstDayOfWeek argument affects the DatePart function when the interval type argument is "w" or "ww". For all other intervalType argument values, it is ignored.

The firstWeekOfYear argument affects the DatePart function only when the intervalType argument is "ww". For all other intervalType argument values, it is ignored.

First Day of Week/Year constants

Day of Week constants

Day of Week enclosure constants

First Week of Year constants

Date Range functions

Date range functions are preset date ranges that can be used in formulas.

WeekToDateFromSun

MonthToDate

YearToDate

Last7Days

Last4WeeksToSun

LastFullWeek

LastFullMonth

AllDatesToToday

AllDatesToYesterday

AllDatesFromToday

AllDatesFromTomorrow

Aged0To30Days, Aged31To60Days, Aged61To90Days

Over90Days

Next30Days, Next31To60Days, Next61To90Days, Next91To365Days

Calendar1stQtr, Calendar2ndQtr, Calendar3rdQtr, Calendar4thQtr

Calendar1stHalf, Calendar2ndHalf

LastYearMTD

LastYearYTD

WeekToDateFromSun

Basic and Crystal syntax.

Returns

None

Action

Specifies a range of Date values that includes all days from last Sunday to Today (including today).

Examples

The following examples are applicable to Crystal syntax:

If {orders.ORDER DATE} in WeekToDateFromSun Then

{orders.ORDER AMOUNT}

Else

0

Returns the Amount if the date falls within the range of last Sunday to Today (including today). The following table identifies the first and last dates included in the date range if today equals 04/13/95.

If today = 04/13/95	{orders.ORDER DATE}	{orders.ORDER AMOUNT}
not in range (previous Saturday)	04/08/95	0
first date included (previous Sunday)	04/09/95	1252.24
	04/11/95	1597.87
	04/12/95	2546.35
last date included	04/13/95	5462.21

Note: If you would like to base the date range on a date other than today's date, you must either change the date via the Date/Time option in the Windows Control Panel, or change the report print date via the Set Print Date/Time command in the Report Menu.

MonthToDate

Basic and Crystal syntax.

Returns

None

Action

Specifies a range of Date values that includes all dates from the first day of the month to today.

Examples

The following examples are applicable to Crystal syntax:

If {orders.ORDER DATE} in MonthToDate Then

{orders.ORDER AMOUNT}

Else

0

Returns the Amount if the date falls on any date from the first day of the month to today. The following table identifies the first and last dates included in the date range if today equals 04/10/98.

If today = 04/10/98	{orders.ORDER DATE}	{orders.ORDER AMOUNT}
not in range	03/31/98	0

first date included	04/01/98	1252.24
	04/05/98	1597.87
	04/09/98	2546.35
last date included	04/10/98	5462.21

Note: If you would like to base the date range on a date other than today's date, you must either change the date via the Date/Time option in the Windows Control Panel, or change the report print date via the Set Print Date/Time command in the Report Menu.

YearToDate

Basic and Crystal syntax.

Returns

None

Action

Specifies a range of Date values that includes all days from the first day of the calendar year to today.

Examples

The following examples are applicable to Crystal syntax:

If {orders.ORDER DATE} in YearToDate Then

{orders.ORDER AMOUNT}

Else

0

Returns the Order Amount if the date falls within the range of the first day of the calendar year to today. The following table identifies the first and last dates included in the date range if today equals 04/10/98.

If today = 04/10/98	{orders.ORDER DATE}	{orders.ORDER AMOUNT}
not in range	12/31/97	0
first date included	01/01/98	1252.24
	02/28/98	1597.87
	03/15/98	2546.35
last date included	04/10/98	5462.21

Note: If you would like to base the date range on a date other than today's date, you must either change the date via the Date/Time option in the Windows Control Panel, or change the report print date via the Set Print Date/Time command in the Report Menu.

Last7Days

Basic and Crystal syntax.

Returns

None

Action

Specifies a range of Date values that includes all dates from seven days ago to today (including today).

Examples

The following examples are applicable to Crystal syntax:

If {orders.ORDER DATE} in Last7Days Then

{orders.ORDER AMOUNT}

Else

0

Returns the Amount if the date falls on any date from seven days ago to today (including today). The following table identifies the first and last dates included in the date range if today equals 04/10/98.

If today = 04/10/98	{orders.ORDER DATE}	{orders.ORDER AMOUNT}
not in range	04/03/98	0
first date included	04/04/98	1252.24
	04/06/98	1597.87
	04/08/98	2546.35
last date included	04/10/98	5462.21

Note: If you would like to base the date range on a date other than today's date, you must either change the date via the Date/Time option in the Windows Control Panel, or change the report print date via the Set Print Date/Time command in the Report Menu.

Last4WeeksToSun

Basic and Crystal syntax.

Returns

None

Action

Specifies a range of dates that includes the four weeks previous to last Sunday. A week begins on a Monday and ends on a Sunday. For example, September, 1996, begins on a Sunday.

- If today is September 22 (Sunday), Last4WeeksToSun begins on August 26 (Monday) and ends on September 22 (today, Sunday).
- If today is September 28 (a Saturday), Last4WeeksToSun begins on August 26 (a Monday) and ends on September 22 (the previous Sunday).

Examples

The following examples are applicable to Crystal syntax:

If {orders.ORDER DATE} in Last4WeeksToSun Then

{orders.ORDER AMOUNT}

Else

0

Returns the Amount if the date falls within the last 4 weeks to Sunday (See Action section for specifics). The following table identifies the first and last dates included in the date range if today equals 04/10/95 (Monday).

If today = 04/10/95	{orders. ORDER DATE}	{orders. ORDER AMOUNT}
not in range	03/12/95	0
first date included	03/13/95	1252.24
	03/27/95	1597.87
	04/05/95	2546.35
last date included (Sunday before current date)	04/09/95	5462.21

Note: If you would like to base the date range on a date other than today's date, you must either change the date via the Date/Time option in the Windows Control Panel, or change the report print date via the Set Print Date/Time command in the Report Menu.

LastFullWeek

Basic and Crystal syntax.

Returns

None

Action

Specifies a range of Date values that includes all dates from Sunday to Saturday of the previous week.

Examples

The following examples are applicable to Crystal syntax:

If {orders.ORDER DATE} in LastFullWeek Then

{orders.ORDER AMOUNT}

Else

0

Returns the Amount if the date falls on any date from Sunday to Saturday of the last full week. The following table identifies the first and last dates included in the date range if today equals 04/10/98.

If today = 04/10/98	{orders.ORDER DATE}	{orders.ORDER AMOUNT}
not in range	03/28/98	0
first date included	03/29/98	1252.24
last date included	04/04/98	5462.21

Note: If you would like to base the date range on a date other than today's date, you must either change the date via the Date/Time option in the Windows Control Panel, or change the report print date via the Set Print Date/Time command in the Report Menu.

LastFullMonth

Basic and Crystal syntax.

Returns

None

Action

Specifies a range of Date values that includes all dates from the first to last day of the previous month.

Examples

The following examples are applicable to Crystal syntax:

If {orders.ORDER DATE} in LastFullMonth Then

{orders.ORDER AMOUNT}

Else

0

Returns the Amount if the date falls on any date from the previous month. The following table identifies the first and last dates included in the date range if today equals 04/10/98.

If today = 04/10/98	{orders.ORDER DATE}	{orders.ORDER AMOUNT}
not in range	02/28/98	0
first date included	03/01/98	1252.24
	03/15/98	1597.87
	03/27/98	2546.35
last date included	03/31/98	5462.21

Note: If you would like to base the date range on a date other than today's date, you must either change the date via the Date/Time option in the Windows Control Panel, or change the report print date via the Set Print Date/Time command in the Report Menu.

AllDatesToToday

Basic and Crystal syntax.

Returns

None

Action

Specifies a range of Date values that includes every day up to and including the present day.

Examples

The following examples are applicable to Crystal syntax:

If {orders.ORDER DATE} in AllDatesToToday Then

{orders.ORDER AMOUNT}

Else

0

Returns the Amount if the date falls on any date from the first date entered to today (including today). The following table identifies the first and last dates included in the date range if today equals 04/10/98.

If today = 04/10/98	{orders.ORDER DATE}	{orders.ORDER AMOUNT}
first date included (earliest date in field)	12/10/97	1252.24
	01/05/98	1597.87
	03/05/98	2546.35
last date included	04/10/98	5462.21
not in range	04/11/98	0

Note: If you would like to base the date range on a date other than today's date, you must either change the date via the Date/Time option in the Windows Control Panel, or change the report print date via the Set Print Date/Time command in the Report Menu.

AllDatesToYesterday

Basic and Crystal syntax.

Returns

None

Action

Specifies a range of Date values that includes every day up through the previous day. AllDatesToYesterday includes all dates before today, but does not include the present day.

Examples

The following examples are applicable to Crystal syntax:

If {orders.ORDER DATE} in AllDatesToYesterday Then

{orders.ORDER AMOUNT}

Else

0

Returns the Amount if the date falls on any date from the first date entered to yesterday. The following table identifies the first and last dates included in the date range if today equals 04/10/98.

If today = 04/10/98	{orders.ORDER DATE}	{orders.ORDER AMOUNT}
first date included (earliest date in field)	12/10/97	1252.24
	01/05/98	1597.87
	03/05/98	2546.35
last date included	04/09/98	5462.21
not in range	04/10/98	0

Note: If you would like to base the date range on a date other than today's date, you must either change the date via the Date/Time option in the Windows Control Panel, or change the report print date via the Set Print Date/Time command in the Report Menu.

AllDatesFromToday

Basic and Crystal syntax.

Returns

None

Action

Specifies a range of Date values that includes any date from the present day to any future Date value that may appear in a field. AllDatesFromToday includes the present day.

Examples

The following examples are applicable to Crystal syntax:

If {orders.ORDER DATE} in AllDatesFromToday Then

{orders.ORDER AMOUNT}

Else

0

Returns the Amount if the date falls on any date from today (including today) to all future dates. The following table identifies the first and last dates included in the date range if today equals 04/10/98.

If today = 04/10/98	{orders.ORDER DATE}	{orders.ORDER AMOUNT}
not in range	04/09/98	0

first date included	04/10/98	1252.24
	07/08/98	1597.87
	09/24/98	2546.35
last date included (latest date in field)	10/08/98	5462.21

Note: If you would like to base the date range on a date other than today's date, you must either change the date via the Date/Time option in the Windows Control Panel, or change the report print date via the Set Print Date/Time command in the Report Menu.

AllDatesFromTomorrow

Basic and Crystal syntax.

Returns

None

Action

Specifies a range of Date values that fall after the present day. AllDatesFromTomorrow does not include the present day, but does include any future date.

Examples

The following examples are applicable to Crystal syntax:

If {orders.ORDER DATE} in AllDatesFromTomorrow Then

{orders.ORDER AMOUNT}

Else

0

Returns the Amount if the date falls on any date from tomorrow to all future dates. The following table identifies the first and last dates included in the date range if today equals 04/10/98.

If today = 04/10/98	{orders.ORDER DATE}	{orders.ORDER AMOUNT}
not in range	04/10/98	0
first date included	04/11/98	1252.24
	07/08/98	1597.87
	09/24/98	2546.35

last date included (latest date in field)	10/08/98	5462.21

Note: If you would like to base the date range on a date other than today's date, you must either change the date via the Date/Time option in the Windows Control Panel, or change the report print date via the Set Print Date/Time command in the Report Menu.

Aged0To30Days, Aged31To60Days, Aged61To90Days

Basic and Crystal syntax.

Returns

None

Action

Specifies a range of Date values that fall within a certain time period before the present date. If the current date is 12/30/98, Aged0To30Days specifies the period from 12/1/98 to the present date, Aged31To60Days specifies the period from 11/1/98 to 11/30/98, and Aged61To90Days specifies the period 10/2/98 to 10/31/98.

Examples

The following examples are applicable to Crystal syntax:

If {orders.ORDER DATE} in Aged0To30Days Then

{orders.ORDER AMOUNT}

Else

0

Returns the Amount if the date falls within the previous 30 days from today's date. The following table identifies the first and last dates included in the date range if today equals 04/10/98.

If today = 04/10/98	{orders.ORDER DATE}	{orders.ORDER AMOUNT}
not in range	03/10/98	0
first date included	03/11/98	1252.24
	03/15/98	1597.87
	04/05/98	2546.35
last date included	04/10/98	5462.21

If {orders.ORDER DATE} in Aged31To60Days Then

{orders.ORDER AMOUNT}

Else

0

Returns the Amount if the date falls within the previous 31 to 60 days from today's date. The following table identifies the first and last dates included in the date range if today equals 04/10/98.

If today = 04/10/98	{orders.ORDER DATE}	{orders.ORDER AMOUNT}
not in range	02/08/98	0
first date included	02/09/98	1252.24
	02/25/98	1597.87
	03/05/98	2546.35
last date included	03/10/98	5462.21

Note: If you would like to base the date range on a date other than today's date, you must either change the date via the Date/Time option in the Windows Control Panel, or change the report print date via the Set Print Date/Time command in the Report Menu.

Over90Days

Basic and Crystal syntax.

Returns

None

Action

Specifies a range of Date values that includes all values that are more than 90 days older than the current date.

Examples

The following examples are applicable to Crystal syntax:

If {orders.ORDER DATE} in Over90Days Then

{orders.ORDER AMOUNT}

Else

0

Returns the Amount if the date falls in the range of over 90 days from today's date. The following table identifies the first and last dates included in the date range if today equals 04/10/98.

If today = 04/10/98	{orders.ORDER DATE}	{orders.ORDER AMOUNT}
first date included (earliest date in field)	12/10/97	1252.24
	12/31/97	1597.87
	01/05/98	2546.35
last date included	01/09/98	5462.21
not in range	01/10/98	0

Note: If you would like to base the date range on a date other than today's date, you must either change the date via the Date/Time option in the Windows Control Panel, or change the report print date via the Set Print Date/Time command in the Report Menu.

Next30Days, Next31To60Days, Next61To90Days, Next91To365Days

Basic and Crystal syntax.

Returns

None

Action

Specifies a range of Date values that includes all dates in the period specified starting from today (including today).

Examples

The following examples are applicable to Crystal syntax:

If {orders.ORDER DATE} in Next30Days Then

{orders.ORDER AMOUNT}

Else

0

Returns the Amount if the date falls anywhere in the next 30 days starting from today (including today). The following table identifies the first and last dates included in the date range if today equals 04/10/98.

If today = 04/10/98	{orders.ORDER DATE}	{orders.ORDER AMOUNT}

not in range	04/09/98	0
first date included	04/10/98	1252.24
	04/30/98	1597.87
	05/05/98	2546.35
last date included	05/10/98	5462.21

If {orders.ORDER DATE} in Next31To60Days Then

{orders.ORDER AMOUNT}

Else

0

Returns the Amount if the date falls anywhere in the next 31 to 60 days starting from today. The following table identifies the first and last dates included in the date range if today equals 04/10/98.

If today = 04/10/98	{orders.ORDER DATE}	{orders.ORDER AMOUNT}
not in range	05/10/98	0
first date included	05/11/98	1252.24
	05/30/98	1597.87
	06/03/98	2546.35
last date included	06/09/98	5462.21

Note: If you would like to base the date range on a date other than today's date, you must either change the date via the Date/Time option in the Windows Control Panel, or change the report print date via the Set Print Date/Time command in the Report Menu.

Calendar1stQtr, Calendar2ndQtr, Calendar3rdQtr, Calendar4thQtr

Basic and Crystal syntax.

Returns

None

Action

Specifies a range of Date values that falls within the 1st, 2nd, 3rd, or 4th quarter of the current calendar year. The first quarter of the calendar year includes all dates from January 1st through March 31st. The second quarter of the calendar year includes all dates from April 1st through June 30th. The third quarter of the calendar year includes all dates from July 1st through September 30th. The fourth quarter of the calendar year includes all dates from October 1st through December 31st.

Examples

The following examples are applicable to Crystal syntax:

If {orders.ORDER DATE} in Calendar1stQtr Then

{orders.ORDER AMOUNT}

Else

0

Returns the Amount if the date falls on any date in the 1st quarter of the calendar year. The following table identifies the first and last dates included in the date range if today equals 04/10/98.

If today = 04/10/98	{orders.ORDER DATE}	{orders.ORDER AMOUNT}
not in range	12/10/97	0
	01/01/97	0
first date included	01/01/98	1252.24
	02/05/98	1597.87
	03/15/98	2546.35
last date included	03/31/98	5462.21

Calendar1stHalf, Calendar2ndHalf

Basic and Crystal syntax.

Returns

None

Action

Specifies a range of Date values that includes all dates that fall within the first or second half of the current calendar year, respectively. The first half of the calendar year includes all dates from

January 1st through June 30th. The second half of the calendar year includes all dates from July 1st through December 31st.

Examples

The following examples are applicable to Crystal syntax:

If {orders.ORDER DATE} in Calendar1stHalf Then

{orders.ORDER AMOUNT}

Else

0

Returns the Amount if the date falls on any date in the 1st half of the calendar year. The following table identifies the first and last dates included in the date range if today equals 04/10/98.

If today = 04/10/98	{orders.ORDER DATE}	{orders.ORDER AMOUNT}
not in range	12/10/97	0
	01/01/97	0
first date included	01/01/98	1252.24
	03/05/98	1597.87
	05/15/98	2546.35
last date included	06/30/98	5462.21

LastYearMTD

Basic and Crystal syntax.

Returns

None

Action

Specifies a range of Date values in the previous year that matches the current month to date.

Examples

The following examples are applicable to Crystal syntax:

If {orders.ORDER DATE} in LastYearMTD Then

{orders.ORDER AMOUNT}

Else

0

Returns the Amount if the date falls on any date in the current month last year, up to the current date last year. The following table identifies the first and last dates included in the date range if today equals 04/10/98.

If today = 04/10/98	{orders.ORDER DATE}	{orders.ORDER AMOUNT}
not in range	03/31/97	0
first date included	04/01/97	1252.24
	04/05/97	1597.87
	04/08/97	2546.35
last date included	04/10/97	5462.21

Note: If you would like to base the date range on a date other than today's date, you must either change the date via the Date/Time option in the Windows Control Panel, or change the report print date via the Set Print Date/Time command in the Report Menu.

LastYearYTD

Basic and Crystal syntax.

Returns

None

Action

Specifies a range of Date values that includes all dates in the last year, up to the current date last year.

Examples

The following examples are applicable to Crystal syntax:

If {orders.ORDER DATE} in LastYearYTD Then

{orders.ORDER AMOUNT}

Else

0

Returns the Amount if the date falls on any date in the last year, up to the current date last year. The following table identifies the first and last dates included in the date range if today equals 04/10/98.

If today = 04/10/98	{orders.ORDER DATE}	{orders.ORDER AMOUNT}
not in range	12/31/96	0
first date included	01/01/97	1252.24
	02/16/97	1597.87
	03/24/97	2546.35
last date included	04/10/97	5462.21

Note: If you would like to base the date range on a date other than today's date, you must either change the date via the Date/Time option in the Windows Control Panel, or change the report print date via the Set Print Date/Time command in the Report Menu.

Arrays

Array (x,...)

MakeArray (x, ...)

UBound (array)

Array summary functions (x)

SummaryFunction (x)

Array (x,...)

Basic syntax.

Array and MakeArray (x, ...) are equivalent functions. However, Array cannot be used in Crystal syntax since it is a keyword for declaring array variables in that syntax.

Arguments

The given arguments are array elements with which to initialize the array. The number of array elements can be any number between 1 and 1000. All the arguments must be of the same type. Unlike Visual Basic, you may not have 0 number of elements.

Returns

Array of values.

Action

The Array function creates an array, initializes it with the given array elements and returns it.

Typical use

To create and initialize an array variable in Basic syntax.

In Crystal syntax, you can use the square bracket notation for arrays instead.

Example

The following example is applicable to Basic syntax:

The following sets up an array of days for a week, starting with Monday:

Dim week, firstDay, lastDay

week = Array("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", _

"Saturday", "Sunday")

Rem firstDay contains "Monday"

firstDay = week(1)

Rem lastDay contains "Sunday"

lastDay = week(7)

formula = firstDay

Returns "Monday".

Comments

This function is designed to work like the Visual Basic function of the same name.

MakeArray (x, ...)

Basic and Crystal syntax.

Array (x,...) and MakeArray are equivalent functions. However, Array cannot be used in Crystal syntax since it is a keyword for declaring array variables in that syntax.

Arguments

The given arguments are array elements with which to initialize the array. The number of array elements can be any number between 1 and 1000. All the arguments must be of the same type.

Returns

Array of values.

Action

MakeArray creates an array, initializes it with the given array elements and returns it.

Typical use

To create and initialize an array variable, especially in Basic syntax.

In Crystal syntax, you can use the square bracket notation for arrays instead.

Example

The following example is applicable to Crystal syntax:

```
Local NumberVar Array x := MakeArray (1, 1000, 10, 100, 10);
```

Creates and initializes an array x with five Number elements: 1, 1000, 10, 100, 10.

UBound (array)

Basic and Crystal syntax.

Argument

array is an array value, expression or variable.

Returns

Number value.

Action

UBound returns a Number containing the largest available subscript for the given array.

Typical use

Commonly used to obtain the size of an array before using a looping control mechanism to systematically manipulate elements of the array.

Examples

The following examples are for both Basic and Crystal syntax:

```
Rem Basic syntax

Dim simpleArray(10) As Number

formula = UBound(simpleArray)


//Crystal syntax

Local NumberVar Array simpleArray;

Redim simpleArray[10];

UBound(simpleArray)
```

Returns 10.

```
Rem Basic syntax

Dim dateArray () As Date

dateArray = Array (CDate(#12/25/1998#), CDate(#12/24/1999#))

formula = UBound(dateArray)


//Crystal syntax

Local DateVar Array dateArray;

dateArray := [CDate(#12/25/1998#), CDate(#12/24/1999#)];

UBound (dateArray)
```

Returns 2.

Comments

- This function is designed to work like the Visual Basic function of the same name.

  Note: Unlike Visual Basic, only 1 dimensional arrays are supported in Crystal Reports. Also, array subscripts start at 1 in Crystal Reports and not 0 as is the default in Visual Basic.

- UBound returns 1 for an uninitialized array variable, however, it causes an error to subscript an uninitialized array variable. The reason for this behavior is that at present Crystal Reports does not support arrays with zero elements. To work around this problem, always initialize your array variables. For example, initialize Global array variables in a formula located in the report header and local array variables within the formula itself.

Array summary functions (x)

Average (x)

See Average (fld), Average (fld, condFld), Average (fld, condFld, cond), Average (x)

Count (x)

See Count (fld), Count (fld, condFld), Count (fld, condFld, cond), Count (x).

DistinctCount

See DistinctCount (fld), DistinctCount (fld, condFld), DistinctCount (fld, condFld, cond), DistinctCount (x)

Minimum

See Minimum (fld), Minimum (fld, condFld), Minimum (fld, condFld, cond), Minimum (x).

PopulationStdDev (x)

See PopulationStdDev (fld), PopulationStdDev (fld, condFld), PopulationStdDev (fld, condFld, cond), PopulationStdDev (x)

PopulationVariance (x)

See PopulationVariance (fld), PopulationVariance (fld, condFld), PopulationVariance (fld, condFld, cond), PopulationVariance (x).

StdDev (x)

See StdDev (fld), StdDev (fld, condFld), StdDev (fld, condFld, cond), StdDev (x).

Sum (x)

See Sum (fld), Sum (fld, fld), Sum (fld, condFld), Sum (fld, condFld, cond), Sum (x)

Variance

See Variance (fld), Variance (fld, condFld), Variance (fld, condFld, cond), Variance (x).

SummaryFunction (x)

Basic and Crystal syntax.

Arguments

x is an array of values that can be evaluated by the function being used.

Returns

Fractional Number

Action

SummaryFunction (x) summarizes the values in an array of constants, data field values, or formulas (a*b, c/d, etc.) separated by commas.

Typical uses

Use any time you want to summarize two or more values.

Examples

The following examples are applicable to Crystal syntax:

Note: Use the Array (x,...) function to create the arrays instead of square brackets to make Basic syntax examples.

```
SummaryFunction([25,50,75,100])
```

Summarizes the values in an array of constants.

```
SummaryFunction([{file.PERIOD 8}, {file.PERIOD 9}, {file.PERIOD 10}])
```

Summarizes the values in an array of field values.

```
Average ([10,12,32,48])
```

Calculates the average of the values in an array of constants.

```
Count([1,2,3,4,5])
```

Returns 5. Counts the total number of values in the array.

```
DistinctCount([1,3,5,3,2,5])
```

Returns 4. Counts the number of distinct values in the array. Duplicate values are ignored.

```
Variance([{file.QTY1}, {file.QTY2}, {file.QTY3}, {file.QTY4}])
```

Returns 17.00 where Qty1 = 2, Qty2 = 2, Qty3 = 10, and Qty4 = 8.

Type Conversion functions

These functions allow you to convert from one data type to another.

CBool (number or currency)

CCur

CDbl

CStr

CDate

CTime

CDateTime

CBool (number or currency)

Basic and Crystal syntax.

Argument

A single argument that can be either a Number or Currency value or expression.

Returns

Boolean value

Action

CBool returns True if the argument is positive or negative but not 0 and returns False if the argument is 0.

Example

The following example is applicable to Basic and Crystal syntax:

```
CBool({Item.NETSALES})
```

Returns True if there is an increase or decrease in net sales; returns False if net sales is 0.

Comment

This function is designed to work like the Visual Basic function of the same name.

CCur

Basic and Crystal syntax.

Overloads

- CCur (number or currency)
- CCur (string)

Arguments

CCur (number or currency)	number or currency is a Number or Currency value or expression.
CCur (string)	string is a String value or expression.

Returns

Currency value

Action

CCur returns a Currency value created by converting the given Number, Currency or String type argument to a Currency value. If the conversion cannot be done, an error occurs.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
CCur (200.20)
```

In Crystal syntax this is equivalent to writing $200.20. In Basic syntax, CCur (200.20) is the way to directly create a Currency value of 200.20.

```
CCur ("$200.20")
```

Returns the Currency value CCur (200.20) given the String value "$200.20".

```
CCur ("two hundred dollars")
```

Causes an error to occur. The CCur function cannot convert the String "two hundred dollars" to a Currency value.

Comments

- This function is designed to work like the Visual Basic function of the same name.
- You can use the IsNumeric (str) or NumericText (str) function to check if the String argument can be converted to a Currency before doing the actual conversion. That way, if the conversion cannot be done, you can handle the situation appropriately.
- In Crystal syntax, you can use the To currency operator operator instead of CCur when converting from a Number value to a Currency value.

CDbl

Basic and Crystal syntax.

CDbl and ToNumber (numeric), ToNumber (string), ToNumber (Boolean) are equivalent functions.

Overloads

- CDbl (number or currency)
- CDbl (string)

Arguments

CDbl (number or currency)	number or currency is a Number or Currency value or expression.
CDbl (string)	string is a String value or expression.

Returns

Number value

Action

CDbl returns a Number value created by converting the given Number, Currency or String type argument to a Number value. If the conversion cannot be done, an error occurs.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
CDbl ({Orders.Order Amount})
```

Returns the value of the Currency field {Orders.Order Amount} converted to Number type.

```
CDbl ("$120.20")
```

Returns the Number value of 120.2.

Comments

- This function is designed to work like the Visual Basic function of the same name.

You can use the IsNumeric (str) or NumericText (str) function to check if the String argument can be converted to a Currency before doing the actual conversion. That way, if the conversion cannot be done, you can handle the situation appropriately.

CStr

Basic and Crystal syntax.

CStr and ToText are equivalent functions.

Overloads

- CStr (x)
- CStr (x, y)
- CStr (x, y, z)
- CStr (x, y, z, w)
- CStr (x, y, z, w, q)

Arguments

Converting . . .	Description
Boolean values	• x is a Boolean value that is converted to a String, either "True" or "False".

Number and Currency values	• x is a Number or Currency value to be converted into a text string; it can be a whole or fractional value. • y is a whole number indicating the number of decimal places to carry the value in x to (This argument is optional.). • z is a single character text string indicating the character to be used to separate thousands in x. Default is the character specified in your International or Regional settings control panel. (This argument is optional.) • w is a single character text string indicating the character to be used as a decimal separator in x. Default is the character specified in your International or Regional settings control panel. (This argument is optional.)
Number and Currency values (formatting)	• x is a Number or Currency value to be converted into a text string; it can be a whole or fractional value. • y is a text string used to indicate the format for displaying the value in x. See Format Strings for information on creating a format string. • z is a whole number indicating the number of decimal places to carry the value in x to. (This argument is optional.) • w is a single character text string indicating the character to be used to separate thousands in x. Default is the character specified in your International or Regional settings control panel. (This argument is optional.) • q is a single character text string indicating the character to be used as a decimal separator in x. The default is the character specified in your International or Regional settings control panel. (This argument is optional.)
Date values	• x is a Date value to be converted into a text string. • y is a text string that defines how the value in x is to be formatted. See Format strings for Date, Time, and DateTime values for more information on creating this format string. (This argument is optional.)
Time values	• x is a Time value to be converted into a text string. • y is a text string that defines how the value in x is to be formatted. See Format strings for Date, Time, and DateTime values for more information on creating this format string. (This argument is optional.) • z is a text string to be used as a label for A.M. (morning) hours. (This argument is optional.) • w is a text string to be used as a label for P.M. (evening) hours. (This argument is optional.)

DateTime values	• x is a DateTime value to be converted into a text string. • y is a text string of characters that indicate how the resulting text string will be formatted. See Format strings for Date, Time, and DateTime values for more information on creating a format string. (This argument is optional.) • z is a text string to be used as a label for A.M. (morning) hours. (This argument is optional.) • w is a text string to be used as a label for P.M. (evening) hours. (This argument is optional.)

Returns

Text String

Action

The CStr function converts Numbers, Currency, Date, Time, and DateTime values to text strings.

Typical uses

Use this function to convert a Number, Currency, Date, Time, or DateTime value to a text string to appear as text in your report (form letters, comments, etc.).

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
CStr ({Orders.Shipped})
```

Returns "True" if the value of {Orders.Shipped} is True.

```
CStr(123.45)
```

Returns "123.45".

```
CStr(12345.6749,2)
```

Returns "12345.67".

```
CStr(12345.6750,2)
```

Returns "12345.68".

```
CStr(12345.4999,0)
```

Returns "12345".

```
CStr(12345.5000,0)
```

Returns "12346".

```
CStr({file.AMT} * {file.QUANTITY})
```

Returns "44,890.20" where Amt = 24.45 and Quantity = 1836.

CStr is useful when you want to build a sentence by combining (concatenating) a converted number or other value with other text strings:

```
"The base price of item # " + {file.ITEM NUMBER} + " is $" + CStr
({file.BASE PRICE}) + "."
```

Prints the sentence "The base price of item A1/4520/B12 is $50.00." where the Item number is A1/4520/B12 and the Base Price is 50.00, converted to text and formatted with two decimal places.

```
CStr(CDate(1996, 11, 1), "yy MMM dd, dddd")
```

Returns "96 Nov 01, Monday".

```
CStr(CDateTime(1995,10,12,3,30,11),"HH:mm, yy MMMM ddd")
```

Returns "03:30, 95 October Mon".

```
CStr(CTime(12, 10, 10), "HH*mm*ss tt", "amStr", "pmStr")
```

Returns "12*10*10 pmStr".

Comments

The overloads of CStr that take only one argument work like the Visual Basic function of the same name.

Converting Boolean values:

- The CStr function, when used with Boolean values, is most useful for combining (concatenating) a Boolean value with other text. Otherwise, a Boolean field can be formatted to appear as True or False in your report simply by changing the format on the Boolean Tab of the Format Editor.

Converting numbers and currency values:

- If the number of decimal places is specified, this function does not truncate the number when converted to text, but rounds it to the number of decimal places specified. See Round (x), Round (x, #places), for more information on the rounding procedure.

Converting Date, Time, and DateTime values:

- Any character, with the exception of the Date or Time format characters, can be used within the format string. For example, you may wish to use a slash to separate the different elements (month, day, year) of the date, as in "12/30/95", or you may wish to use a colon to separate the different elements (hours, minutes, seconds) of the time, as in "12:30:10".
- If you wish to use any of the above characters in the format string, they must appear in quotes. For example: CStr(CDateTime(1995,10,12,13,20,22), "MM/dd/yy hh 'h' mm 'min' ss 'sec' ", 'am', 'pm') = "10/12/95 1 h 20 min 22 sec pm"

Passing optional arguments:

- Many arguments for the CStr function have been specified as optional. However, you can only leave an argument blank if all the arguments that follow it are left blank as well. In other words, you can not leave the y and z arguments blank and provide an argument for w. It is possible, however, to leave one, two, or all of the optional arguments blank, as long as no arguments are supplied after the blank arguments. The following combinations are possible when supplying arguments to the CStr function:

- CStr (x)
- CStr (x, y)
- CStr (x, y, z)
- CStr (x, y, z, w)
- CStr (x, y, z, w, q)
- Using the "t" or "tt" format characters in a time format string provides default strings for indicating a.m. (morning) and p.m. (evening) hours. "t" produces just a single character, "a" or "p", while "tt" produces the entire string, "am" or "pm". You can pass your own custom strings for indicating am/pm strings. (See the arguments for converting Time and DateTime values with CStr in the Arguments section above.) If you do pass your own am/pm strings, the "t" and "tt" format characters will have the same effect on them (producing single character vs. multiple character strings). However, the "t" and "tt" format characters are optional and only necessary when default am/pm strings are needed.

CDate

Basic and Crystal syntax.

The DateValue and Date functions are equivalent to CDate. However, Date can only be used in Crystal syntax since it is a type name in Basic syntax.

Overloads

- CDate (number)
- CDate (string)
- CDate (dateTime)
- CDate (YYYY, MM, DD)

Arguments

CDate (number)	Number is a value representing the number of days starting from December 30, 1899. It can be positive or negative, and is truncated if fractional.
CDate (string)	string is a text string representing a date, example: "September 20, 1999"; many formats are supported for the string
CDate (dateTime)	dateTime is a DateTime value.
CDate (YY,MM,DD)	• year is a whole number representing a year, example: 1996. • month is a whole number representing a month, example: 12 for December. • day is a whole number representing a day of the month, example: 05.

Returns

Date value

Action

CDate (number) converts and returns a Date given a number which is the number of days starting from December 30, 1899.

CDate (string) converts and returns a Date given a string.

CDate (dateTime) converts and returns a Date given a DateTime value.

CDate (YYYY, MM, DD) uses the given arguments to create a Date value.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
CDate ("Dec. 31, 1999")
```

Returns the Date value for Dec. 31, 1999.

```
CDate (50)
```

Returns the Date value for February 18, 1900.

```
CDate (#Oct. 20, 1999 12:02pm#)
```

Returns the Date value for October 20, 1999.

```
CDate (1930, 7, 30)
```

Returns the Date value for July 30, 1930.

Comments

You can use the IsDate function to check if a String argument can be converted to a Date before doing the actual conversion. That way, if the conversion cannot be done, you can handle the situation appropriately.

Related topics

DateSerial (year, month, day)

CTime

Basic and Crystal syntax.

The TimeValue and Time functions are equivalent to CTime. However, Time can only be used in Crystal syntax since it is a type name in Basic syntax.

Overloads

- CTime (number)
- CTime (string)
- CTime (dateTime)
- CTime (HH, MM, SS)

Arguments

CTime (number)	• Returns a Time value given a number in units of 24 hours • number can be fractional or negative as well
CTime (string)	Returns a Time value that represents the time, given a String expression specifying a time from 0:00:00 (12:00:00 A.M.) to 23:59:59 (11:59:59 P.M.), inclusive.
CTime (dateTime)	DateTime is a DateTime value.
CTime (hour, min, sec)	• Hour is a whole number representing an hour of the day. • Min is a whole number representing a minute. • Sec is a whole number representing a second.

Returns

A Time value.

Action

CTime (number) converts the given number to a Time value; the given number is in units of 24 hours, can be fractional or negative as well.

CTime (string) converts and returns a time given a string that identifies a Time value.

CTime (dateTime) converts and returns a time given a DateTime value.

CTime (hh, mm, ss) converts and returns a time given the arguments of the hour, minute and seconds.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
CTime (.2)
```

```
CTime (1.2)
```

Both return the same value 4:48:00 a.m.

```
CTime (-.2)
```

Returns 7:12:00 pm

```
CTime ("Jan. 6, 1999 2:30pm")
```

Returns 2:30:00 pm

```
CTime (#Feb. 24, 1999 2:40pm#)
```

Returns 2:40:00 pm

```
CTime (18, 30, 30)
```

Returns 6:30:30 pm

Comments

You can use the IsTime function to check if a String argument can be converted to a Time before doing the actual conversion. That way, if the conversion cannot be done, you can handle the situation appropriately.

Related topics

TimeSerial (hour, minute, second)

IsTime

CDateTime

Basic and Crystal syntax.

The DateTimeValue and DateTime functions are equivalent to CDateTime. However, DateTime can only be used in Crystal syntax since it is a type name in Basic syntax.

Overloads

- CDateTime (string)
- CDateTime (number)
- CDateTime (date)
- CDateTime (date, time)
- CDateTime (YYYY, MM, DD)
- CDateTime (YYYY, MM, DD, HH, MM, SS)

Arguments

CDateTime (date)	• date is a Date value.
CDateTime (date, time)	• date is a Date value. • time is a Time value.
CDateTime (number)	• number represents a number of days from December 30, 1899, example: 20 represents 20 days from December, 1899, which is January 19, 1900. • number can be positive, negative or fractional.
CDateTime (string)	• string represents a date and time, example: "September 15, 1999, 10:45a.m."

CDateTime (year, month, day) CDateTime (year, month, day, hour, min, sec)	• year is a whole number representing a calendar year, example: 1996. • month is a whole number representing a month, example: 1 for January. • day is a whole number representing a day of the month, example: 10. • hour is a whole number representing an hour of the day, example: 12. • min is a whole number representing a minute, example: 59. • sec is a whole number representing seconds, example: 30.

Returns

A DateTime value.

Action

CDateTime (string) converts the given string to a DateTime value.

CDateTime (number) converts the given number to a DateTime value, where number holds the number of days since Dec. 30, 1899.

CDateTime (date) uses the given date to create the date portion, assigns 12:00:00 AM for the time portion for the returned DateTime value.

CDateTime (date, time) uses the given Date and Time data types to create a combined DateTime value.

CDateTime (YYYY, MM, DD) uses the given arguments to create the date portion, and assigns 12:00:00 AM for the time portion for the returned DateTime value.

CDateTime (YYYY, MM, DD, HH, MM, SS) uses the given arguments to create a combined DateTime value.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
CDateTime ("10/4/1999 10:20am")
```

Returns the DateTime value October 4, 1999 10:20:00 am

```
CDateTime (12.5)
```

Returns the DateTime value January 11, 1900 12:00:00 pm

```
CDateTime (.5)
```

Returns the DateTime value December 27, 1899 12:00:00 pm

```
CDateTime (CDate ("Dec. 25, 1999"))
```

Returns the DateTime value December 25, 1999 12:00:00 am

```
CDateTime (CDate ("November 10, 1999"), CTime("12:20am"))
```

Returns the DateTime value November 10, 1999 12:20:00 am

```
CDateTime (1945, 8, 21, 0, 0, 0)
```

Returns the DateTime value August 21, 1945 12:00:00 am

```
CDateTime (1945, 8, 21, 10, 0, 0)
```

Returns the DateTime value August 21, 1945 10:00:00 am

Comments

You can use the IsDateTime function to check if a String argument can be converted to a DateTime before doing the actual conversion. That way, if the conversion cannot be done, you can handle the situation appropriately.

Related topics

IsDateTime

Programming shortcut functions

These functions provide shortcut alternatives to using control structures to achieve the same results.

Choose (index, choice1, choice2, ..., choiceN)

IIF (expression, truePart, falsePart)

Switch (expr1, value1, expr2, value2, ..., exprN, valueN)

Choose (index, choice1, choice2, ..., choiceN)

Arguments

- index is a Number or numeric expression that specifies the index of the choice. It should be between 1 and the number of available choices. If it is out of bounds, Choose returns a default value. (The default value returned depends on the type of the choices. For example, if the choices are of Number type, the default value is 0 and if the choices are of String type, the default value is the empty string ("").)
- choice is one of the choices to choose from. All choices must be of the same type. A choice can be any simple type (Number, Currency, String, Boolean, Date, Time or DateTime) or range type (Number Range, Currency Range, String Range, Date Range, Time Range or DateTime Range), but it may not be an array.

Returns

A value from the given list of choices. The type of the returned value is the same as the type of the choices.

Action

Choose returns a value from the list of choices based on the value of index. For example, if index is 1, it returns choice1 and if index is 2 it returns choice2 and so forth.

Typical uses

- Choose can be used instead of the If-then-else operators or the Select control structures in some situations. For example, if you are mapping an index value to a String value, it can be simplest to use Choose.
- One situation where Choose may be better than a control structure is when writing record selection formulas so that they can be pushed down to the database server.

  See Switch (expr1, value1, expr2, value2, ..., exprN, valueN) function for an example of this.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
Choose (3, "Poor", "Fair", "Good", "Excellent")
```

Returns the String "Good".

```
Choose (2, 10 To 20, 100 To 200, 60 To 70)
```

Returns the Number Range value 100 To 200.

If a company has fiscal year starting July 1 (the 3rd calendar quarter) and it wants to print the fiscal quarter of each order date:

Rem Basic syntax

formula = Choose (DatePart ("q", {Orders.Order Date}), _

"third fiscal quarter", "fourth fiscal quarter", _

" first fiscal quarter", "second fiscal quarter")

//Crystal syntax

Choose (DatePart ("q", {Orders.Order Date}),

"third fiscal quarter", "fourth fiscal quarter",

"first fiscal quarter", "second fiscal quarter")

If {Orders.Order Date} is February 18, 1996, then the DatePart function call returns 1 since this date is in the first calendar quarter. Thus, the first choice value, namely the String value "third fiscal quarter" is returned by the formula.

Comments

- This function is designed to work like the Visual Basic function of the same name. A difference is that the function does not return a Variant and thus all the choices must have the same type.
- Every argument of the Choose function is evaluated before the choice is returned. Thus, you should watch out for undesirable side effects when using Choose. For example, if one of the choice expressions results in division by zero, an error will occur, even if that choice is not the choice that is returned by the Choose function.

Related topics

IIF (expression, truePart, falsePart) function

Switch (expr1, value1, expr2, value2, ..., exprN, valueN) function

If-then-else operators

Select statements (Basic syntax)

Select expressions (Crystal syntax)

IIF (expression, truePart, falsePart)

Basic and Crystal syntax.

Argument

- expression is a Boolean expression.
- truePart is the value returned if expression is True. It can be any simple type (Number, Currency, String, Boolean, Date, Time or DateTime) or range type (Number Range, Currency Range, String Range, Date Range, Time Range or DateTime Range), but it may not be an array.
- falsePart is the value returned if expression is False. It must be of the same type as truePart.

Returns

truePart if expression is True and falsePart if expression is False. The type of the returned value is the same as the type of truePart and falsePart.

Action

IIF returns one of two parts, depending on the evaluation of the expression.

Typical uses

- IIF can be used as an alternative to the If/Then/Else control structure in some situations.
- One situation where IIF may be better than a control structure is when writing record selection formulas so that they can be pushed down to the database server. See Switch (expr1, value1, expr2, value2, ..., exprN, valueN) for an example and Using enhanced record selection formulas for an explanation of the techniques involved.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
IIF ({Orders.Order Amount} > 10000, "large order", "standard order")
```

Returns the String value "large order" if {Orders.Order Amount} is more than $10,000 and the String value "standard order" otherwise.

You can insert a formula field that represents the order amount if the payment has been made already, and is 0 otherwise:

Rem Basic syntax

formula = IIF ({Orders.Payment Received}, {Orders.Order Amount}, 0)

//Crystal syntax

IIF ({Orders.Payment Received}, {Orders.Order Amount}, 0)

Comments

- This function is designed to work like the Visual Basic function of the same name. A difference is that the function does not return a Variant and thus truePart and falsePart must have the same type.
- Every argument of the IIF function is evaluated before the result is returned. Thus, you should watch out for undesirable side effects when using IIF. For example, if falsePart results in division by zero, an error will occur, even if expression is True and so truePart is returned.

Related topics

Choose (index, choice1, choice2, ..., choiceN)

Switch (expr1, value1, expr2, value2, ..., exprN, valueN)

Select statements (Basic syntax)

Select expressions (Crystal syntax)

Switch (expr1, value1, expr2, value2, ..., exprN, valueN)

Basic and Crystal syntax.

Arguments

- expr1, expr2, ..., exprN are Boolean expressions.
- value1, value2, ..., valueN are the possible values that may be returned. All the values must be of the same type. A value can be any simple type (Number, Currency, String, Boolean, Date, Time or DateTime) or range type (Number Range, Currency Range, String Range, Date Range, Time Range or DateTime Range), but it may not be an array.

Returns

One of the values in the value list value1, value2, ..., valueN. The type of the returned value is the same as the type of the values.

Action

The argument list for Switch consists of pairs of expressions and values. Switch evaluates the expressions from left to right, and returns the value associated with the first expression to evaluate to True.

For example, if expression1 is True, the Switch returns value1. If expression1 is False and expression2 is True, then Switch returns value2. If expression1 and expression2 are False and expression3 is True, then Switch returns value3.

If all of the expressions are False, then Switch returns a default value. (The default value returned depends on the type of the values in the value list. For example, if the values are of

Number type, the default value is 0 and if the values are of String type, the default value is the empty string ("").)

Typical uses

Switch can be used instead of If-then-else operators or Select statements (Basic syntax) and Select expressions (Crystal syntax) in some situations.

One situation where Switch may be better than a control structure is when writing record selection formulas so that they can be pushed down to the database server. See below for an example as well as Using enhanced record selection formulas for an explanation of the techniques involved.

Examples

The following examples are applicable to both Basic and Crystal syntax:

Suppose that a company wants to classify orders into "large", "medium" or "small" based on the order size:

```
Rem Basic syntax

formula = Switch ({Orders.Order Amount} > 5000, "large", _

{Orders.Order Amount} > 1000, "medium", _

True, "small")


//Crystal syntax

Switch ({Orders.Order Amount} > 5000, "large",

{Orders.Order Amount} > 1000, "medium",

True, "small")
```

If {Orders.Order Amount} is greater than $5,000 then the formula returns the String value "high". Otherwise, if {Orders.Order Amount} is greater than $1,000, the formula returns the String "medium". Otherwise, the formula returns "small".

Here is an example showing how to use the Switch function for writing efficient record selection formulas that can be pushed down to the database server. Writing this formula using If-then-else operators or Select statements (Basic syntax) and Select expressions (Crystal syntax) would not let it be pushed down.

See Using enhanced record selection formulas for further explanation of the techniques involved.

Note: This example is in Crystal syntax since record selection formulas edited with the Formula Editor must be in Crystal syntax only.

Sample report

A more complete version of the example, with 27 Switch conditions, is provided with the sample report named Date Range Formula.rpt.

Sample reports are located in the Crystal Reports directory under `\Samples\En\Reports`.

When you preview the report, you supply a reference order date parameter, {?reference date} and a range condition, {?reference condition} to indicate a range of dates around the reference order date. The report is then previewed with only the records whose order dates fall in the specified range.

The entire record selection is performed on the database server. This is because the Switch function and all its arguments can be evaluated before accessing the database.

Note:

- In this sample report, the Switch function is returning a Date Range value.
- The indentation style of the code makes the Switch function call look similar to a control structure and makes reading the function easier.
- The last expression is the Boolean value True.

  This Switch function call provides for an explicit default value since if none of the other expression are True, the last expression is automatically True, and so the Date Range CDate (1899, 12, 30) To CDate(1899, 12, 30) is returned.

```
//Crystal syntax record selection formula

{Orders.Order Date} in

Switch

(

{reference condition} = "Aged 0 to 30 days",

({reference date} - 30) To {reference date},

{reference condition} = "Aged 31 to 60 days",

({reference date} - 60) To ({reference date} - 31),

{reference condition} = "Aged 61 to 90 days",

({reference date} - 90) To ({reference date} - 61),

True, // provide default handling and specify a valid range

CDate(1899, 12, 30) To CDate(1899, 12, 30)

)
```

Comments

- This function is designed to work like the Visual Basic function of the same name. A difference is that the function does not return a Variant and thus all the values must have the same type.
- Every argument of the Switch function is evaluated before the result is returned. Thus, you should watch out for undesirable side effects when using Switch.

  For example, if one of the values results in division by zero, an error will occur, even if that value is not the value that is returned by the Switch function.

Related topics

Choose (index, choice1, choice2, ..., choiceN)

IIF (expression, truePart, falsePart)

Select statements (Basic syntax)

Select expressions (Crystal syntax)

Evaluation Time functions

Evaluation Time functions can control how data is presented by controlling when it is calculated or presented.

BeforeReadingRecords

WhileReadingRecords

WhilePrintingRecords

EvaluateAfter (x)

BeforeReadingRecords

Basic and Crystal syntax.

Returns

None

Action

Specifies that the formula is to be evaluated before the database records are read.

Typical uses

Formulas are normally evaluated at the following times:

- If no database or group field is included in the formula, the formula is evaluated before the program reads database records.
- If a database field is included in the formula, the formula is evaluated while the program reads database records.
- If a group field, page number field, subtotal, etc., is included in the formula, the formula is evaluated after database records are read and while the data from the records is being printed in the report.

BeforeReadingRecords forces the formula to be evaluated before the program reads database records. When this function is used in a formula, the Formula Checker returns an error message if you attempt to include elements in the formula (database fields, groups, etc.) that must be evaluated at a later time (while reading or while printing records).

Examples

The following example is applicable to Crystal syntax:

BeforeReadingRecords;

ToNumber ("12345")

Forces the formula (which contains no database or group fields), to be evaluated before reading records.

Note: If you try to include a database field or group in this formula, you will get an error message.

WhileReadingRecords

Basic and Crystal syntax.

Returns

None

Action

Forces the program to evaluate the formula while it is reading database record data.

Typical uses

Formulas are normally evaluated at the following times:

- If no database or group field is included in the formula, the formula is evaluated before the program reads database records.
- If a database is included in the formula, the formula is evaluated while the program reads database records.
- If a group field, page # field, subtotal, etc. is included in the formula, the formula is evaluated after database records are read and while the data from the records is being printed in the report.

WhileReadingRecords forces the formula to be evaluated while the program reads database records. When this function is used in a formula, the Formula Checker returns an error message if you attempt to include elements in the formula (groups, page number fields, etc.) that must be evaluated at a later time (while printing records).This function can also be used to force a formula that includes no database fields and no group fields to process while reading records instead of before reading records.

Examples

The following examples are applicable to Crystal syntax:

WhileReadingRecords;

ToNumber({orders detail.QUANTITY})

Forces the formula (which contains a database field) to be evaluated at its normal time (while reading records).

Note: If you try to include a group in this formula, you will get an error message.

WhileReadingRecords;

ToNumber ("12345")

Forces the formula (which contains no database fields or groups), to be evaluated later than it would normally be evaluated. In this case, it causes the formula to be evaluated while reading records instead of before reading records.

Note: If you try to include a group in this formula, you will get an error message.

WhilePrintingRecords

Basic and Crystal syntax.

Returns

None

Action

Forces the program to evaluate the formula while it is printing database record data.

Typical uses

Formulas are normally evaluated at the following times:

- If no database or group field is included in the formula, the formula is evaluated before the program reads database records.
- If a database is included in the formula, the formula is evaluated while the program reads database records.
- If a group field, page # field, subtotal, etc. is included in the formula, the formula is evaluated after database records are read and while the data from the records is being printed in the report.

WhilePrintingRecords forces the formula to be evaluated while the program prints database records.

This function can also force a formula that includes no database fields and no group fields to process while printing records instead of before reading records. It will also force a formula that includes database fields to process while printing records instead of while reading records.

Examples

The following examples are applicable to Crystal syntax:

WhilePrintingRecords;

3* Sum ({file.QTY1}, {file.QTY2})

Forces the formula (which contains a group) to be evaluated at its normal time (while printing records).

WhilePrintingRecords;

ToNumber ("12345")

Forces the formula (which contains no database fields or groups), to be evaluated later than it would normally be evaluated. In this case, it causes the formula to be evaluated while printing records instead of before reading records.

Note: Using this function in a formula forces the formula to be evaluated at print time. For more information on evaluation time considerations, see Evaluation Time functions.

EvaluateAfter (x)

Basic and Crystal syntax.

Arguments

x is any valid formula name.

Returns

None

Action

This function can be used to force one formula to be evaluated after another. For example, two separate formulas in your report may have the same evaluation time (BeforeReadingRecords, WhileReadingRecords, or WhilePrintingRecords), or they appear in the same section of the report and, therefore, are automatically evaluated while records are printing. In such cases, it is often unclear which formula is evaluated first.

In many situations, it may be unimportant which formula is evaluated first. However, some situations may require that one formula be evaluated after another. For instance, one formula may set the value of a variable while the other formula reads the value of that variable. The EvaluateAfter function can be used to force the second formula to be evaluated after the first formula has completed.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
EvaluateAfter({@SetVariable})
```

Forces the formula to wait until the @SetVariable formula has completed calculations.

Print State functions

These functions deal with the state of a report being previewed.

Previous (fld)

PreviousValue (fld)

Next (fld)

NextValue (fld)

IsNull (fld)

PreviousIsNull (fld)

NextIsNull (fld)

PageNumber

TotalPageCount

PageNofM

RecordNumber

GroupNumber

Record Selection

GroupSelection

InRepeatedGroupHeader

OnFirstRecord

OnLastRecord

Previous (fld)

Crystal syntax.

PreviousValue (fld) and Previous are equivalent functions. However, you can use Previous only with Crystal syntax, and PreviousValue only with Basic syntax.

Arguments

fld is any valid database or formula field in the report.

Returns

A field value of the same type that was passed as the fld argument.

Action

Previous (fld) returns the value of the specified field in the previous record.

Typical uses

You can use previous to identify the last record in a previous range or the last record occurring before a new range begins, or to test for duplicate values.

Examples

The following examples are applicable to Crystal syntax:

If Previous({file.QTY}) <> 0 Then

{file.QTY}/2

Else

{file.QTY}

Tests the previous value in the {file.QTY} field to see if it is a zero value. If it is not, it divides the value by two. If it is a zero value, it returns the value itself.

If Previous ({customer.CUSTOMER ID}) = {customer.CUSTOMER ID} Then

"Repeated Value"

Else

""

This flags repeated values in the {customer.CUSTOMER ID} field.

If Remainder(Previous ({file.SERIALNUM}), 300) = 0 Then

"Beginning, new block"

Else

""

Divides the previous value in the {file.SERIALNUM} field by 300. If there is no remainder, it flags the current value as "Beginning, new block". If there is a remainder (Else) it prints no flag. (This formula divides serial numbers into blocks of 300.)

Note: Using this function in a formula forces the formula to be evaluated at print time. For more information on evaluation time considerations, see Evaluation Time functions.

PreviousValue (fld)

Basic syntax

PreviousValue and Previous (fld) are equivalent functions. However, you can use PreviousValue only with Basic syntax and Previous only with Crystal syntax

Argument

fld is the name of a database field.

Returns

Value of specified field; type depends on type of that field.

Action

PreviousValue returns the value of the previous record of the specified field.

Typical use

You can use previous to identify the last record in a previous range or the last record occurring before a new range begins, or to test for duplicate values.

Examples

The following examples are applicable to Basic syntax:

If PreviousValue ({Orders Detail.Quantity}) <> 0 Then

formula = {Orders Detail.Quantity} / 2

Else

formula = {Orders Detail.Quantity}

End If

Tests the previous value in the {Orders Detail.Quantity} field to see if it is a zero value. If it is not, it divides the current value by two. If it is a zero value, it returns the current value itself.

If PreviousValue ({Orders.Order ID}) Mod 10 = 0 Then

formula = "Beginning of new block"

Else

formula = "Same block"

End If

Divides orders into blocks of 10. If the previous Order ID is a multiple of 10, it flags the current value as "Beginning of new block". Otherwise, it prints "Same block".

If PreviousValue ({Customer.Customer ID}) = {Customer.Customer ID} Then formula = "Customer Repeated"

Else

formula = "Customer Changed"

End If

This flags repeated values in the {Customer.Customer ID} field.

Note: Using this function in a formula forces the formula to be evaluated at print time. For more information on evaluation time considerations, see Evaluation Time functions.

Next (fld)

Crystal syntax.

Next and NextValue (fld) are equivalent functions. However, you can use Next only with Crystal syntax, and NextValue (fld) only with Basic syntax.

Arguments

fld is any valid database field or formula field.

Returns

A field value of the same type as the argument.

Action

Next(fld) returns the value of the specified field for the next record.

Typical uses

You can use Next to identify the first or last record in a range, to identify the starting point for a new group, or to test for duplicate values.

Examples

The following examples are applicable to Crystal syntax:

If Next({file.QTY})<>0 Then

{file.QTY}/2

Else

{file.QTY}

Tests the next value in the {file.QTY} field to see if it is a zero value. If it is not, it divides the value by two. If it is a zero value, it returns the value itself.

If Remainder(Next ({file.SERIALNUM}), 300) = 0 Then

"End of block"

Else

""

Divides the next value in the {file.SERIALNUM} field by 300. If there is no remainder, it flags the current value as "End of block". If there is a remainder (Else) it prints no flag. (This formula divides serial numbers into blocks of 300.)

Note: Using this function in a formula forces the formula to be evaluated at print time. For more information on evaluation time considerations, see Evaluation Time functions.

NextValue (fld)

Basic syntax.

NextValue and Next (fld) are equivalent functions. However, you can use NextValue only with Basic syntax and Next only with Crystal syntax.

Argument

fld is the name of a database field.

Returns

Value of specified field; type depends on type of that field.

Action

NextValue returns the value of the next record of the specified field.

Typical use

You can use NextValue to identify the first or last record in a range, to identify the starting point for a new group, or to test for duplicate values.

Examples

The following examples are applicable to Basic syntax:

If NextValue ({Orders Detail.Quantity}) <> 0 Then

formula = {Orders Detail.Quantity} / 2

Else

formula = {Orders Detail.Quantity}

End If

Tests the next value in the {Orders Detail.Quantity} field to see if it is zero. If it is not, it divides the current value by two. If it is zero, it returns the current value itself.

If NextValue ({Orders.Order ID}) Mod 10 = 0 Then

formula = "End of block"

Else

formula = "More to come"

End If

This formula divides orders into blocks of 10. If the next Order ID is a multiple of 10, it flags the current value as "End of block". Otherwise, it prints "More to come".

Note: Using this function in a formula forces the formula to be evaluated at print time. For more information on evaluation time considerations, see Evaluation Time functions.

IsNull (fld)

Basic and Crystal syntax.

Arguments

fld is any valid database, memo, or BLOB field.

Returns

Boolean Value

Action

Evaluates the field specified in the current record and returns TRUE if the field contains a null value.

Typical uses

You can use this function in a record selection formula to limit the report to records that have something other than a null value in the field specified. You can also use it to have the program take some action whenever it encounters a null value.

Examples

The following examples are applicable to both Basic and Crystal syntax:

IsNull({orders.ORDER AMOUNT}) = produces the following results:

If{orders.ORDER AMOUNT} shows this value	IsNull({orders.ORDER AMOUNT}) returns
200.00	False
(null)	True
100.00	False
347.12	False

```
Not IsNull({orders.ORDER AMOUNT})
```

When used as a record selection formula, includes in the report only those records that have something other than a null value in the {orders.ORDER AMOUNT} field.

Comments

Some databases support null data values and others do not. This function will not work if the active database(s) do not support null values. If the database(s) do support null values, the function can be used to reference the null values that get created as the result of a failed lookup while joining.

Note: Zero and blank (empty or " ") are not null values.

Related topics

Null fields and how to use IsNull (Basic syntax)

Null fields and how to use IsNull (Crystal syntax)

PreviousIsNull (fld)

Basic and Crystal syntax.

Arguments

fld is any numeric, formula, memo, or BLOB field in the report.

Returns

Boolean Value

Action

Evaluates the field specified in the previous record and returns TRUE if the field contains a null value.

Typical uses

You can use this function to test for the first item in a list and to take some action when that first item is identified.

Examples

The following example is applicable to Crystal syntax:

If PreviousIsNull ({employee.EMPLOYEE ID}) Then

"First employee of record"

Else

""

In an employee database with no null values in the Employee ID field, flags the first employee on the list.

Comments

Some databases support null data values and others do not. This function will not work if the active database(s) do not support null values. If the database(s) do support null values, the function can be used to reference the null values that get created as the result of a failed lookup while joining.

Note:

- Zero and blank (empty or " ") are not null values.
- Using this function in a formula forces the formula to be evaluated at print time. For more information on evaluation time considerations, see Evaluation Time functions.

NextIsNull (fld)

Basic and Crystal syntax.

Arguments
fld is any valid database, formula, memo or BLOB field in the report.

Returns
Boolean Value

Action
Evaluates the field specified in the next record and returns TRUE if the field contains a null value.

Typical uses
You can use this function to test for the last item in a list and to take some action when you identify that last item.

Examples
The following example is applicable to Crystal syntax:

If NextIsNull ({employee.EMPLOYEE ID}) Then

"Last employee of record"

Else

""

In an employee database with no null values in the Employee ID field, flags the last employee on the list.

Comments
Some databases support null data values and others do not. This function will not work if the active database(s) do not support null values. If the database(s) do support null values, the function can be used to reference the null values that get created as the result of a failed lookup while joining.

Note:

- Zero and blank (empty or " ") are not null values.
- Using this function in a formula forces the formula to be evaluated at print time. For more information on evaluation time considerations, see Evaluation Time functions.

PageNumber

Basic and Crystal syntax.

Returns
Whole Number

Action
PageNumber inserts the current page number as a field in a formula.

Typical uses

Use this function any time you want to determine the page number, at print time, and use that number in calculating formula results. For example, you can use this function when you want to print something on certain pages and not on others.

Examples

The following examples are applicable to Crystal syntax:

```
PageNumber
```

Creates a field containing the current page number that you can place anywhere on the page. Using this formula is the equivalent of using the Page Number field in the Special Fields folder in the Field Explorer.

If PageNumber > 1 Then

PageNumber

Else

0

Creates a field that prints the page number on every page but the first. By activating the Suppress if Zero check box on the Number tab of the Custom Style dialog box in the Format Editor, the formula will print nothing on the first page instead of a zero.

Note:

- You do not have to have a page number field in your report to use the PageNumber function in a report formula.
- Using this function in a formula forces the formula to be evaluated at print time.

  For more information on evaluation time considerations, see the Evaluation Time functions.

TotalPageCount

Basic and Crystal syntax.

Returns

Whole number

Action

Passes the report and returns the total number of pages.

Typical uses

You might use this function, for example, to place the number of pages into the report header or footer.

Examples

The following example is applicable to both Basic and Crystal syntax:

```
TotalPageCount
```

Returns 2 where your report contains a total of 2 pages.

Comments

This function will add an extra pass to the report.

PageNofM

Basic and Crystal syntax.

Returns

A text string

Action

PageNofM inserts the "Page [current page number] of [total page count]" as a field in a formula.

Typical uses

Use this function any time you want to determine the page number as compared to the total number of pages, at print time, and use that number in calculating formula results. For example, you can use this function when you want to print something on certain pages and not on others.

Examples

The following example is applicable to both Basic and Crystal syntax:

```
PageNofM
```

Creates a field containing the current page number and the total page count, which you can place anywhere on the page. Using this formula is the equivalent of using the PageNofM field from the Insert|Special Field menu.

RecordNumber

Basic and Crystal syntax.

Returns

Whole Number

Action

RecordNumber returns the current record number.

Typical uses

You can use this function in creating a record selection formula to print some records and exclude others.

Examples

The following example is applicable to both Basic and Crystal syntax:

```
RecordNumber
```

Creates a field containing the current record number that you can place on your report. Using this formula is the equivalent of using the Record Number field in the Special Fields folder in the Field Explorer.

Note: Using this function in a formula forces the formula to be evaluated at print time. For more information on evaluation time considerations, see the Evaluation Time functions.

GroupNumber

Basic and Crystal syntax.

Returns

Whole Number

Action

Returns the current group number.

Typical uses

You can use this function in a group selection formula to print some groups and exclude others.

Examples

The following example is applicable to both Basic and Crystal syntax:

```
GroupNumber
```

Creates a field containing the current group number that you can place on your report. Using this formula is the equivalent of using the GroupNumber field in the Special Fields folder in the Field Explorer.

Note: Using this function in a formula forces the formula to be evaluated at print time. For more information on evaluation time considerations, see the Evaluation Time functions.

RecordSelection

Basic and Crystal syntax.

Returns

The record selection formula

Action

PrintTime inserts the record selection formula for the report in a formula that you can place on your report.

Typical uses

You can use this function any time you want to print the record selection formula on the report, or whenever you want to set something as conditional on the record selection formula.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
RecordSelection
```

Creates a field that contains the record selection formula. You can then place this field on your report. Using this formula is the equivalent of using the Record Selection Formula field in the Special Fields folder in the Field Explorer.

GroupSelection

Basic and Crystal syntax.

Returns
The group selection formula

Action
PrintTime inserts the group selection formula for the report in a formula that you can place on your report.

Typical uses
You can use this function any time you want to print the group selection formula on the report, or whenever you want to set something as conditional on the group selection formula.

Examples
The following examples are applicable to both Basic and Crystal syntax:

```
GroupSelection
```

Creates a field that contains the group selection formula. You can then place this field on your report. Using this formula is the equivalent of using the Group Selection Formula field in the Special Fields folder in the Field Explorer.

InRepeatedGroupHeader

Basic and Crystal syntax.

Returns
Boolean

Action
Returns TRUE when a Group Header section is repeated on a second, third, etc., page.

Typical Uses
You can use this function to add conditional formatting to any group headers that appear when a group has broken onto a second, third, or fourth, etc., page. For example, you may want to add the text: "cont." to the group header when it appears on subsequent pages after the first.

OnFirstRecord

Basic and Crystal syntax.

Returns
Boolean

Action
Returns TRUE when the current record being evaluated is the first record in the report.

OnLastRecord

Basic and Crystal syntax.

Returns
Boolean

Action
Returns TRUE when the current record being evaluated is the last record in the report.

Document properties functions

These functions return values of attributes pertaining to a document.

PrintDate

PrintTime

ModificationDate

ModificationTime

DataDate

DataTime

ReportTitle

ReportComments

Filename

FileAuthor

FileCreationDate

PrintDate

Basic and Crystal syntax.

Returns
Date Value

Action
PrintDate inserts the date the report is printed as a field in a formula.

Typical uses
You can use this function any time you want to print the date of printing on the report, or to condition something in the report on the print date.

Examples
The following examples are applicable to Crystal syntax:

```
PrintDate
```

Creates a field containing the date the report is printed that you can place on your report. Using this formula is the equivalent of using the PrintDate field in the Special Fields folder in the Field Explorer.

If PrintDate >= CDate(1992,01,01) Then

"Please excuse the delayed report."

Else

""

Prints the message apologizing for the delay if the report prints after January 1, 1992, and prints nothing if the report prints before that date.

Comments

The PrintDate function retrieves a value from the setting established in the report via the Set Print Date/Time command on the Report menu. By default, this value matches the settings in the Windows Date/Time control panel unless changed by the user.

PrintTime

Basic and Crystal syntax.

Returns

Time Value

Action

PrintTime inserts the time the report is printed as a field in a formula.

Typical uses

You can use this function any time you want to print the time of printing on the report, or to condition something in the report on the print time.

Examples

The following example is applicable to both Basic and Crystal syntax:

```
PrintTime
```

Creates a field containing the time the report is printed that you can place on your report. Using this formula is the equivalent of using the Print Time field in the Special Fields folder in the Field Explorer.

ModificationDate

Basic and Crystal syntax.

Returns

Date Value

Action

ModificationDate inserts the date the report was last modified as a field in a formula.

Typical uses

Use this function any time you want to print the date the report was last modified, or to condition something in the report on the modification date.

Examples

The following example is applicable to both Basic and Crystal syntax:

```
ModificationDate
```

Creates a field that you can place on your report which will return the date the report was last modified. Using this formula is the equivalent of using the ModificationDate field in the Special Fields folder in the Field Explorer.

ModificationTime

Basic and Crystal syntax.

Returns

Time Value

Action

ModificationTime inserts the time the report was last modified as a field in a formula.

Typical uses

Use this function any time you want to time the date the report was last modified, or to condition something in the report on the modification time.

Examples

The following example is applicable to both Basic and Crystal syntax:

```
ModificationTime
```

Creates a field containing the time of the report was last modified, which you can place on your report. Using this formula is the equivalent of using the ModificationTime field in the Special Fields folder in the Field Explorer.

DataDate

Basic and Crystal syntax.

Returns

Date Value

Action

DataDate returns the date that the report was last refreshed.

Typical uses

Use DataDate any time you want to date stamp your data.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
DataDate
```

Returns 1998/05/06 if the report was last refreshed on 1998/05/06.

```
DataDate
```

Returns 1973/12/06 if the report was last refreshed on 1973/12/06.

Comments
The date format is determined by the settings in the international or regional settings control panel.

DataTime

Basic and Crystal syntax.

Returns
Time Value

Action
DataTime returns the time that the report was last refreshed.

Typical uses
Use DataTime any time you want to time stamp your data.

Examples
The following examples are applicable to both Basic and Crystal syntax:

```
DataTime
```

Returns 12:45:23 if the report was last refreshed at 12:45:23.

```
DataTime
```

Returns 17:26:10 if the report was last refreshed at 17:26:10.

ReportTitle

Basic and Crystal syntax.

Returns
String

Action
ReportTitle inserts the title of the report as a field in a formula.

Typical uses
You can use this function any time you want to print the title of the report.

Examples
The following example is applicable to both Basic and Crystal syntax:

```
ReportTitle
```

Creates a field containing the title that you can place on your report. Using this formula is the equivalent of using the ReportTitle field in the Special Fields folder in the Field Explorer.

ReportComments

Basic and Crystal syntax.

Returns

String

Action

ReportComments inserts the comments included with the report as a field in a formula.

Typical uses

You can use this function any time you want to print the comments for report entered in the Document Properties dialog box.

Examples

The following example is applicable to both Basic and Crystal syntax:

```
ReportComments
```

Creates a field containing the comments that you can place on your report. Using this formula is the equivalent of using the ReportComments field in the Special Fields folder in the Field Explorer.

Filename

Basic and Crystal syntax.

Returns

String

Action

Filename inserts (in a formula) the file name and file path that are included with the report as a field.

Typical uses

You can use this function any time you want to print the file name for the report.

Examples

The following example is applicable to both Basic and Crystal syntax:

```
Filename
```

Creates a field containing the filename. Using this formula is the equivalent of using the File Path and Name field in the Special Fields folder in the Field Explorer.

FileAuthor

Basic and Crystal syntax.

Returns

String

Action

FileAuthor inserts (in a formula) the file author name that is included with the report as a field.

Typical uses

You can use this function any time you want to print the name of the file author (as specified in the Document Properties dialog box).

Examples

The following example is applicable to both Basic and Crystal syntax:

```
FileAuthor
```

Creates a field containing the name of the file author. Using this formula is the equivalent of using the File Author field in the Special Fields folder in the Field Explorer.

FileCreationDate

Basic and Crystal syntax.

Returns

String

Action

FileCreationDate inserts (in a formula) the report creation date that is included with the report as a field.

Typical uses

You can use this function any time you want to print the date when the report was created (as shown in the Document Properties dialog box).

Examples

The following example is applicable to both Basic and Crystal syntax:

```
FileCreationDate
```

Creates a field containing the report creation date. Using this formula is the equivalent of using the File Creation Date field in the in the Special Fields folder in the Field Explorer.

Alerts functions

The functions related to Report Alerts allow you to create print time formulas that refer to the results of report alerts. Any formula that references an alert becomes a print time formula.

IsAlertEnabled(alertName)

IsAlertTriggered(alertName)

AlertMessage(alertName)

For general information on alerts, see About Report Alerts.

IsAlertEnabled(alertName)

Basic and Crystal syntax.

Arguments

alertName is the name of an alert (in quotation marks) that you've already created.

Locate the available alerts in the Alert Names list, which appears in the Formula Editor's Functions tree once you've created an alert.

Returns

Boolean value indicating whether or not the alert is enabled.

Tip: To enable or disable an existing alert, click Create Alerts on the Report Menu. Select the desired alert and click Edit. In the Edit Alert dialog box, select or clear the Enabled option.

Examples

Crystal Syntax:

```
If IsAlertEnabled("myAlert") Then
 AlertMessage("myAlert")
 Else
 "myAlert is disabled"
```

Basic Syntax:

```
If IsAlertEnabled("myAlert") Then
 Formula = AlertMessage("myAlert")
 Else
 Formula = "myAlert is disabled"
 End If
```

Use this formula to display a message that lets report users know whether or not an alert is enabled. For instance, open the sample report called Alerting.rpt. Create the formula, replacing "myAlert" with "Sales" (which is the name of the alert in the sample report). Then drag the formula field into the Report Header section. By default, the Sales alert is enabled, so its alert message will appear in the Report Header. If you disable the alert (in the Edit Alert dialog box), the message displayed in the report will change to the "disabled" message that you specified in the formula.

Comment

Any formula that references an alert becomes a print time formula.

IsAlertTriggered(alertName)

Basic and Crystal syntax.

Arguments

alertName is the name of an alert (in quotation marks) that you've already created.

Locate the available alerts in the Alert Names list, which appears in the Formula Editor's Functions tree once you've created an alert.

Returns

Boolean value indicating whether or not the alert is "triggered"—that is, whether or not the report data meets the alert condition formula.

Examples

Crystal syntax:

```
If IsAlertTriggered("myAlert") Then
 AlertMessage("myAlert")
 Else
 "not triggered"
```

Basic syntax:

```
If IsAlertTriggered("myAlert") Then
 Formula = AlertMessage("myAlert")
 Else
 Formula = "not triggered"
 End If
```

Use this formula to display messages that let report users know which records have triggered the report alert. For instance, open the sample report called Alerting.rpt. Create the formula, replacing "myAlert" with "Sales" (which is the name of the alert in the sample report). Then drag the formula field into the Details section. (You might have to resize the Details section in order to see the formula field properly.) The formula displays the alert message for records that meet the alert conditions; otherwise, the formula displays your "not triggered" message.

Comment

Any formula that references an alert becomes a print time formula.

AlertMessage(alertName)

Basic and Crystal syntax.

Arguments

alertName is the name of an alert (in quotation marks) that you've already created.

Locate the available alerts in the Alert Names list, which appears in the Formula Editor's Functions tree once you've created an alert.

Returns

- Returns the alert message string for records that "trigger" the alert (that is, for records that meet the alert condition formula.)
- Returns an empty string ("") for records that do not meet the alert condition formula.

Examples

Crystal syntax:

```
If IsAlertEnabled("myAlert") Then
 AlertMessage("myAlert")
 Else
 "myAlert is disabled"
```

Basic syntax:

```
If IsAlertEnabled("myAlert") Then
 Formula = AlertMessage("myAlert")
 Else
 Formula = "myAlert is disabled"
 End If
```

In these examples, `AlertMessage` returns the message that you defined when you created the alert.

Comment

Any formula that references an alert becomes a print time formula.

Additional Functions

FRInterestCoverage (CashFlow, InterestExpenses)

FRCashFlowVsTotalDebt (CashFlow, TotalDebt)

FRReturnOnEquity (NetProfit, TotalEquity)

FRReturnOnNetFixedAssets (NetProfit, NetFixedAssets)

FRReturnOnTotalAssets (NetProfit, TotalAssets)

FRReturnOnInvestedCapital (NetProfit, TotalBankDebt, TotalEquity)

FRReturnOnCommonEquity (NetProfit, PreferredDividend, CommonEquity)

FREarningsPerCommonShare (NetProfit, PreferredDividend, NumOfCommonShare)

FRAccRecTurnover (AccountReceivable, Sales, NumOfDays)

FRInventoryTurnover (Inventory, Sales, NumOfDays)

FRPriceEarningsRatio (MarketPrice, EarningsPerShare)

FRDividendYield (Dividend, MarketPrice)

Now

Picture (string, picture)

LooksLike (string, mask)

Soundex (string)

DateTimeTo2000 (dateTime, number)

DTSToDateTime (DateTimeString)

DTSToTimeField (DateTimeString)

DateTimeToDate (dateTime)

DateTimeToTime (dateTime)

DateTimeToSeconds (dateTime)

DateTo2000 (Date, Number)

Basic and Crystal syntax.

Arguments

- Date: accepts only valid date fields, with either 2 digit or 4 digit years.
- a database field defined as a date field. File|Report Options should be set to Convert Date-time fields to Date.
- the CDate function (CDate(yyyy, MM, dd)). A four digit year is required by this function. If two digits are entered it will assume 00xx (for example, 98 AD).
- Number: a number between 0 and 99 corresponding with the desired windowing year.

Returns

a date field with a four digit year

Action

- 2 digit years (xx):

  If the Year value is greater than the windowing number, 19 is appended before the 2 digits (19xx). If the Year value is less than or equal to the windowing number, 20 is appended (20xx).

- 4 digit years (19xx):

  If the last two digits in the Year value are greater than the windowing number, the Year is preserved as found in the date field (19xx). If the last two digits in the Year value are less than or equal to the windowing number, the first two digits are changed to 20 (20xx). If the first two digits in the year field are 20, the Year is preserved as found in the date field (20xx).

Note:

- If the year is less than or equal to 1899 and greater or equal to 100, then there will be no change to the date.
- Whether your database dates are interpreted as two digit years or 4 digit years (19xx) is dependent on the database driver you are using. The result of the formula should be the same in either case.

Examples

The following examples are applicable to Basic and Crystal syntax:

Here the window value is greater than the year and so it will change it to 20XX.

```
DateTo2000(CDate(1993,12,12), 99)
```

should return 2093/12/12

Here the window value is less than the year and so it will not change the year.

```
DateTo2000(CDate(1993,12,12), 92)
```

should return 1993/12/12

Other Examples

This is an example of a date that will not be affected by the function.

```
DateTo2000(CDate(1899,12,12), 99)
```

should return 1899/12/12

This is an example of a date that will not be affected by the function.

```
DateTo2000(CDate(100,12,12), 99)
```

should return 100/12/12

Some databases maintain the year of a date as a two digit field and so this will simulate that date field.

```
DateTo2000(CDate(98,12,12), 99)
```

should return 2098/12/12

```
DateTo2000(CDate(98,12,12), 97)
```

should return 1998/12/12

```
DateTo2000(CDate(9,12,12), 10)
```

should return 2009/12/12

```
DateTo2000(CDate(1,12,12), 0)
```

should return 1901/12/12

DTSTo2000 (DateString, Number)

Basic and Crystal syntax.

Arguments

- Date String: accepts only valid date fields, with either 2 digit or 4 digit years.
- a database field defined as a date field. File|Report Options should be set to Convert Date-time fields to Date-time String.
- a database field defined as a string field, but containing a date.
- a date-time string entered in the format: "yyyy/MM/dd HH:mm:ss.00" or "yy/MM/dd HH:mm:ss:00", example: "1997/04/11 12:12:12.00" or "97/04/11 12:12:12.00".
- Number: a number between 0 and 99 corresponding with the desired windowing year.

Returns

a DateTime string with a four digit year

Action

- 2 digit years (xx):

  If the Year value is greater than the windowing number, 19 is appended before the 2 digits (19xx). If the Year value is less than or equal to the windowing number, 20 is appended (20xx).

- 4 digit years (19xx):

  If the last two digits in the Year value are greater than the windowing number, the Year is preserved as found in the date field (19xx). If the two digits in the Year value are less than or equal to the windowing number, the first two digits are changed to 20 (20xx). If the first two digits in the year field are 20, the Year is preserved as found in the date field (20xx).

  Note: If the year is less than or equal to 1899 and greater than or equal to 100, there will be no change to the date.

Examples

Note: Your computer date setting must be "yy/mm/dd" or "yyyy/mm/dd" in order for the following examples to return the correct results.

The following examples are applicable to Basic and Crystal syntax:

Here the window value is greater than the year and so it will change it to 20XX.

```
DTSTo2000(ToText(CDateTime(1988,12,12,12,12,12)), 90)
```

should return 2088/12/12 12:12:12PM

Here the window value is less than the year and so it will not change the year.

```
DTSTo2000(ToText(CDateTime(1984,12,12,1,2,3)), 83)
```

should return 1984/12/12 1:02:03AM

Other Examples

These are examples of a date that will not be affected by the function.

```
DTSTo2000(ToText(CDateTime(1899,12,12,1,2,3)), 99)
```

should return 1899/12/12 1:02:03AM

```
DTSTo2000(ToText(CDateTime(1899,12,12)), 99)
```

should return 1899/12/12 12:00:00PM

```
DTSTo2000(ToText(CDateTime(100,12,12,1,2,3)), 99)
```

should return 100/12/12 1:02:03AM

Some databases maintain the year of a date as a two digit field and so these samples will simulate that date field.

```
DTSTo2000(ToText(CDateTime(98,12,12,1,2,3)), 99)
```

should return 2098/12/12 1:02:03AM

```
DTSTo2000(ToText(CDateTime(98,12,12,1,2,3)), 97)
```

should return 1998/12/12 1:02:03AM

```
DTSTo2000(ToText(CDateTime(98,12,12)), 97)
```

should return 1998/12/12 12:00:00PM

```
DTSTo2000(ToText(CDateTime(9,12,12,1,2,3)), 10)
```

should return 2009/12/12 1:02:03AM

```
DTSTo2000(ToText(CDateTime(1,12,12,1,2,3)), 0)
```

should return 1901/12/12 1:02:03AM

DTSToDate (DateTimeString)

Basic and Crystal syntax.

Arguments

DateTimeString is a string including a date and a Time value.

Returns

Date Value

Action

Evaluates the string specified and returns only the date.

Examples

```
DTSToDate ("2000/01/13 11:30:15")
```

Returns the Date value, January 13, 2000. The month must be entered as 01 rather than just 1.

Comments

The Date value returned by DTSToDate is the date at midnight. The Time value is dropped, and midnight is established as the implied time of the date. This is only relevant when performing operations with the resulting date, such as adding a Time value.

Note: For new formulas, use the CDate or DateValue functions instead of DTSToDate. The CDate and DateValue functions are able to convert more types of strings.

DTSToSeconds (DateTimeString)

Basic and Crystal syntax.

Arguments

DateTimeString is a string including a Date and a Time value.

Returns

Whole Number

Action

Evaluates the string specified and converts the Time value to the number of seconds from 00:00:00 (12:00 midnight) to the specified time.

Examples

The following example is applicable to Crystal syntax:

```
DTSToSeconds("2000/01/13 11:30:15")
```

Returns 41415.

Comments

Instead of using the DTSToSeconds function, you can use:

```
DateDiff ("s", #12:00 am#, CDateTime (CDate(0,0,0), CTime
(DateTimeString)))
```

The above expression is able to convert more types of strings.

DTSToTimeString (DateTimeString)

Basic and Crystal syntax.

Arguments

DateTimeString is a string including a Date and a Time value.

Returns
Time String Value

Action
Evaluates the string specified and returns only the Time value in military format (00:00:00).

Examples
The following example is applicable to Crystal syntax:

```
DTSToTime(DateTime(1998,10,25,8,35,30))
```

Returns 08:35:30.

ExchGetId (address)

Basic and Crystal syntax.

Argument
Address is the Address of sender/recipient (string data type).

Return
String

Action
The ExchGetID (address) function will begin by determining whether the address in X500 or X400 format. Once this has been solved, the function will then determine the ID.

- If the field is in X500 format, the function will extract the last instance of the "CN=" code (not case sensitive).
- If the field is in the X400 format, it will extract the SMTP or MS Ids.

Note: The fields must follow the address type standards for the functions to operate them.

Examples
The following examples are applicable to Crystal syntax:

Field = /o=Seagate Software/ou=CRYSTALNT

/cn=Configuration/cn=Servers/cn=ESPRESSO/cn=Fredt

ExchGetID({Field})

Returns "Fredt".

Field = c=US; a= ;p=Microsoft; o=appsga;

dda:smtp=James@crystal.com

ExchGetID({Field})

Returns "James@crystal.com".

Field = c=US; a= ;p=Microsoft; o=appsga; dda:ms=com/

crystal/James

ExchGetID({Field})

Returns "com/crystal/James".

ExchGetOrganization (address)

Basic and Crystal syntax.

Arguments

Address is the Address of the sender/recipient (string data type).

Returns

String

Action

The function will begin by determining whether the address is in X500 or X400 format. Once this has been solved, the function will then determine the Organization Name.

- If the field is in X500 format, the function will extract the last instance of the "/O=" code (not case sensitive).
- If the field is in the X400 format, it will extract the instance of the "P=" code (not case sensitive).

Note: The fields must follow the address type standards for the functions to operate on them.

Examples

The following examples are applicable to Crystal syntax:

Field = /o=Seagate Software/ou=CRYSTALNT/

cn=Configuration/

cn=Servers/cn=ESPRESSO/cn=Fredt

ExchGetOrganization({Field})

Returns "Seagate Software".

Field = c=US; a= ; p=Microsoft; o=appsga; da:ms=com/

Rich/James.

ExchGetOrganization({Field})

Returns "Microsoft".

Field = c=US; a= ; p=Microsoft; o=appsga;

da:smtp=James@crystal.com.

ExchGetOrganization({Field})

Returns "Microsoft".

ExchGetSite (address)

Basic and Crystal syntax.

Argument(s)

Address is the Address of sender/recipient (string data type).

Return(s)

String

Action

The function will begin by determining whether the address is in X500 or X400 format. Once this has been solved, the function will then determine the Site Name.

- If the field is in X500 format, the function will extract the last instance of the "/OU=" code (not case sensitive).
- If the field is in the X400 format, it will extract the instance of the "O=" code (not case sensitive).

Note: The fields must follow the address type standards for the functions to operate on them.

Examples

The following examples are applicable to Crystal syntax:

Field = /o=Seagate Software /ou=CRYSTALNT/

cn=Configuration/cn=Servers/cn=ESPRESSO/cn=Fredt

ExchGetSite({Field})

Returns "CRYSTALNT".

Field = c=US; a= ;p=Microsoft; o=appsga;

dda:smtp=James@Rich.com.

ExchGetSite({Field})

Returns "appsga".

ExtractString (origin, startString, endString)

Basic and Crystal syntax.

Arguments

- Origin: String
- StartString: String
- EndString: String

Returns

String

Action

This function will return the first occurrence (in the Origin string) of a string that starts with the StartString and ends with EndString. If the EndString is not found, the string starting with StartString until the end of the string is returned.

Examples

The following examples are applicable to Crystal syntax:

StringVar strTest := "James and crystal";

ExtractString (strTest,"J","s");

Returns "ame".

StringVar strTest := "James and crystal";

ExtractString (strTest, "", "s");

Returns "Jame".

EventNumber (eventNum)

Basic and Crystal syntax.

Arguments

eventNum is the numeric code that describes the event that was logged (number data type).

Returns

Number

Action

The function will return the appropriate text description for the eventNum argument.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
Message Tracking Log Event Number
```

Returns 4.

```
EventNumber({Message Tracking Log Event Number})
```

Returns "Message Submission".

```
Message Tracking Log Event Number
```

Returns 1000.

```
EventNumber ({Message Tracking Log Event Number})
```

Returns "Local Delivery".

ExchGetPath (Path)

Basic and Crystal syntax.

Arguments

Path is the Address of sender/receiver (string data type).

Returns

String

Action

This function will return the container information in an address field.

- If the address type is X500, the function will return all of the information from the first instance of the "CN=" code until the last instance of the "CN=" code. If there is only one instance of the "CN=" code, the function will return NULL.
- If the address type is X400, the function will return all "OU*=" codes (residing between the "P=" and "O=" codes).
- If the address starts with "DDA:", the function will return all information after the "DDA:" code.
- If the field is blank, "UNKNOWN ADDRESS" will be returned.

Examples

The following examples are applicable to Crystal syntax:

Field = /o=Seagate Software/ou=CRYSTALNT/

cn=Configuration/cn=Servers/cn=ESPRESSO/cn=Fredt

ExchGetPath ({Field})

Returns "cn=Configuration/cn=Servers/cn=ESPRESSO".

Field - c=US; a= ; p=Microsoft; ou1=James;

ou2=RichCorp; o=appsga;

dda:smtp=James@Rich.com

ExchGetPath({Field})

Returns "ou1=James; ou2=Richcorp".

Field = DDA:MS=CRYSTAL/JAMES/crystal

ExchGetpath9{Field})

Returns "MS=CRYSTAL/JAMES/crystal".

ByteToText (byte)

Basic and Crystal syntax.

Arguments

byte is the size of a message in bytes.

Returns

Number

Action

This function will return the size of a message in the "appropriate" unit.

- If the argument is less than 1024, the result will be in bytes.
- If the argument is between 1024 and 1048576, then the result will be in kilobytes.
- Otherwise, the result will be in megabytes.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
ByteToText(1000)
```

Returns "1000 bytes".

```
ByteToText(345555)
```

Returns "337KB".

FRCurrentRatio (CurrentAssets, CurrentLiabilities)

Basic and Crystal syntax.

Format

```
FRCurrentRatio(CurrentAssets, CurrentLiabilities)
```

Arguments

- CurrentAssets is a Number or formula.

  In general, CurrentAssets consist of cash, accounts receivable, inventories, prepaid expenses, and short term marketable investment in a company's balance sheet.

- CurrentLiabilities is a Number or formula.

  In general, CurrentLiabilities consist of bank advances, accounts payable, dividends/income taxes payable, and a portion of long term loans due within one year in a company's balance sheet.

Action

FRCurrentRatio returns the ratio between CurrentAssets and CurrentLiabilities (for example, CurrentRatio = CurrentAssets / CurrentLiabilities).

Typical uses

This ratio is used to measure the short term liquidity of a company. It is a measure of how much the value of current assets exceeds the value of current liabilities. The higher the ratio, the higher the liquidity.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
FRCurrentRatio(5, 2.5)
```

Returns 2.

```
FRCurrentRatio(4.2, 2)
```

Returns 2.1.

```
FRCurrentRatio(1.1, 2.2)
```

Returns 0.5.

Note: None of the arguments can be negative; you may have to check for negative values before passing the arguments.

Comment

FRCurrentRatio is one of the many financial analysis tools used in interpreting the financial position of a company. As in the case of all ratios, it has to be used with caution. It can just be used as a clue and not solid proof for forming a judgment on the financial status of a company.

Neither one of the ratios can be used alone for doing financial analysis and there are no fixed rules on the results of the ratios. Apart from differences between type of industries, result varies among different companies within the same industry and within different account periods. The results should only be used for relative comparison and trend analysis, rather than treating them as something absolute.

FRQuickRatio (CurrentAssets, Inventories, CurrentLiabilities)

Basic and Crystal syntax.

Format

```
FRQuickRatio(CurrentAssets, Inventories, CurrentLiabilities)
```

Arguments

- CurrentAssets is a Number or formula. In general, CurrentAssets consist of cash, accounts receivable, inventories, prepaid expenses, and short term marketable investments in a company's balance sheet.
- Inventories is a Number or formula. In general, Inventories consist of raw materials, work in progress, and finished goods in a company's balance sheet.
- CurrentLiabilities is a Number or formula. In general, CurrentLiabilities consists of bank advances, accounts payable, dividends/income taxes payable, and a portion of long term loans due within one year in a company's balance sheet.

Action

FRQuickRatio returns the ratio between CurrentAssets (less Inventories) and CurrentLiabilities (for example, QuickRatio = (CurrentAssets - Inventories) / CurrentLiabilities).

Typical uses

Since it takes time to convert raw materials to finished goods, and even finished goods are not really a liquid asset, this ratio removes inventories from current assets in calculating the liquidity of the company. It is a measure of how well current liabilities are covered by cash and ready cash equivalent. The higher the ratio, the higher the liquidity.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
FRQuickRatio(7.2, 2.2, 2)
```

Returns 2.5.

```
FRQuickRatio(3, 1.5, 3)
```

Returns 0.5.

Note:

- None of the arguments can be negative; you may have to check for negative values before passing the arguments.
- The Formula Editor will automatically check to see if Inventories is smaller than or equal to CurrentAssets. If Inventories is smaller than CurrentAssets, you will receive an error message.

Comment

FRQuickRatio is one of the many financial analysis tools used in interpreting the financial position of a company. As in the case of all ratios, it has to be used with caution. It can just be used as a clue and not solid proof for forming a judgment on the financial status of a company.

Neither one of the ratios can be used alone for doing financial analysis and there are no fixed rules on the results of the ratios. Apart from differences between type of industries, result varies among different companies within the same industry and within different account periods. The results should only be used for relative comparison and trend analysis, rather than treating them as something absolute.

FRDebtEquityRatio (TotalLiabilities, TotalEquity)

Basic and Crystal syntax.

Format

```
FRDebtEquityRatio(TotalLiabilities, TotalEquity)
```

Arguments

- TotalLiabilities is a Number or formula. In general, TotalLiabilities consist of all the current and long term liabilities in a company's balance sheet. However, some analysts would only include short and long term banks debts in calculating this ratio.
- TotalEquity is a Number or formula. In general, TotalEquity consist of common stocks, preferred stocks, capital surplus, and retained earnings in a company's balance sheet.

Action

FRDebtEquityRatio returns the ratio between TotalLiabilities and TotalEquity (for example, DebtEquityRatio = TotalLiabilities / TotalEquity).

Typical uses

This ratio is used to measure the capital structure of the company. It can be used to find out the relationship of total liabilities/debt to a company's equity and can be a warning that the company's borrowing is excessive. The higher the ratio, the higher the financial risk.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
FRDebtEquityRatio(2.6, 1.3)
```

Returns 2.

```
FRDebtEquityRatio(1.5, 2.5)
```

Returns 0.6.

Note: None of the arguments can be negative; you may have to check for negative values before passing the arguments.

Comment

FRDebtEquityRatio is one of the many financial analysis tools used in interpreting the financial position of a company. As in the case of all ratios, it has to be used with caution. It can just be used as a clue and not solid proof for forming a judgment on the financial status of a company.

Neither one of the ratios can be used alone for doing financial analysis and there are no fixed rules on the results of the ratios. Apart from differences between type of industries, result varies among different companies within the same industry and within different account periods. The results should only be used for relative comparison and trend analysis, rather than treating them as something absolute.

FREquityVsTotalAssets (TotalEquity, TotalAssets)

Basic and Crystal syntax.

Format

```
FREquityVsTotalAssets(TotalEquity, TotalAssets)
```

Arguments

- TotalEquity is a Number or formula. In general, TotalEquity consist of common stocks, preferred stocks, capital surplus, and retained earnings in a company's balance sheet.
- TotalAssets is a Number or formula. In general, TotalAssets consist of total current assets, net fixed assets and all other assets in a company's balance sheet.

Action

FREquityVsTotalAssets returns the ratio between TotalEquity and TotalAssets (for example, EquityVsTotalAssets = TotalEquity / TotalAssets).

Typical uses

This ratio is used to measure the financial structure of a company. The higher the ratio, the stronger a company's capital structure and the better it is capitalized.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
FREquityVsTotalAssets(2, 4)
```

Returns 0.5.

```
FREquityVsTotalAssets(2.82, 18.8)
```

Returns 0.15.

Note:

- None of the arguments can be negative; you may have to check for negative values before passing the arguments.
- TotalEquity must be smaller or equal to TotalAssets, otherwise you will receive an error message.

Comment

FREquityVsTotalAssets is one of the many financial analysis tools used in interpreting the financial position of a company. As in the case of all ratios, it has to be used with caution. It can just be used as a clue and not solid proof for forming a judgment on the financial status of a company.

Neither one of the ratios can be used alone for doing financial analysis and there are no fixed rules on the results of the ratios. Apart from differences between type of industries, result varies among different companies within the same industry and within different account periods. The results should only be used for relative comparison and trend analysis, rather than treating them as something absolute.

FRNetProfitMargin (NetProfit, Sales)

Basic and Crystal syntax.

Format

```
FRNetProfitMargin(NetProfit, Sales)
```

Arguments

- NetProfit is a Number or formula. NetProfit is the amount of earnings a company earns during the year from the sale of its products or services minus all the expense items. It is the bottom line figure in a company's income statement. Some analysts may want to use the net profit before extraordinary items and/or income taxes for calculating this ratio.
- Sales is a Number or formula. Sales is the total sales during the period that appears in a company's income statement.

Action

FRNetProfitMargin returns the ratio between NetProfit and Sales (for example, NetProfitMargin = NetProfit / Sales).

Typical uses

This ratio is used to measure how well the company is doing. The higher the result, the more profitable the company.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
FRNetProfitMargin(3, 50)
```

Returns 0.06.

```
FRNetProfitMargin(5.6, 100)
```

Returns 0.056.

Note: None of the arguments can be negative; you may have to check for negative values before passing the arguments.

Comment

FRNetProfitMargin is one of the many financial analysis tools used in interpreting the financial position of a company. As in the case of all ratios, it has to be used with caution. It can just be used as a clue and not solid proof for forming a judgment on the financial status of a company.

Neither one of the ratios can be used alone for doing financial analysis and there are no fixed rules on the results of the ratios. Apart from differences between type of industries, result varies among different companies within the same industry and within different account periods. The results should only be used for relative comparison and trend analysis, rather than treating them as something absolute.

FRGrossProfitMargin (GrossProfit, Sales)

Basic and Crystal syntax.

Format

```
FRGrossProfitMargin(GrossProfit, Sales)
```

Arguments

- GrossProfit is a Number or formula. In general, GrossProfit equals sales minus cost of goods sold in a company's income statement.
- Sales is a Number or formula. Sales is the total sales during the period that appears in a company's income statement.

Action

FRGrossProfitMargin returns the ratio between GrossProfit and Sales (for example, GrossProfitMargin = GrossProfitMargin / Sales).

Typical uses

This is a measure of how much return is received from the sales of goods prior to deducting all other expenses. The result varies from industry to industry depending on accounting treatment of cost of production that is included in cost of goods sold.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
FRGrossProfitMargin(3.4, 50)
```

Returns 0.068.

```
FRGrossProfitMargin(36.15, 120.5)
```

Returns 0.3.

Note: None of the arguments can be negative; you may have to check for negative values before passing the arguments.

Comment

FRGrossProfitMargin is one of the many financial analysis tools used in interpreting the financial position of a company. As in the case of all ratios, it has to be used with caution. It can just be used as a clue and not solid proof for forming a judgment on the financial status of a company.

Neither one of the ratios can be used alone for doing financial analysis and there are no fixed rules on the results of the ratios. Apart from differences between type of industries, result varies among different companies within the same industry and within different account periods. The results should only be used for relative comparison and trend analysis, rather than treating them as something absolute.

FROperatingProfitMargin (OperatingProfit, Sales)

Basic and Crystal syntax.

Format

```
FROperatingProfitMargin(OperatingProfit, Sales)
```

Arguments

- OperatingProfit is a Number or formula. In general, OperatingProfit equals gross profit less all selling, administrative and general expenses in a company's income statement.
- Sales is a Number or formula. Sales is the total sales during the period that appears in a company's income statement.

Action

FROperatingProfitMargin returns the ratio between OperatingProfit and Sales (for example, OperatingProfitMargin = OperatingProfit / Sales).

Typical uses

FROperatingProfitMargin is used to measure the rate of profit on sales after allowing for all the directly related expenses. It is an indication of how profitable a company's operation is. The higher the result, the higher its profitability.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
FROperatingProfitMargin(3.3, 30)
```

Returns 0.11.

```
FROperatingProfitMargin(11, 16.5)
```

Returns 0.6875.

Note: None of the arguments can be negative; you may have to check for negative values before passing the arguments.

Comment

FROperatingProfitMargin is one of the many financial analysis tools used in interpreting the financial position of a company. As in the case of all ratios, it has to be used with caution. It can just be used as a clue and not solid proof for forming a judgment on the financial status of a company.

Neither one of the ratios can be used alone for doing financial analysis and there are no fixed rules on the results of the ratios. Apart from differences between type of industries, result varies among different companies within the same industry and within different account periods. The results should only be used for relative comparison and trend analysis, rather than treating them as something absolute.

FRInterestCoverage (CashFlow, InterestExpenses)

Basic and Crystal syntax.

Format

```
FRInterestCoverage(CashFlow, InterestExpenses)
```

Arguments

- CashFlow is a Number or formula. In general, CashFlow equals the net profit of the company plus all the expense items not involving cash (such as depreciation, capital losses, etc.) minus all the profit not involving cash (such as capital gains, etc.). This information can be found in a company's income statement and statement of change in financial position.
- InterestExpenses is a Number or formula. InterestExpenses is the total interest expenses that can be found in a company's income statement.

Action

FRInterestCoverage returns the ratios between CashFlow and InterestExpenses (for example, InterestCoverage = CashFlow / InterestExpenses).

Typical uses

This ratio is used to measure the ability of a company to pay the interest charges on its debt with the cash flows generated from its operation. It is usually measured in number of times (for example, 3X, etc.). The higher the result, the higher the ability for the company to meet interest payment even in time of interest rate hike.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
FRInterestCoverage(120, 50)
```

Returns 24.

```
FRInterestCoverage(30, 100)
```

Returns 0.3.

Note: None of the arguments can be negative; you may have to check for negative values before passing the arguments.

Comment

FRInterestCoverage is one of the many financial analysis tools used in interpreting the financial position of a company. As in the case of all ratios, it has to be used with caution. It can just be used as a clue and not solid proof for forming a judgment on the financial status of a company.

Neither one of the ratios can be used alone for doing financial analysis and there are no fixed rules on the results of the ratios. Apart from differences between type of industries, result varies among different companies within the same industry and within different account periods. The results should only be used for relative comparison and trend analysis, rather than treating them as something absolute.

FRCashFlowVsTotalDebt (CashFlow, TotalDebt)

Basic and Crystal syntax.

Format

```
FRCashFlowVsTotalDebt(CashFlow, TotalDebt)
```

Arguments

- CashFlow is a Number or formula. In general, CashFlow equals the net profit of the company plus all the expense items not involving cash (such as depreciation, capital losses, etc.) and minus all the profit not involving cash (such as capital gains, etc.). This information can be found in a company's income statement and statement of change in financial position.
- TotalDebt is a Number or formula. In general, TotalDebt is the total short term and long term bank borrowings. Some analysts may even include all the short and long term liabilities as part of total debt. All this information can be found in a company's income statement.

Action

FRCashFlowVsTotalDebt returns the ratio between CashFlow and TotalDebt (for example, CashFlowVsTotalDebt = CashFlow / TotalDebt).

Typical uses

This ratio is used to measure the ability of a company to pay off its total debts with cash flows generated from operations. It is usually measured in number of times. The higher the result, the higher the ability for the company to pay off its debts.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
FRCashFlowVsTotalDebt(45, 30)
```

Returns 1.5.

```
FRCashFlowVsTotalDebt(74.4, 93)
```

Returns 0.8.

Note: None of the arguments can be negative; you may have to check for negative values before passing the arguments.

Comment

FRCashFlowVstotalDebt is one of the many financial analysis tools used in interpreting the financial position of a company. As in the case of all ratios, it has to be used with caution. It can just be used as a clue and not solid proof for forming a judgment on the financial status of a company.

Neither one of the ratios can be used alone for doing financial analysis and there are no fixed rules on the results of the ratios. Apart from differences between type of industries, result varies among different companies within the same industry and within different account periods. The results should only be used for relative comparison and trend analysis, rather than treating them as something absolute.

FRReturnOnEquity (NetProfit, TotalEquity)

Basic and Crystal syntax.

Format

```
FRReturnOnEquity(NetProfit, TotalEquity)
```

Arguments

- NetProfit is a Number or formula. NetProfit is the amount of earnings a company earns during the year from the sale of its products or services minus all the expense items. It is the bottom line figure in a company's income statement. Some analysts may want to use the net profit before extraordinary items and/or income taxes for calculating this ratio.
- TotalEquity is a Number or formula. In general, TotalEquity consist of common stocks, preferred stocks, capital surplus, and retained earnings in a company's balance sheet.

Action

FRReturnOnEquity returns the ratio between NetProfit and TotalEquity (for example, ReturnOnEquity = NetProfit / TotalEquity).

Typical uses

This ratio is used to measure the returns on the total capital employed to run the company. The higher the result, the higher the return.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
FRReturnOnEquity(34.5,150)
```

Returns 0.23.

```
FRReturnOnEquity(12, 15)
```

Returns 0.8.

Note: None of the arguments can be negative; you may have to check for negative values before passing the arguments.

Comment

FRReturnOnEquity is one of the many financial analysis tools used in interpreting the financial position of a company. As in the case of all ratios, it has to be used with caution. It can just be used as a clue and not solid proof for forming a judgment on the financial status of a company.

Neither one of the ratios can be used alone for doing financial analysis and there are no fixed rules on the results of the ratios. Apart from differences between type of industries, result varies among different companies within the same industry and within different account periods. The results should only be used for relative comparison and trend analysis, rather than treating them as something absolute.

FRReturnOnNetFixedAssets (NetProfit, NetFixedAssets)

Basic and Crystal syntax.

Format

```
FRReturnOnNetFixedAssets(NetProfit, NetFixedAssets)
```

Arguments

- NetProfit is a Number or formula. NetProfit is the amount of earnings a company earns during the year from the sale of its products or services minus all the expense items. It is the bottom line figure in a company's income statement. Some analysts may want to use the net profit before extraordinary items and/or income taxes for calculating this ratio.
- NetFixedAssets is a Number or formula. NetFixedAssets equals total fixed assets minus accumulated depreciation in a company's balance sheet.

Action

FRReturnOnNetFixedAssets returns the ratio between NetProfit and NetFixedAssets (for example, ReturnOnNetFixedAssets = NetProfit / NetFixedAssets).

Typical uses

This ratio is used to measure the return on capital assets employed. This ratio is normally used for measuring the efficiency of manufacturing companies or servicing industries that required employment of capital assets to provide its services (airlines, etc.). The higher the ratio the better the return.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
FRReturnOnNetFixedAssets(34, 50)
```

Returns 0.68.

```
FRReturnOnNetFixedAssets(50.4, 42)
```

Returns 1.2.

Note: None of the arguments can be negative; you may have to check for negative values before passing the arguments.

Comment

FRReturnOnNetFixedAssets is one of the many financial analysis tools used in interpreting the financial position of a company. As in the case of all ratios, it has to be used with caution. They can just be used as a clue and not solid proof for forming a judgment on the financial status of a company.

Neither one of the ratios can be used alone for doing financial analysis and there are no fixed rules on the results of the ratios. Apart from differences between type of industries, result varies among different companies within the same industry and within different account periods. The results should only be used for relative comparison and trend analysis, rather than treating them as something absolute.

FRReturnOnTotalAssets (NetProfit, TotalAssets)

Basic and Crystal syntax.

Format

```
FRReturnOnTotalAssets(NetProfit, TotalAssets)
```

Arguments

- NetProfit is a Number or formula. NetProfit is the amount of earnings a company earns during the year from the sale of its products or services minus all the expense items. It is the bottom line figure in a company's income statement. Some analysts may want to use the net profit before extraordinary items and/or income taxes for calculating this ratio.
- TotalAssets is a Number or formula. In general, TotalAssets consist of total current assets, net fixed assets and all other assets in a company's balance sheet.

Action

FRReturnOnTotalAssets returns the ratio between NetProfit and TotalAssets (for example, ReturnOnTotalAssets = NetProfit / TotalAssets).

Typical uses

This measures the return on total assets employed. It is a measure of the efficiency of the company in employing its total assets. The higher the result the better is its efficiency.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
FRReturnOnTotalAssets(23, 100)
```

Returns 0.23.

```
FRReturnOnTotalAssets(442, 340)
```

Returns 1.3.

Note: None of the arguments can be negative; you may have to check for negative values before passing the arguments.

Comment

FRReturnOnTotalAssets is one of the many financial analysis tools used in interpreting the financial position of a company. As in the case of all ratios, it has to be used with caution. They can just be used as a clue and not solid proof for forming a judgment on the financial status of a company.

Neither one of the ratios can be used alone for doing financial analysis and there are no fixed rules on the results of the ratios. Apart from differences between type of industries, result varies among different companies within the same industry and within different account periods. The results should only be used for relative comparison and trend analysis, rather than treating them as something absolute.

FRReturnOnInvestedCapital (NetProfit, TotalBankDebt, TotalEquity)

Basic and Crystal syntax.

Format

```
FRReturnOnInvestedCapital(NetProfit, TotalBankDebts, TotalEquity)
```

Arguments

- NetProfit is a Number or formula. NetProfit is the amount of earnings a company earns during the year from the sale of its products or services minus all the expense items. It is the bottom line figure in a company's income statement. Some analysts may want to use the net profit before extraordinary items and/or income taxes for calculating this ratio.
- TotalBankDebts is a Number or formula. TotalBankDebts includes all the short and long term bank debts of a company. They can all be found in a company's balance sheet.
- TotalEquity is a Number or formula. In general, TotalEquity consists of common stocks, preferred stocks, capital surplus, and retained earnings in a company's balance sheet.

Action

FRReturnOnInvestedCapital returns the ratio between NetProfit and InvestedCapital (which is TotalBankDebts plus TotalEquity) (for example, ReturnOnInvested Capital = NetProfit / (TotalBankDebts + TotalEquity)).

Typical uses

This ratio is used to measure the return on a company's invested capital, which includes total bank borrowing and the company's total equity. The higher the return the better.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
FRReturnOnInvestedCapital(23, 17, 33)
```

Returns 0.46.

```
FRReturnOnInvestedCapital(54, 23, 17)
```

Returns 1.35.

Note: None of the arguments can be negative; you may have to check for negative values before passing the arguments.

Comment

FRReturnOnInvestedCapital is one of the many financial analysis tools used in interpreting the financial position of a company. As in the case of all ratios, it has to be used with caution. They can just be used as a clue and not solid proof for forming a judgment on the financial status of a company.

Neither one of the ratios can be used alone for doing financial analysis and there are no fixed rules on the results of the ratios. Apart from differences between type of industries, result varies among different companies within the same industry and within different account periods. The results should only be used for relative comparison and trend analysis, rather than treating them as something absolute.

FRReturnOnCommonEquity (NetProfit, PreferredDividend, CommonEquity)

Basic and Crystal syntax.

Format

```
FRReturnOnCommonEquity(NetProfit, PreferredDividend, CommonEquity)
```

Arguments

- NetProfit is a Number or formula. NetProfit is the amount of earnings a company earns during the year from the sale of its products or services minus all the expense items. It is the bottom line figure in a company's income statement. Some analysts may want to use the net profit before extraordinary items and/or income taxes for calculating this ratio.
- PreferredDividend is a Number or formula. PreferredDividend is the total dividend paid out to the preferred shareholders during the accounting period being studied. This information can be found in a company's income statement and statement of change in financial position. It can also be derived in a company's balance sheet.
- CommonEquity is a Number or formula. In general, CommonEquity is the sum of all common shares, contributed surplus, and retained earnings. All of these can be found in a company's balance sheet.

Action

FRReturnOnCommonEquity returns the ratio between the net profit distributable to common shareholders (NetProfitreferredDividend) and CommonEquity (for example, ReturnOnCommonEqutiy = (NetProfit - PreferredDividend) / CommonEquity).

Typical uses

This ratio is used to measure the return on common equity employed. Note that preferred dividend is deducted prior to doing the calculation. The higher the result the higher the return.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
FRReturnOnCommonEquity(340, 40, 600)
```

Returns 0.5.

```
FRReturnOnCommonEquity(245, 50, 150)
```

Returns 1.3.

Note:

- None of the arguments can be negative; you may have to check for negative values before passing the arguments.
- The Formula Editor will automatically check to see if NetProfit is smaller than PreferredDividend. If NetProfit is smaller than PreferredDividend, you will receive an error message.

Comment

FRReturnOnCommonEquity is one of the many financial analysis tools used in interpreting the financial position of a company. As in the case of all ratios, it has to be used with caution. It can just be used as a clue and not solid proof for forming a judgment on the financial status of a company.

Neither one of the ratios can be used alone for doing financial analysis and there are no fixed rules on the results of the ratios. Apart from differences between type of industries, result varies among different companies within the same industry and within different account periods. The results should only be used for relative comparison and trend analysis, rather than treating them as something absolute.

FREarningsPerCommonShare (NetProfit, PreferredDividend, NumOfCommonShare)

Basic and Crystal syntax.

Format

```
FREarningsPerCommonShare(NetProfit, PreferredDividend,
NumOfCommonShare)
```

Arguments

- NetProfit is a Number or formula. NetProfit is the amount of earnings a company earns during the year from the sale of its products or services minus all the expense items. It is the bottom line figure in a company's income statement. Some analysts may want to use the net profit before extraordinary items and/or income taxes for calculating this ratio.
- PreferredDividend is a Number or formula. PreferredDividend is the total dividend paid out to the preferred shareholders during the accounting period being studied. This information can be found in a company's income statement and statement of change in financial position. It can also be derived in a company's balance sheet.
- NumOfCommonShare is a Number or formula. In general, NumOfCommonShare is the number of common shares issued and outstanding. This information can be found in a company's balance sheet.

Action

FREarningsPerCommonShare returns the ratio between net profit distributable to common shareholders (NetProfitreferredDividend) and NumOfCommonShare (for example, EarningPerCommonShare = (NetProfit - PreferredDividend) / NumOfCommonShare).

Typical uses

This ratio is used to calculate how much profit that a common share is entitled to. This ratio is used side by side with dividend per common share to determine the dividend pay out ratio. Some analysts may want to perform a fully diluted earnings per share. In this case, they have to take into account of all the common shares that will be converted from all the convertible preferred shares.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
FREarningsPerCommonShare(550000, 50000, 100000)
```

Returns 5.

```
FREarningsPerCommonShare(450000, 10000, 2000000)
```

Returns 0.22.

Note:

- None of the arguments can be negative; you may have to check for negative values before passing the arguments.
- NetProfit can not be smaller that PreferredDividend, otherwise you will receive an error message.

Comment

FREarningsPerCommonShare is one of the many financial analysis tools used in interpreting the financial position of a company. As in the case of all ratios, it has to be used with caution. It can just be used as a clue and not solid proof for forming a judgment on the financial status of a company.

Neither one of the ratios can be used alone for doing financial analysis and there are no fixed rules on the results of the ratios. Apart from differences between type of industries, result varies among different companies within the same industry and within different account periods. The results should only be used for relative comparison and trend analysis, rather than treating them as something absolute.

FRAccRecTurnover (AccountReceivable, Sales, NumOfDays)

Basic and Crystal syntax.

Format

```
FRAccRecTurnover(AccountReceivable, Sales, NumOfDays)
```

Arguments

- AccountReceivable is a Number or formula. AccountReceivable is the total accounts receivable in a company's balance sheet.
- Sales is a Number or formula. Sales is the total sales during the period that appears in a company's income statement.
- NumOfDays is a Number or formula. In general, it is the number of days in a year. Some analysts use 360 and some use 365. The default is 360 for this function.

Action

FRAccRecTurnover returns the average turnover of AccountReceivable in days (for example, AccRecTurnover = (AccountReceivable / Sales) * 360).

Typical uses

This ratio measures the turnover of accounts receivable in days. A smaller ratio represents higher turnover which means that accounts receivable is paid shortly after they are incurred. A higher turnover is generally preferable over a lower turnover which might be an indication of doubtful accounts.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
FRAccRecTurnover(100, 600, 360)
```

Returns 60.

```
FRAccRecTurnover(200, 300, 360)
```

Returns 240

Note: None of the arguments can be negative; you may have to check for negative values before passing the arguments.

Comment

FRAccRecTurnover is one of the many financial analysis tools used in interpreting the financial position of a company. As in the case of all ratios, it has to be used with caution. It can just be used as a clue and not solid proof for forming a judgment on the financial status of a company.

Neither one of the ratios can be used alone for doing financial analysis and there are no fixed rules on the results of the ratios. Apart from differences between type of industries, result varies among different companies within the same industry and within different account periods. The results should only be used for relative comparison and trend analysis, rather than treating them as something absolute.

FRInventoryTurnover (Inventory, Sales, NumOfDays)

Basic and Crystal syntax.

Format

```
FRInventoryTurnover(Inventory, Sales, NumOfDays)
```

Arguments

- Inventory is a Number or formula. Inventory is the total inventories in a company's balance sheet. In general, it includes all raw material, work in progress and finished goods.
- Sales is a Number or formula. Sales is the total sales during the period that appears in a company's income statement.
- NumOfDays is a Number or formula. In general, NumOfDays is the number of days in a year. Some analysts use 360 and some use 365. The default is 360 for this function.

Action

FRInventoryTurnover returns the average turnover of inventory in days (for example, InventoryTurnover = (Inventory/Sales) * 360).

Typical uses

This ratio measures the turnover of inventories in days. A smaller ratio represents higher turnover, which means that inventories are sold quickly and are not stuck in the warehouses. A high turnover is generally preferred over a low turnover, which might be an indication of stale or obsolete inventory.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
FRInventoryTurnover(40, 400, 360)
```

Returns 36.

```
FRInventoryTurnover(36, 90, 360)
```

Returns 144.

Note: None of the arguments can be negative; you may have to check for negative values before passing the arguments.

Comment

FRInventoryTurnover is one of the many financial analysis tools used in interpreting the financial position of a company. As in the case of all ratios, it has to be used with caution. It can just be used as a clue and not solid proof for forming a judgment on the financial status of a company.

Neither one of the ratios can be used alone for doing financial analysis and there are no fixed rules on the results of the ratios. Apart from differences between type of industries, result varies among different companies within the same industry and within different account periods. The results should only be used for relative comparison and trend analysis, rather than treating them as something absolute.

FRPriceEarningsRatio (MarketPrice, EarningsPerShare)

Basic and Crystal syntax.

Format

```
FRPriceEarningsRatio(MarketPrice, EarningsPerShare)
```

Arguments

- MarketPrice is a Number or formula. MarketPrice is the current market price of a company's common share. This information is publicly available if the company's shares are traded in a stock exchange.
- EarningsPerShare is a Number or formula. EarningsPerShare is the amount of earnings that is attributable to each common share. This information is available in a company's financial reports.

Action

FRPriceEarningsRatio returns the ratio between MarketPrice and EarningsPerShare (for example, PriceEarningsRatio = MarketPrice / EarningsPerShare (12 months)).

Typical uses

This ratio is used as a comparison between the share's market price and its earning ability. It is a convenient figure and it can be used easily to compare one company's earning ability with another.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
FRPriceEarningsRatio(3.5, 2)
```

Returns 1.75.

```
FRPriceEarningsRatio(27, 1.5)
```

Returns 18.

Note: None of the arguments can be negative; you may have to check for negative values before passing the arguments.

Comment

FRPriceEarningsRatio is one of the many financial analysis tools used in interpreting the financial position of a company. As in the case of all ratios, it has to be used with caution. It can just be used as a clue and not solid proof for forming a judgment on the financial status of a company.

Neither one of the ratios can be used alone for doing financial analysis and there are no fixed rules on the results of the ratios. Apart from differences between type of industries, result varies among different companies within the same industry and within different account periods. The results should only be used for relative comparison and trend analysis, rather than treating them as something absolute.

FRDividendYield (Dividend, MarketPrice)

Basic and Crystal syntax.

Format

```
FRDividendYield(Dividend, MarketPrice)
```

Arguments

- Dividend is a Number or formula. Dividend is the indicated annual dividend per common share.
- MarketPrice is a Number or formula. MarketPrice is the current market price of a company's common share. This information is publicly available if a company's shares are traded in a stock exchange.

Action

FRDividendYield returns the ratio between Dividend and MarketPrice (for example, DividendYield = Dividend / MarketPrice).

Typical uses

This ratio is used as an indication of the return on investment for a common share investor. It is a superficial comparison between shares of different companies. Some analysts may also want to use the same function for calculating yield for preferred share.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
FRDividendYield(3,25)
```

Returns 0.12.

```
FRDividendYield(5, 80)
```

Returns 0.0625.

Note: None of the arguments can be negative; you may have to check for negative values before passing the arguments.

Comment

FRDividendYield is one of the many financial analysis tools used in interpreting the financial position of a company. As in the case of all ratios, it has to be used with caution. It can just be used as a clue and not solid proof for forming a judgment on the financial status of a company.

Neither one of the ratios can be used alone for doing financial analysis and there are no fixed rules on the results of the ratios. Apart from differences between type of industries, result varies among different companies within the same industry and within different account periods. The results should only be used for relative comparison and trend analysis, rather than treating them as something absolute.

Now

Basic and Crystal syntax.

Returns

Text String

Action

Now prints the current time on a report. The time is taken from your computer's internal clock.

Typical Uses

Use Now any time you want to time-stamp your report.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
Now
```

Returns 4:45:18 if you begin printing your report at 4:45:18.

```
Now
```

Returns 17:14:33 if you begin printing your report at 17:14:33.

Comments

- The time format is determined by the settings in the International section of the Windows Control Panel.
- The CurrentTime function is identical to the Now function, which was included with previous version of the program. Now is provided for compatibility with reports created with earlier versions of the program. New reports should use the CurrentTime function.

Picture (string, picture)

Basic and Crystal syntax.

Arguments

- String is a text string to be formatted according to the picture format.
- Picture is a text string representing the way you want the characters in the string to be printed.

Return

string

Action

Picture (string, picture) prints a string or values in a text string in a predetermined format.

Typical Uses

Picture can be used to print a string "2065555555" in a telephone number format "(206) 555555", or to print other strings in fixed formats.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
Picture ({customer.FAX},"(xxx) xxxxxx" )
```

Prints the values in the Fax field in the format (805) 555555, (313) 555555, etc.

```
Picture ({customer.REGION}, "[xx]")
```

Prints the values in the Region field in the format [AK], [CA], etc.

Comments

- Picture must be enclosed in quotes.
- Use the character "x" in the mask to designate characters from the string.
- If the database field includes brackets or other formatting, for example (xxx) xxxxxx, you will not be able to convert it to xxxxxxx. The parentheses and hyphen will still appear.

LooksLike (string, mask)

Basic and Crystal syntax.

Arguments

- string is the text string or field containing text string values that are being compared to the mask.
- mask is a text string that provides a mask for comparing the value in the string argument.

Returns

Boolean Value

Action

Enables you to locate field values using a standard DOS wildcard (**?** = wildcard for single character, * = wildcard for any number of characters). It does this by comparing a string to a mask that contains one or more wildcards. The function returns True if the string matches the mask and False if the string does not match the mask.

Typical uses

Use this function whenever you want to locate records based on two or more field values that are spelled differently yet look alike (have an essentially similar structure).

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
LooksLike ("Snyder","Snder")
```

Returns TRUE.

```
LooksLike ("Schneider","Sder")
```

Returns FALSE.

```
LooksLike ("Schneider", "Sder")
```

Returns TRUE.

The following example is applicable to Crystal syntax:

If LooksLike ({customer.CONTACT FIRST NAME},"ris" Then

{customer.CONTACT FIRST NAME}

Else

""

Returns Cris, Kris, and Iris, but not Chris (assuming all of these values are in the {customer.CONTACT FIRST NAME} database).

Soundex (string)

Basic and Crystal syntax.

Arguments

string is one of two or more strings that sound alike.

Returns

Text String

Action

Soundex (string) evaluates a text string and returns a four character value that symbolizes the way string sounds.

Typical uses

Use this function whenever you want to locate records based on two or more field values that are spelled differently yet sound alike.

This function can also be used to find customer names that have been misspelled.

Examples

The following examples are applicable to Crystal syntax:

If Soundex{customer.LAST NAME} = Soundex ("Snider") Then

{customer.LAST NAME}

Else

""

Runs a Soundex on each value in the Last Name field and prints only those values that have a Soundex value equal to the Soundex value of the name Snider. This formula returns the names Sneider, Schneider, Snyder, and Snider if they are all in the database.

Comments

- The value string must be in quotes.
- Soundex works only with values that begin with the same letter: For example, Soundex will return the same value for Chris and Cris (C620) but not for Kris (K620)
- Soundex creates a code based on the first character in the string plus three characters based on the following:
- Non-alpha characters (i.e. numbers and punctuation) become .
- The characters a, e, i, o, u, y, h, and w are ignored (except when they are the first character in the original string).
- The characters b, f, p, and v become 1.
- The characters c, g, j, k, q, s, x, and z become 2.
- The characters d and t become 3.

- The character l becomes 4.
- The characters m and n become 5.
- The character r becomes 6.
- If the resulting code is only 2 or 3 characters long, Soundex uses zeros to fill out the code to four characters. For example, in the name Lauren, only the L, r, and n are translated (Lrn), so the resulting Soundex code becomes L650.
- If the resulting code is more than four characters long, all characters after the fourth character are ignored. For example, in the name Patrick, the P, t, r, c, and k can be translated (Ptrck), but the resulting Soundex code will only be four characters: P362.

DateTimeTo2000 (dateTime, number)

Basic and Crystal syntax.

Arguments

- dateTime: accepts only valid date-time fields, with either 2 digit or 4 digit years.
- a database field defined as a date-time field. File|Report Options should be set to Convert Date-time fields to Date-time Field. (This option only works for ODBC datasources in 6.0.)
- the CDateTime function (CDateTime(yyyy, MM, dd, hh, mm, ss)). A four digit year is required by this function. If two digits are entered it will assume 00xx, example: 98 AD.
- Number: a number between 0 and 99 corresponding with the desired windowing year.

Returns

a date-time field with a four digit year

Action

- 2 digit years (xx):

  If the Year value is greater than the windowing number, 19 is appended before the 2 digits (19xx). If the Year value is less than or equal to the windowing number, 20 is appended (20xx).

- 4 digit years (19xx):

  If the last two digits in the Year value are greater than the windowing number, the Year is preserved as found in the date field (19xx). If the two digits in the Year value are less than or equal to the windowing number, the first two digits are changed to 20 (20xx). If the first two digits in the year field are 20, the Year is preserved as found in the date field (20xx).

Note: If the year is less than or equal to 1899 and greater or equal to 100, then there will be no change to the date.

Examples

The following examples are applicable to Basic and Crystal syntax:

Here the window value is greater than the year and so it will change it to 20XX.

`DateTimeTo2000(CDateTime(1998,12,12,3,2,1), 99)`

should return 2098/12/12 3:02:01

Here the window value is less than the year and so it will not change the year.

`DateTimeTo2000(CDateTime(1995,1,2,3,2,1), 94)`

should return 1995/01/02 3:02:01

Other Examples

These are examples of a date that will not be affected by the function.

`DateTimeTo2000(CDateTime(1899,12,12,5,6,7), 99)`

should return 1899/12/12 5:06:07

`DateTimeTo2000(CDateTime(999,12,12,5,6,7), 99)`

should return 999/12/12 5:06:07

Some databases maintain the year of a date as a two digit field and so these samples will simulate that date field.

`DateTimeTo2000(CDateTime (93,12,12,5,6,7), 96)`

should return 2093/12/12 5:06:07

`DateTimeTo2000(CDateTime (98,12,12,5,6,7), 50)`

should return 1998/12/12 5:06:07

`DateTimeTo2000(CDateTime (9,12,12,5,6,7), 10)`

should return 2009/12/12 5:06:07

`DateTimeTo2000(CDateTime (2,12,12,5,6,7), 1)`

should return 1902/12/12 5:06:07

DTSToDateTime (DateTimeString)

Basic and Crystal syntax.

Arguments

DateTimeString is a string including a date and a Time value.

Returns

DateTime value

Action

Evaluates the string specified and returns a DateTime data type.

Examples

`DTSToDateTime ("2000/01/13 11:30:15")`

Returns the Date value, January 13, 2000 11:30:15 am. The month must be entered as 01 rather than just 1.

Comments

For new formulas, use the CDateTime or DateTimeValue functions instead of DTSToDateTime. The CDateTime and DateTimeValue functions are able to convert more types of strings.

DTSToTimeField (DateTimeString)

Basic and Crystal syntax.

Arguments

DTSToTimeField is a string including a Date and a Time value.

Returns

Time value

Action

Evaluates the string specified and returns a Time data type.

Examples

```
DTSToTimeField ("2000/01/13 11:30:15")
```

Returns the Time value, 11:30:15 am. The month must be entered as 01 rather than just 1.

Comments

For new formulas, use the CTime or TimeValue functions instead of DTSToTimeField. The CTime and TimeValue functions are able to convert more types of strings.

DateTimeToDate (dateTime)

Basic and Crystal syntax.

Arguments

dateTime is a DateTime value.

Returns

Date value

Action

Evaluates the DateTime value specified and converts it to a Date value.

Examples

The following example is applicable to both Basic and Crystal syntax:

```
DateTimeToDate(#11/01/1998 11:15:00 AM#)
```

Returns the Date value 11/01/1998.

Comments

For new formulas, you can use the CDate or DateValue functions instead of DateTimeToDate when converting a DateTime value to a Date value. The CDate and DateValue functions are more efficient since they are not UFLs.

DateTimeToTime (dateTime)

Basic and Crystal syntax.

Arguments

dateTime is a DateTime value.

Returns

Time value

Action

Evaluates the DateTime value specified and converts it to a Time value.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
DateTimeToTime(#11/01/1998 11:15:00 AM#)
```

Returns 11:15:00 AM.

Comments

For new formulas, you can use the CTime or TimeValue functions instead of DateTimeToTime when converting a DateTime value to a Time value. The CTime and TimeValue functions are more efficient since they are not UFLs.

DateTimeToSeconds (dateTime)

Basic and Crystal syntax.

Arguments

dateTime is a DateTime value.

Returns

Time value (in seconds)

Action

Evaluates the DateTime value and returns the number of seconds that have passed between 00:00:00 (12:00 midnight) and the specified time.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
DateTimeToSeconds(#11/01/1998 11:30:15 AM#)
```

Returns 41415.

Comments

Instead of using DateTimeToSeconds (dt), you can use:

```
DateDiff ("s", #12:00 am#, CDateTime (CDate(0,0,0), CTime (dt)))
```

The above expression does not use UFLs and can be more efficient.

Functions to duplicate group fields

Use these functions whenever you want to duplicate a group field in a formula.

Note: In order to use these functions, you must have already entered a group field in your report with identical parameters to those you plan to reproduce in the formula. For example, the same field, same sortndroupy field, same date/Boolean condition (if applicable), or same action (average, count).

Sort and group by field = character, number, or dollar value

Average (fld, condFld)

See Average (fld), Average (fld, condFld), Average (fld, condFld, cond), Average (x)

Count (fld, condFld)

See Count (fld), Count (fld, condFld), Count (fld, condFld, cond), Count (x)

DistinctCount (fld, condFld)

See DistinctCount (fld), DistinctCount (fld, condFld), DistinctCount (fld, condFld, cond), DistinctCount (x)

Maximum (fld, condFld)

See Maximum (fld), Maximum (fld, condFld), Maximum (fld, condFld, cond), Maximum (x)

Minimum (fld, condFld)

See Minimum (fld), Minimum (fld, condFld), Minimum (fld, condFld, cond), Minimum (x)

PopulationStdDev (fld, condFld)

See PopulationStdDev (fld), PopulationStdDev (fld, condFld), PopulationStdDev (fld, condFld, cond), PopulationStdDev (x)

PopulationVariance (fld, condFld)

See PopulationVariance (fld), PopulationVariance (fld, condFld), PopulationVariance (fld, condFld, cond), PopulationVariance (x)

StdDev (fld, condFld)

See StdDev (fld), StdDev (fld, condFld), StdDev (fld, condFld, cond), StdDev (x)

Sum (fld, condFld)

See Sum (fld), Sum (fld, fld), Sum (fld, condFld), Sum (fld, condFld, cond), Sum (x)

Variance (fld, condFld)

See Variance (fld), Variance (fld, condFld), Variance (fld, condFld, cond), Variance (x)

Sort and group by field = date or Boolean

Average (fld, condFld, cond)

See Average (fld), Average (fld, condFld), Average (fld, condFld, cond), Average (x)

Count (fld, condFld, cond)

See Count (fld), Count (fld, condFld), Count (fld, condFld, cond), Count (x)

DistinctCount (fld, condFld, cond)

See DistinctCount (fld), DistinctCount (fld, condFld), DistinctCount (fld, condFld, cond), DistinctCount (x)

Maximum (fld, condFld, cond)

See Maximum (fld), Maximum (fld, condFld), Maximum (fld, condFld, cond), Maximum (x)

Minimum (fld, condFld, cond)

See Minimum (fld), Minimum (fld, condFld), Minimum (fld, condFld, cond), Minimum (x)

PopulationStdDev (fld, condFld, cond)

See PopulationStdDev (fld), PopulationStdDev (fld, condFld), PopulationStdDev (fld, condFld, cond), PopulationStdDev (x)

PopulationVariance (fld, condFld, cond)

See PopulationVariance (fld), PopulationVariance (fld, condFld), PopulationVariance (fld, condFld, cond), PopulationVariance (x)

StdDev (fld, condFld, cond)

See StdDev (fld), StdDev (fld, condFld), StdDev (fld, condFld, cond), StdDev (x)

Sum (fld, condFld, cond)

See Sum (fld), Sum (fld, fld), Sum (fld, condFld), Sum (fld, condFld, cond), Sum (x)

Variance (fld, condFld, cond)

See Variance (fld), Variance (fld, condFld), Variance (fld, condFld, cond), Variance (x)

Grand total functions

Use these functions whenever you want to evaluate all the values in a given field for the entire report and return a calculated value. For example, a "grand total" average or a "grand total" count.

Average (fld)

See Average (fld), Average (fld, condFld), Average (fld, condFld, cond), Average (x)

Count (fld)

See Count (fld), Count (fld, condFld), Count (fld, condFld, cond), Count (x)

DistinctCount (fld)

See DistinctCount (fld), DistinctCount (fld, condFld), DistinctCount (fld, condFld, cond), DistinctCount (x)

Maximum (fld)

See Maximum (fld), Maximum (fld, condFld), Maximum (fld, condFld, cond), Maximum (x)

Minimum (fld)

See Minimum (fld), Minimum (fld, condFld), Minimum (fld, condFld, cond), Minimum (x)

PopulationStdDev (fld)

See PopulationStdDev (fld), PopulationStdDev (fld, condFld), PopulationStdDev (fld, condFld, cond), PopulationStdDev (x)

PopulationVariance (fld)

See PopulationVariance (fld), PopulationVariance (fld, condFld), PopulationVariance (fld, condFld, cond), PopulationVariance (x)

StdDev (fld)

See StdDev (fld), StdDev (fld, condFld), StdDev (fld, condFld, cond), StdDev (x)

Sum (fld)

See Sum (fld), Sum (fld, fld), Sum (fld, condFld), Sum (fld, condFld, cond), Sum (x)

Variance (fld)

See Variance (fld), Variance (fld, condFld), Variance (fld, condFld, cond), Variance (x)

Conditional formatting functions

GridRowColumnValue (name)

CurrentFieldValue

DefaultAttribute

RGB (red, green, blue)

GridRowColumnValue (name)

Basic and Crystal syntax.

Arguments

name is a String value which is an Alias for Formulas. The Alias for Formulas is a name you have specified for a level of dimension in an OLAP grid or a cross-tab.

Returns

Value of a level of dimension in an OLAP grid or cross-tab, that has been specified by the argument, name. The type of the value thus depends on the type of the specified dimension.

Action

GridRowColumnValue returns the value of a specified level of dimension in an OLAP grid or cross-tab.

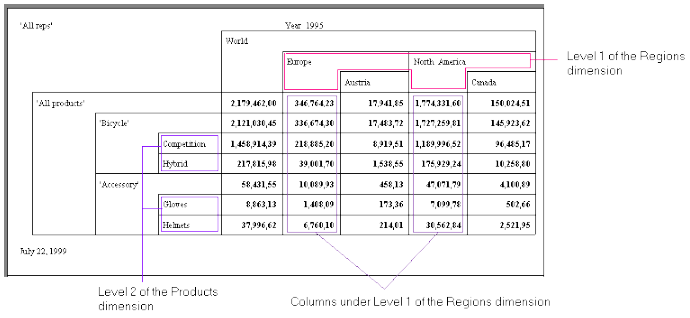
Typical uses

To format cells under a level of dimension in an OLAP grid or cross-tab, based on the value of that level of dimension. For instance, you can format the font color for cells in a level of dimension based on the value of that level of dimension. The attributes of a cell that you can format conditionally using a formula include: font, border, object positioning, etc. They are accessible through the Format Editor. See Conditional attribute properties.

Examples

The following example is applicable to both Crystal and Basic syntax:

Assume an OLAP grid with Products in the rows, and Regions in the columns.


Level 1 of the Regions dimension
Level 2 of the Products dimension
Columns under Level 1 of the Regions dimension

'All reps'

			Year 1995				
			World				
				Europe		North America	
					Austria		Canada
'All products'			2,179,462,00	346,764,23	17,941,85	1,774,331,60	150,024,51
	'Bicycle'		2,121,030,45	336,674,30	17,483,72	1,727,259,81	145,923,62
		Competition	1,458,914,39	218,885,20	8,919,51	1,189,996,52	96,485,17
		Hybrid	217,815,98	39,001,70	1,538,55	175,929,24	10,258,80
	'Accessory'		58,431,55	10,089,93	458,13	47,071,79	4,100,89
		Gloves	8,863,13	1,408,09	173,36	7,099,78	502,66
		Helmets	37,996,62	6,760,10	214,01	30,562,84	2,521,95

July 22, 1999

Suppose you would like to use GridRowcolumnValue to conditionally format cells under Level 1 of the Regions dimension and Level 2 of the Products dimension. In particular, you want to format the font color of the total sales of individual products for all of Europe to be red, and that under North America to be blue.

1. Set up an Alias for Formulas for Level 1 of the Regions dimension to be "Continent". (You can do so using the OLAP Report Expert or Format OLAP Grid Object command. See Formatting the grid.

   After setting up the necessary Alias for Formulas, you can then express the formatting condition as: if GridRowColumnValue("Continent") returns "Europe", then you would like the font color of the cells for each Level 2 input of the Products dimension to be red, otherwise, since there are only two continents in this scenario, the font color should be blue.

2. Click in the OLAP grid on one of the cells representing total sales for Europe for a particular (Level 2) product.

   The cells for all Level 2 inputs for Products combined with all Level 1 inputs for Regions are also selected. Formatting done to one of these cells will be carried over to all other selected cells.

3. Use the Format Field command to enter a conditional formatting formula for the selected cells. See Working with conditional formatting.

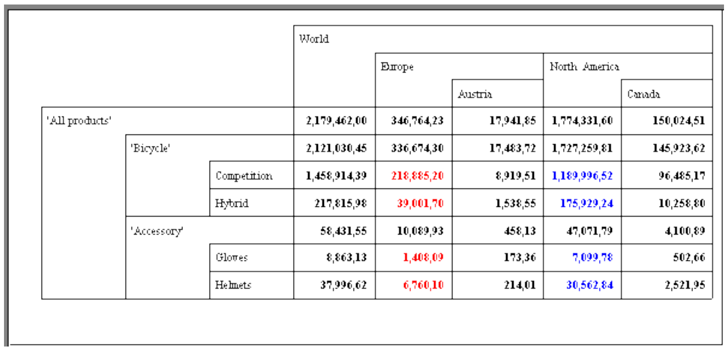
   In Crystal syntax, you can enter a formula like:

```
If GridRowColumnValue("Continent") = "Europe" Then crRed Else crBlue
```

In Basic syntax, the formula will be:

```
If GridRowColumnValue("Continent") = "Europe" Then formula = crRed
Else formula = crBlue
```

You will see the total sales of individual products for Europe highlighted in red, and the total sales of individual products for North America highlighted in blue.



			World				
				Europe		North America	
					Austria		Canada
'All products'			2,179,462,00	346,764,23	17,941,85	1,774,331,60	150,024,51
	'Bicycle'		2,121,030,45	336,674,30	17,483,72	1,727,259,81	145,923,62
		Competition	1,458,914,39	218,885,20	8,919,51	1,189,996,52	96,485,17
		Hybrid	217,815,98	39,001,70	1,538,55	175,929,24	10,258,80
	'Accessory'		58,431,55	10,089,93	458,13	47,071,79	4,100,89
		Gloves	8,863,13	1,408,09	173,36	7,099,78	502,66
		Helmets	37,996,62	6,760,10	214,01	30,562,84	2,521,95

CurrentFieldValue

Basic and Crystal syntax.

Returns

Can return any field type

Action

CurrentFieldValue returns the current value of the field that is about to be printed.

Typical Uses

You can use CurrentFieldValue to help do conditional formatting on a field element, field object or cross-tab summary or label.

Examples

In Crystal syntax, you can enter a formula like:

```
if CurrentFieldValue = 0 then crRed
```

In Basic syntax, the formula will be:

```
If CurrentFieldValue = 0 Then formula = crRed
```

Formats the font color of the object to print red when the value of the field is zero.

Comments

- CurrentFieldValue is only available when entering field formatting formulas.
- The return type depends on the context of the formula. For example, if you are formatting a string field, CurrentFieldValue returns a string type, if you are formatting a date field, CurrentFieldValue returns a date type.

DefaultAttribute

Basic and Crystal syntax.

Returns

Can return any field type

Action

The value returned by DefaultAttribute depends upon the value selected for the attribute being formatted in the attribute value interface.

Examples

DefaultAttribute can be used in conjunction with CurrentFieldValue to conditionally format fields based upon their values and default condition.

The following examples are applicable to Crystal syntax:

```
if CurrentFieldValue < 0 then DefaultAttribute else crBlack
```

In Basic syntax, the formula will be:

If CurrentFieldValue < 0 Then

formula = DefaultAttribute

Else

formula = crBlack

End If

Formats the font color of the object to print in the default color when the value of the field is less than zero and to print black when the value of the field is greater than zero.

Comments

DefaultAttribute is only available when entering formatting formulas.

RGB (red, green, blue)

RGB and Color are equivalent functions. However, RGB is preferred in Basic syntax and Color is preferred in Crystal syntax.

Arguments

- red is a Number value representing the red component of the resultant color, value is between 0 and 255.
- green is a Number value representing the green component of the resultant color, value is between 0 and 255.
- blue is a Number value representing the blue component of the resultant color, value is between 0 and 255.

Returns

Font color attribute.

Action

RGB returns the color attribute as specified by the red, green and blue components.

Typical use

To specify a custom color when writing a conditional formatting formula for the font or background color of a report field.

Example

The following example is applicable to Basic syntax:

If CurrentFieldValue > 10000 Then

formula = RGB (150,230,150)

Else

formula = RGB (10,19,200)

End If

This formula formats a Number field to display in the color as specified by RGB (150,230,150) if its current value is greater than 10000, otherwise to display it in RGB (10,19,200).

Related topics

Color

Color

Color and RGB (red, green, blue) equivalent functions. However, RGB is preferred in Basic syntax and Color is preferred in Crystal syntax.

Arguments

- red is a Number value representing the red component of the resultant color, value is between 0 and 255.
- green is a Number value representing the green component of the resultant color, value is between 0 and 255.
- blue is a Number value representing the blue component of the resultant color, value is between 0 and 255.

Returns
Font color attribute.

Action
Color returns the color attribute as specified by the red, green and blue components.

Typical use
To specify a custom color when writing a conditional formatting formula for the font or background color of a report field.

Example
The following example is applicable to Crystal syntax:

if CurrentFieldValue > 10000 then

Color (150,230,150)

else

Color (10,19,200)

This formula formats a Number field to display in the color as specified by Color (150,230,150) if its current value is greater than 10000, otherwise to display it in Color (10,19,200).

Related topics
RGB (red, green, blue)

Business Calendar

IsBusinessDay (date)

NextNBusinessDays (#days)

Note: These functions assume a Business Calendar has been set up for your system. Currently, the Business Calendar is only supported in Seagate Info but not in Crystal Reports. If you are using this function in Crystal Reports, or if you have not specified a Business Calendar for your Seagate Info system, this function assumes everyday is a business day.

IsBusinessDay (date)

Basic and Crystal syntax.

Note: This function assumes a Business Calendar has been set up for your system. Currently, the Business Calendar is only supported in Seagate Info but not in Crystal Reports. If you are using this function in Crystal Reports, or if you have not specified a Business Calendar for your Seagate Info system, this function assumes everyday is a business day.

Arguments
(date) is a Date value.

Returns
Boolean value

Action

Evaluates the date specified in the argument, and returns TRUE if the Date value falls on a business day.

Typical Uses

You can use this function in a record selection formula to limit the report to records for business days, or to records that are not for business days.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
IsBusinessDay({Customer.ORDER DATE})
```

Returns TRUE, where {Customer.ORDER DATE} has a value of 10/14/98, and is specified to be a business day according the usiness Calendar for your system.

```
IsBusinessDay({Customer.ORDER DATE})
```

Returns FALSE, where {Customer.ORDER DATE} has a value of 10/18/98 which is a Sunday, and is specified to be a non-business day according to the Business Calendar for your system.

Comments

Currently, this function only accepts a date but not a dateTime field as an argument.

NextNBusinessDays (#days)

Basic and Crystal syntax.

Note: Currently only Seagate Info supports the Business Calendar, Since Crystal Reports does not support the Business Calendar, if you are using this function in Crystal Reports, or if you have not specified a Business Calendar for your Seagate Info system, this function assumes everyday is a business day.

Arguments

#days is a positive, whole number indicating the number of business days that you would like to know the dates for if you start counting from today.

Returns

An array of dates

Action

NextNBusinessDays (x) returns an array of dates, indicating the next x number of business days as of and including the current date, based on the Business Calendar you have specified for your system.

Typical Uses

You could pass a number of days to this function, and the function would return an array of dates indicating that number of business days as of the current day (according to the program's business calendar).

Examples

The following examples are applicable to both Basic and Crystal syntax:

Assuming the current date is Sept. 24, 1999, which is a Friday, and a Business Calendar specifies no holidays in the rest of September, and only Saturdays and Sundays are non-business days.

```
NextNBusinessDays(5)
```

Returns an array of 5 dates, including Sept. 24, 27, 28, 29 and 30, 1999.

Comments

- (#days) can not exceed either +1000 or 000.
- This function assumes that a Business Calendar indicating the business days, weekends and holidays to have been set up.

Operators

Operators are special symbols or words that describe an operation or an action to take place between two or more values. Operators are used in formulas. The program reads the operators in a formula and performs the actions specified.

Click the appropriate link to jump to that section:

Arithmetic operators

Arithmetic operators are used to calculate *number* or *dollar* values.

Add operator

Basic and Crystal syntax.

Usage

```
x+y
```

Adds values x and y.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
5 + 6
```

Returns 11.

```
{file.QTY1} + {file.QTY2}
```

Returns 1487 where {file.QTY1} = 366 and {file.QTY2} = 1121.

```
{file.AMT1} + {file.AMT2} + {file.AMT3} + {file.AMT4}
```

Returns 20 where {file.AMT1} = 2, {file.AMT2} = 4, {file.AMT3} = 6, {file.AMT4} = 8.

```
{file.CLASS1} + 25
```

Returns 37 where {file.CLASS1} = 12.

```
CDate (1991, 04, 05) + 12
```

Returns the date value April 17, 1991.

Related topics

Formula 8

Formula 16

Subtract operator

Basic and Crystal syntax.

Usage

```
x
```

Subtracts value y from value x.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
244 - 112
```

Returns 132.

```
{file.SALES} - {file.COGS} - {file.S&A} = 214,972
```

Returns 132, where {file.SALES} = 455,031, {file.COGS} = 188,213, and {file.S&A} = 51846.

```
{file.ONHAND} - 877 = 114
```

Returns 114, where {file.ONHAND} = 991.

```
CDate (1991, 04, 15) - 12 = Apr 03 91
```

Returns Apr 03 91.

Related topics

Formula 1

Formula 2

Formula 4

Formula 5

Formula 7

Formula 12

Multiply operator

Basic and Crystal syntax.

Usage

```
x*y
```

Multiplies value x by value y.

Examples

The following examples are is applicable to both Basic and Crystal syntax:

```
2883 * 1999
```

Returns 5,763,117.

```
{file.AMOUNT} * {file.QTY} * {file.DISCOUNT}
```

Returns 26.25 where {file.AMOUNT} = 25.00, {file.QTY} = 7, and {file.DISCOUNT} = .15

```
{file.EXMPT} * 356.00
```

Returns 152,012 where {file.EXMPT} = 427.

Related topics

Formula 1

Formula 3

Formula 4

Formula 12

Formula 16

Divide operator

Basic and Crystal syntax.

Usage

```
x/y
```

Divides value x by value y.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
25/5
```

Returns 5

```
1/3
```

Returns .333333

```
{file.SALES} / {file.FORECAST}
```

Returns .875, where {file.SALES} = 52533, {file.FORECAST} = 60000

```
{file.DAYSDUE} / 5
```

Returns 22, where {file.DAYSDUE} = 110

Note: If the denominator = 0, the report will be halted with a divide by zero protection. If you want to avoid this type of problem, you should put a test in.

The following example is applicable to Crystal syntax:

```
If {file.FORECAST} = 0 Then

0

Else

{file.SALES} / {file.FORECAST}
```

Related topics

Formula 13

Percent operator

Crystal syntax.

Usage

```
x % y
```

Calculates value x as a percentage of value y. That is (x/y) * 100.

Examples

The following examples are applicable to Crystal syntax:

```
{file.BALANCE OUTSTANDING} % {file.CREDIT LIMIT}
```

Returns 30.00, where {file.BALANCE OUTSTANDING} = $1500 and {file.CREDIT LIMIT} = $5000.

```
{file.AMOUNT} % {file.CREDIT LIMIT}
```

Returns 32.26, where {file.AMOUNT} = 2257.87 and {file.CREDIT LIMIT} = 7000.

Note: If the denominator equals 0, the report will be halted with a divide by zero protection. If you want to avoid this type of problem, you should put a test in.

```
If {file.FORECAST} = 0 Then
0
Else
{file.SALES} % {file.FORECAST}
```

Related topics
Formula 6

Integer divide (x \ y)

Basic and Crystal syntax.

Usage

```
x \ y
```

Divides Number value x by Number value y and returns a quotient that is a whole number.

Note: If x or y is not a whole number, then it is first rounded before the operation.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
23 \ 5
```

Returns 4.

```
3 \ 2
```

Returns 1.

Modulus (x Mod y)

Basic and Crystal syntax.

Usage

```
x Mod y
```

Divides x by y and returns a remainder that is a whole number. The Number values x and y can be positive or negative, and they can be fractional. If x or y is fractional, it will be first rounded before the modulus is taken.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
83 Mod 10
```

Returns 3.

```
10 Mod 3
```

Returns 1.

```
0 Mod 3
```

Returns .

```
10 Mod
```

Returns 1.

```
10 Mod 3.5
```

Returns 2 since 3.5 is first rounded to 4 before the modulus is taken.

Negate operator

Basic and Crystal syntax.

Usage

```
[-(x)]
```

Multiplies the value inside the parentheses by .

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
-()
```

Returns 1 since negative times negative equals positive.

```
-(1)
```

Returns since negative times positive equals negative.

```
-(04)
```

Returns 14.

```
-({file.QTYONHND})
```

Returns 144 where {file.QTYONHND} = 44.

```
- (- (158)
```

Returns since 15 - 18 = , -() = +3, -(+3) = .

Related topics

Formula 5

Formula 8

Exponentiate (x ^ y)

Basic and Crystal syntax.

Usage

```
x ^ y
```

Raise x to the power of y.

Note: The Number value y can be fractional, and it can be positive or negative. The Number value x can be fractional, but can be negative only if y is a whole number.

Examples

The following examples are applicable to both Crystal and Basic syntax:

```
3 ^ 2
```

Returns 9.

```
3 ^ .5
```

Returns 1.73205....

```
(.5) ^ 2
```

Returns 12.25. Note that the Exponentiation operator has a higher precedence than Negation, so .5 ^ 2 is different from (.5) ^ 2.

```
3 ^ ()
```

Returns .111...

Note: To express a number to a negative power, enclose the power in parentheses.

Conversion operators

Conversion operators are used to convert one data type to another.

To currency operator

To currency operator

Crystal syntax.

Usage

```
$x
```

Converts x from number to currency.

Examples

The following examples are applicable to Crystal syntax.

Note: Each example assumes you have selected the following format options on the Number tab of the Format Editor: Decimal places = (1.00), Negative sign = (345.00-), Currency symbol = (Float), and Thousands Separator = (1,000.00).

```
$12345678
```

Returns $12,345,678.00.

```
$(123 * 456)
```

Returns $56,088.00

```
$({file.QUANTITY} * 3)
```

Returns $42.00, where {file.QUANTITY} = 14.

```
$({file.MILES} * {file.PLEDGE})
```

$27.95, where {file.MILES} = 13 and {file.PLEDGE} = 2.15.

Note: $ * $ = error. You can not multiply a dollar with a dollar.

Related topics
Formula 5

Comparison operators

Comparison operators are used to compare data in a data field with a constant, with the content of another data field, or with a formula result.

Note: A constant is a fixed value (for example, a text string, date, or a number or dollar value you enter) that remains the same when a calculation is performed.

Equal operator

Not Equal operator

Less Than operator

Greater Than operator

Less or equal operator

Greater or Equal operator

Equal operator

Basic and Crystal syntax.

Usage

```
x = y
```

x is equal to y

The equal operator tells the program to evaluate an expression (x=y) and return a TRUE (if x is equal to y) or FALSE (if x is not equal to y).

Examples

The following examples are applicable to both Crystal and Basic syntax:

```
{file.QUANTITY} = 3
```

TRUE, where {file.QUANTITY} has a value of 3.

```
{file.QUANTITY} = 3
```

FALSE, in all other situations.

```
{file.YTD} = {file.LASTYEARYTD}
```

TRUE, where the value of the field {file.YTD} is identical to the value of the field {file.LASTYEARYTD}.

```
{file.YTD} = {file.LASTYEARYTD}
```

FALSE, in all other situations.

```
({file.SALES} - {file.COGS}) = 22,654
```

TRUE, where calculating the expression {file.SALES}-{file.COGS} produces the value 22,654, for example {file.SALES} = 109,986} and {file.COGS} = 87,332}.

```
({file.SALES} - {file.COGS}) = 22,654
```

FALSE, in all other situations.

```
{customer.LAST NAME} = "Johnson"
```

TRUE, where the text string in the {customer.LAST NAME} field is "Johnson".

```
{customer.LAST NAME} = "Johnson"
```

FALSE, in all other situations.

Comments

This operator is often used in expressions with If-then-else operators. For example in Crystal syntax:

```
If {file.PURCHASES} = 0 Then

"Your account had no activity this month."

Else

"";
```

Which prints the "Your account..." statement if the {file.PURCHASES} field has a zero value, and prints nothing (indicated by the empty string "") if the {file.PURCHASES} field has something other than a zero value.

Related topics

Formula 9

Formula 15

Formula 17

Not Equal operator

Basic and Crystal syntax.

Usage

```
x<>y
```

x is not equal to y.

The Not Equal operator tells the program to evaluate an expression (x< >y) and return a TRUE (if x is not equal to y) or FALSE (if x is equal to y).

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
{orders.ORDER AMOUNT} <> 400
```

TRUE, where {orders.ORDER AMOUNT} is equal to 200 or {orders.ORDER AMOUNT} is equal to 401, etc.

```
{orders.ORDER AMOUNT} <> 400
```

FALSE, where {orders.ORDER AMOUNT} is equal to 400.

```
{file.DAY} <> "Thursday"
```

TRUE, when {file.DAY} = "Friday".

```
{file.DAY} <> "Thursday"
```

FALSE, when {file.DAY} = "Thursday".

```
{file.ONHAND} <>0
```

TRUE, where the value of {file.ONHAND} is 10 or .

```
{file.ONHAND} <>0
```

FALSE, where the value of {file.ONHAND} is zero.

```
{file.AVAILABLE} - {file.USED} <>10
```

TRUE, where the value of the {file.AVAILABLE} field less the value of the {file.USED} field gives a result other than 10.

```
{file.AVAILABLE} - {file.USED} <>10
```

FALSE, where it gives a value of 10.

Comments

This operator is often used with If-then-else operators. For example in Crystal syntax:

```
If {file.SEX} <> "M" Then

"FEMALE"

Else

"MALE"
```

Prints the word "FEMALE" if the value in the {file.SEX} field is not equal to "M", and prints the word "MALE" in all other situations.

Related topics

Formula 5

Formula 11

Formula 15

Less Than operator

Basic and Crystal syntax.

Usage

```
x < y
```

x is less than y

The less than operator tells the Formula Editor to evaluate an expression (x<y) and returns a TRUE (if x is less than y) or FALSE (if x is equal to or greater than y).

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
{file.WEIGHT} < 200
```

TRUE, where {file.WEIGHT} = 150, or {file.WEIGHT} = 199.

```
{file.WEIGHT} < 200
```

FALSE, where {file.WEIGHT} = 200 or {file.WEIGHT} = 400.

```
{file.COST} < {file.PRICE}
```

TRUE, where {file.COST}=350, {file.PRICE} = 400.

```
{file.COST} < {file.PRICE}
```

FALSE, where {file.COST} = 350 and {file.PRICE} = 350, or where {file.COST} = 350 and {file.PRICE} = 325.

Related topics

Formula 2

Formula 5

Formula 8

Formula 16

Greater Than operator

Basic and Crystal syntax.

Usage

```
x > y
```

x is greater than y.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
{file.WEIGHT} > 200
```

FALSE, where {file.WEIGHT}= 150, {file.WEIGHT} = 199, or {file.WEIGHT} = 200.

```
{file.WEIGHT} > 200
```

TRUE, where {file.WEIGHT} = 400 or {file.WEIGHT} = 201.

```
{file.COST} > {file.PRICE}
```

FALSE, where {file.COST =350}, {file.PRICE} = 400, or where {file.COST} = 350 and {file.PRICE} = 350.

```
{file.COST} > {file.PRICE}
```

TRUE, where {file.COST} = 350 and {file.PRICE} = 325.

Related topics

Formula 4

Formula 6

Formula 7

Formula 12

Less or equal operator

Basic and Crystal syntax.

Usage

```
x <= y
```

x is less than or equal to y.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
{file.WEIGHT} <= 200
```

TRUE, where {file.WEIGHT} = 150, {file.WEIGHT} = 200, or {file.WEIGHT} = 199.

```
{file.WEIGHT} <= 200
```

FALSE, where {file.WEIGHT} = 400.

```
{file.COST} <= {file.PRICE}
```

TRUE, where {file.COST} =350, {file.PRICE} = 400, or where {file.COST} = 350 and {file.PRICE} = 350.

```
{file.COST} <= {file.PRICE}
```

FALSE, where {file.COST} = 350 and {file.PRICE} = 325.

Related topics

Formula 14

Greater or Equal operator

Basic and Crystal syntax.

Usage

```
x >= y
```

x is greater than or equal to y.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
{file.WEIGHT} >= 200
```

FALSE, where {file.WEIGHT} = 150 or {file.WEIGHT} = 199.

```
{file.WEIGHT} >= 200
```

TRUE, where {file.WEIGHT} = 400, {file.WEIGHT} = 200, or {file.WEIGHT} = 201.

```
{file.COST} >= {file.PRICE}
```

FALSE, where {file.COST} =350, {file.PRICE} = 400.

```
{file.COST} >= {file.PRICE}
```

TRUE, where {file.COST} = 350 and {file.PRICE} = 325, or where {file.COST} = 350 and {file.PRICE} = 350.

Related topics

Formula 4

Formula 6

String operators

String operators are used to concatenate (join) text strings, to extract substrings from text strings, or to test for the presence of substrings in text strings.

Note: Operators are case-sensitive. For example, "ABC" is not equal to "abc". Nor is "abc" in the string "ABCDEF".

Concatenate operator (x +y)

Concatenate (x & y)

Subscript for arrays and strings

In String operator

Concatenate operator (x +y)

Basic and Crystal syntax.

Concatenate (x & y) and Concatenate (x + y) are synonymous if both operands x and y are Strings.

Usage

```
x + y
```

Concatenates (combines) string x to string y to make one contiguous string.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
"Bread" + " and " + "butter"
```

Returns "Bread and butter".

```
"Your customer number is " + ({customer.CUSTOMER ID}) + " and your
company contact person is " + ({customer.CONTACT FIRST NAME}) + "."
```

Returns "Your customer number is 64 and your company contact person is Bob." Where CustomerID="64" and First Name = "Bob".

Comments

- You can use this operator only if all the elements you are connecting are strings.
- If you want to include a value from a numeric field (for example, an account balance), you must first convert that value to a text string using the ToText function:

```
"Your account balance is " + ToText({file.BALANCE}) + "."
```

Related topics

Concatenate (x & y)

Formula 2

Formula 5

Formula 13

Formula 15

Concatenate (x & y)

Basic and Crystal syntax.

Concatenate (x & y) and Concatenate operator (x +y) are synonymous if both operands x and y are strings.

Usage

```
x & y
```

Concatenates the values of x and y, and returns the result as a String. x and y can be of one of the types Number, Boolean, Currency, Date, Time, DateTime, and String.

Examples

The following examples are applicable to both Crystal and Basic syntax:

```
10 & 20
```

Returns the String "10.0000020.00000", where the number formatting for this formula field specifies 5 decimal places. Use CStr if you want to specify how to format a String when converting from a different type. In this example, you can use CStr on the Number values a and b to convert and format each of them to a String, and then concatenate the resultant Strings.

```
True & False
```

Returns the String "TrueFalse".

```
"The meeting is " & #Nov 15. 1999#
```

Returns the String "The meeting is 11/15/99 12:00:00 AM".

```
"Boy" & "Girl"
```

Returns the String "BoyGirl".

Comments

The format of the returned value is determined by the format that has been set up for that formula field of the corresponding type of operands. However, if you want to convert to a String from a value of a different type and be able to specify formatting for the resultant string, you can use CStr.

Related topics

Concatenate operator (x +y)

Subscript for arrays and strings

Basic syntax (x(y))

Usage

```
x(y)
```

Extracts the y element from string x.

```
x(y To z)
```

Extracts the y to z range of elements from string x.

```
x(n)
```

Extracts the n element of array x.

Note: The subscript ranges are 1ased; they start at 1 rather than 0.

Subscript is used to extract one or more characters from a text string or to extract an element from an array.

Examples

The following examples are applicable to Basic syntax:

```
LNAME (1)
```

Returns "S", where LNAME = "Smith".

```
Postal (6)
```

Returns "V", where Postal Code = "T5A 9V2" (the space between A and 9 counts as an element).

```
{customer.POSTAL CODE} (5 to 7)
```

Returns "9V2", where Postal Code = T5A 9V2.

```
{file.ITEMNUMBER} (4 to 5)
```

Returns "40", where ItemNumber is A1/4020/B10.

Comments

- You must subscript an array by first declaring an array, then referencing the array variable name in Basic syntax. For example:

```
'declare array MyName

Dim MyName () As String

MyName = Array ("Company A", "Company B", "Company C")

'reference array by its name

formula = MyName (1)
```

- Do not confuse Subscript with the In String operator.

  While Subscript tests a target string for the presence of an element and extracts the element (if found) from the string, In String only tests the target string for the presence of the element.

Related topics

Crystal Syntax (x[y])

Crystal Syntax (x[y])

Usage

```
x[y]
```

Extracts the y element from string x.

```
x[y to z]
```

Extracts the y to z range of elements from string x.

```
x[n]
```

Extracts the n element of array x.

Note: The subscript ranges are 1ased; they start at 1 rather than 0.

Subscript is used to extract one or more characters from a text string or to extract an element from an array.

Examples

The following examples are applicable to Crystal syntax:

```
[100,233,466,998][3]
```

Returns 466; 466 is the third element in the array.

```
LNAME [1]
```

Returns "S", where LNAME = "Smith".

```
Postal [6]
```

Returns "V", where Postal Code = "T5A 9V2" (the space between A and 9 counts as an element).

```
{customer.POSTAL CODE} [5 to 7]
```

Returns "9V2", where Postal Code = "T5A 9V2".

```
{file.ITEMNUMBER} [4 to 5]
```

Returns "40", where ItemNumber is A1/4020/B10.

Comments

Do not confuse Subscript with In String operator.

While Subscript tests a target string for the presence of an element and extracts the element (if found) from the string, In String only tests the target string for the presence of the element.

Related topics

Formula 1

Formula 2

Formula 3

Formula 9

Formula 11

In String operator

Basic and Crystal syntax.

Usage

```
x in y
```

Tests for the presence of string x in string y.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
"Elm" in {customer.ADDRESS1}
```

TRUE, where {customer.ADDRESS1} is "1335 Elm Street".

```
"Elm" in {customer.ADDRESS1}
```

TRUE, where {customer.ADDRESS1} is "1335 Elmer Street".

```
"elm" in {file.MOTTO}
```

FALSE, where {file.MOTTO} = "Feel more energy".

(The "el" ending "feel" and the "m" beginning the word "more" are separated by a space which itself counts as an element.)

```
"el m" in {file.MOTTO}
```

TRUE, where {file.MOTTO} = "Feel more energy".

The search string "el m" this time contains the space between the l and the m, which allows for a perfect match.

Note: The "in" operator can also be used to test for the presence of a string in a text range. For example, "V5B" in "V0A" to "V9Z".

Range operators

Range operators are used to create ranges and to see if a value is within the range created. These operators test for consecutive values such as dates, text, or amounts which fall within a range.

In Range operator

Make Range operator (x To y)

Left end point excluded range (x _To y)

Right end point excluded range (x To_ y)

Both end points excluded range (x _To_ y)

Is > x

Is < x

Is >= x

Is <= x

Up from (upFrom x)

Up from but not including (upFrom_ x)

Up to (upTo x)

Up to but not including (upTo_ x)

In Range operator

Basic and Crystal syntax.

Usage

```
x in y
```

Tests a range of value (y) to see if a value (x) falls within the range specified.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
CurrentDate in CDate(1990, 09, 01) to CDate(1990, 09, 20)
```

Returns True, if today's date is September 15, 1990.

```
CurrentDate in CDate(1990, 09, 01) to CDate(1990, 09, 20)
```

Returns False, if today's date is September 21, 1990.

```
{file.QTY} in {file.ONHAND} to ({file.BACKORDER} + {file.ONORDER})
```

Returns True, where {file.QTY} = 20, {file.ONHAND} = 10, {file.BACKORDER} = 5, {file.ONORDER} = 25 (Is 20 in the range that begins with 10 and ends with the sum of 5 and 25?).

```
{file.QTY} in {file.ONHAND} to ({file.BACKORDER} + {file.ONORDER})
```

Returns False, where {file.QTY} = 31, {file.ONHAND} = 10, {file.BACKORDER} = 5, {file.ONORDER} = 25 (Is 31 in the range that begins with 10 and ends with the sum of 5 and 25?).

Comments

The combination of Make Range operator (x To y) and In Range operators is often used with the If-then-else operators operator. For example the Crystal syntax formula:

```
If ({file.AMOUNT} in (100.00 to 250.00)) Then
```

```
(.10 * {file.AMOUNT})
```

```
Else
```

```
0;
```

If the value of {file.AMOUNT} falls within the range 100.00 to 250.00, multiply .10 times {file.AMOUNT}. If it does not, it returns zero.

Related topics

Formula 9

Formula 10

DateTime function

Make Range operator (x To y)

Basic and Crystal syntax.

Usage

```
x to y
```

Create the range x to y.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
100.00 to 250.00
```

The range of consecutive numeric values beginning with 100.00 and ending with 250.00, including the end values.

```
CDate(1990, 09, 01) to CDate(1990, 09, 20)
```

The range of consecutive dates beginning with September 1, 1990, and ending with September 20, 1990. Both September 1 and September 20 are included in the range.

```
"Aaron" to "Lusk"
```

The range of consecutive text values beginning with "Aaron" and ending with "Lusk", including the beginning and end values.

Comments

- You cannot create a formula that has a range as a result. Thus, Make Range is always used in conjunction with other operators such as the In Range operator. The combination of Make Range and In Range produces a formula that gives a single TRUE or FALSE value, not a range.
- The program comes with a number of Date Range functions, such as YearToDate, preset for your convenience.

Related topics

Formula 2

Formula 3

Formula 9

Formula 10

Left end point excluded range (x _To y)

Basic and Crystal syntax.

Usage

```
x _To y
```

x _To y is used to specify a range of values greater than but not including the value x, and less than or equal to the value y. x and y can be of one of the types: Number, Boolean, Currency, Date, Time, DateTime, and String.

Examples

The following example is applicable to Basic syntax:

```
Dim BlackoutDates As Date Range

Rem Blackout period is Jan. 1, 1999 to March 31, 1999, inclusive

BlackoutDates = CDate(#12/31/1998#) _To CDate(#3/ 31/1999#)

If CurrentDate In BlackoutDates Then

formula = "Blackout"

Else

formula = "Free to trade"

End If
```

If today's date is Dec. 31, 1998, then "Free to trade" is returned since it is before the Blackout period.

If today's date is Jan. 1, 1999, then "Blackout" is returned since it is within the Blackout period.

If today's date is March 31, 1999, then "Blackout" is returned since it is within the Blackout period.

If today's date is Oct. 30, 1999, then "Free to trade" is returned since it is after the Blackout period.

Right end point excluded range (x To_ y)

Basic and Crystal syntax.

Usage

```
x To_ y
```

x To_ y is used to specify a range of values greater than or equal to the value x, and less than but not including the value y. x and y can be of one of the types: Number, Boolean, Currency, Date, Time, DateTime, and String.

Examples

The following example is applicable to Basic syntax:

```
Dim BlackoutDates As Date Range

Rem Blackout period is Jan. 1, 1999 to March 31, 1999, inclusive

BlackoutDates = CDate(#1/1/1999#) To_ CDate(#4/1/1999#)

If CurrentDate In BlackoutDates Then

formula = "Blackout"

Else

formula = "Free to trade"

End If
```

If today's date is Dec. 31, 1998, then "Free to trade" is returned since it is before the Blackout period.

If today's date is Jan. 1, 1999, then "Blackout" is returned since it is within the Blackout period.

If today's date is March 31, 1999, then "Blackout" is returned since it is within the Blackout period.

If today's date is April 1, 1999, then "Free to trade" is returned since it is after the Blackout period.

Both end points excluded range (x _To_ y)

Basic and Crystal syntax.

Usage

```
x _To_ y
```

x _To_ y is used to specify a range of values greater than but not including the value x, and less than but not including the value y. x and y can be of one of the types: Number, Boolean, Currency, Date, Time, DateTime, and String.

Examples

The following example is applicable to Basic syntax:

```
Dim BlackoutDates As Date Range

Rem Blackout period is Jan. 1, 1999 to March 31, 1999, inclusive

BlackoutDates = CDate(#12/31/1998#) _To_ CDate(#4/1/1999#)

If CurrentDate In BlackoutDates Then

formula = "Blackout"

Else
```

```
formula = "Free to trade"
End If
```

If today's date is Dec. 31, 1998, then "Free to trade" is returned since it is before the Blackout period.

If today's date is Jan. 1, 1999, then "Blackout" is returned since it is within the Blackout period.

If today's date is March 31, 1999, then "Blackout" is returned since it is within the Blackout period.

If today's date is April 1, 1999, then "Free to trade" is returned since it is after the Blackout period.

Is > x

Basic and Crystal syntax.

Is > x is preferred in Basic syntax, whereas Up from but not including (upFrom_ x) is preferred in Crystal syntax.

Usage

```
Is > x
```

Is > x is used to specify a range of values greater than the value x. x can be of any type: Number, Boolean, Currency, Date, Time, DateTime, and String.

Examples

The following examples are applicable to Basic and Crystal syntax:

```
Is > 250
```

The range of all numbers greater than 250.

```
Is > CDate (2000, 3, 17)
```

The range of dates from March 18, 2000 onwards, including March 18, 2000.

Is < x

Basic and Crystal syntax.

Is < x is preferred in Basic syntax, whereas Up to but not including (upTo_ x) is preferred in Crystal syntax.

Usage

```
Is < x
```

Is < x is used to specify a range of values less than the value x. x can be of any type: Number, Boolean, Currency, Date, Time, DateTime, and String.

Examples

The following example is applicable to Basic syntax:

```
Dim BlackoutDates() As Date Range

Rem Blackout period is any day before 1999 or any date within March
1999

BlackoutDates = Array (Is < CDate(1999, 1, 1), _

CDate(1999, 3, 1) To CDate (1999, 3, 31))

If CurrentDate In BlackoutDates Then

formula = "Blackout"

Else

formula = "Free to trade"

End If
```

If today's date is Dec. 30, 1998, then "Blackout" is returned since it is within the Blackout period before 1999.

If today's date is Jan. 1, 1999, then "Free to trade" is returned since it is not within any blackout period.

If today's date is March 31, 1999, then "Blackout" is returned since it is within the Blackout month of March.

If today's date is Oct. 30, 1999, then "Free to trade" is returned since it is not within any blackout period.

Up from (upFrom x)

Basic and Crystal syntax.

upFrom x is preferred in Crystal syntax, whereas Is >= x is preferred in Basic syntax.

Usage

```
upFrom x
```

upFrom x is used to specify a range of values greater than or equal to the value x. x can be of any type: Number, Boolean, Currency, Date, Time, DateTime, and String.

Examples

The following examples are applicable to Basic and Crystal syntax:

```
upFrom 250
```

The range of all numbers greater than or equal to 250.

```
upFrom CDate (2000, 3, 17)
```

The range of dates from March 17, 2000 onwards including March 17, 2000.

Up from but not including (upFrom_ x)

Basic and Crystal syntax.

UpFrom_ x is preferred in Crystal syntax, whereas Is > x is preferred in Basic syntax.

Usage

```
upFrom_ x
```

upFrom_ x is used to specify a range of values greater than but not including the value x. x can be of any type: Number, Boolean, Currency, Date, Time, DateTime, and String.

Examples

The following examples are applicable to Basic and Crystal syntax:

```
upFrom_ 250
```

The range of all numbers greater than 250.

```
upFrom_ CDate (2000, 3, 17)
```

The range of dates from March 18, 2000 onwards, including March 18, 2000.

Up to (upTo x)

Basic and Crystal syntax.

upTo x is preferred in Crystal syntax, whereas Is <= x is preferred in Basic syntax.

Usage

```
upTo x
```

upTo x is used to specify a range of values less than or equal to the value x. x can be of any type: Number, Boolean, Currency, Date, Time, DateTime, and String.

Examples

The following examples are applicable to Basic and Crystal syntax:

```
upTo 250
```

The range of all numbers less than or equal to 250.

```
upTo CDate (2000, 3, 17)
```

The range of dates before March 17, 2000, up to and including March 17, 2000.

Up to but not including (upTo_ x)

Basic and Crystal syntax.

upTo_ x is preferred in Crystal syntax, whereas Is < x is preferred in Basic syntax.

Usage

```
upTo_ x
```

upTo_ x is used to specify a range of values less than but not including the value x. x can be of any type: Number, Boolean, Currency, Date, Time, DateTime, and String.

Examples

The following examples are applicable to Basic and Crystal syntax:

```
upTo_ 250
```

The range of all numbers less than 250.

```
upTo_ CDate (2000, 3, 17)
```

The range of dates before March 17, 2000, not including March 17, 2000.

Boolean operators

Boolean operators are used to create conditions that require a logical relationship between two or more values. Conditions that use Boolean operators are called Boolean expressions.

- A and B means that both A and B must be true for the condition to be satisfied (to return a TRUE value)
- A or B means that either A or B (or both) must be true for the condition to be satisfied (to return a TRUE value)

Not operator

And operator

Or operator

Xor (Logical exclusion)

Eqv (Logical equivalence)

Imp (Logical implication)

Not operator

Basic and Crystal syntax.

Usage

```
not (x)
```

Reverses the True or False value of x.

Value of x	Not x
True	False
False	True

- Not (Not(False)) = False
- Not (Not(True)) = True

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
not (A>B and B>C)
```

If A=5, B = 4, C = 3, the expression (A>B and B>C) is TRUE. Both conditions tied together by the Boolean operator And are TRUE; thus, the entire statement has a value of TRUE. The Not operator changes the value of the expression to FALSE.

```
not (A>B and B>C)
```

If A=3, B = 4, C = 3, the expression (A>B and B>C) is FALSE. One of the two conditions tied together by the Boolean operator And is FALSE; thus, the entire statement has a value of FALSE. The Not operator changes the value of the expression to TRUE.

```
not ({file.ONHAND} - {file.ORDER} > 0)
```

Returns TRUE if {file.ONHAND} = 10 and {file.ORDER} = 11.

```
not ({file.ONHAND} - {file.ORDER} > 0)
```

Returns FALSE if {file.ONHAND} = 10 and {file.ORDER} = 9.

Related topics

Formula 10

And operator

Basic and Crystal syntax.

Usage

```
x And y
```

Value of x	Value of y	x And y
True	True	True
True	False	False
False	True	False
False	False	False

Examples

The following example is applicable to Crystal syntax:

```
If {file.CREDIT LIMIT} = 5000 and {file.SALESMAN} = "SP" Then
```

```
{file.AMOUNT}
```

```
Else
```

```
0
```

If the credit limit is 5000 and the salesman is SP (both conditions true), then returns the value in the {file.AMOUNT} field, otherwise return zero.

```
A > B and B > C
```

Returns TRUE, where A = 10, B = 6, and C = 3 (both conditions are true).

Returns FALSE, where A=10, B=6, and C=7 (only one of the two conditions are true).

```
(A>B) and (A * C - D > E) and (E / D <= B)
```

Returns TRUE, where A = 7, B = 5, C = 3, D = 2, E = 10 (all three of the conditions are true).

Related topics

Order of precedence (Basic syntax)

Order of precedence (Crystal syntax)

Formula 6

Formula 15

Or operator

Basic and Crystal syntax.

Usage

```
x Or y
```

Value of x	Value of y	x Or y
True	True	True
True	False	True
False	True	True
False	False	False

Examples

The following example is applicable to Crystal syntax:

```
If {file.CREDIT LIMIT} = 5000 or {file.SALESMAN} = "SP"
```

```
Then
```

```
{file.AMOUNT}
```

```
Else
```

```
0;
```

This means that if the credit limit is 5000, or, if the salesman is SP, (either of the conditions are true), then it returns the value in the {file.AMOUNT} field; otherwise it returns nothing.

The following examples are applicable to both Basic and Crystal syntax:

```
A > B or B > C
```

TRUE, where A = 10, B = 6, and C = 3 (both conditions true).

```
A > B or B > C
```

TRUE, where A=10, B=6, and C=7 (either one of the two conditions true).

```
A > B or B > C
```

FALSE, where A=5, B=6, and C=7 (neither of the two conditions true).

```
(A>B) or (A * C - D > E) or (E / D<= B)
```

TRUE, where A = 5, B = 5, C = 3, D = 2, E = 12 (At least one of the three conditions is true. In this case only (A * C - D > E) is true).

Related topics

Formula 6

Xor (Logical exclusion)

Basic and Crystal syntax.

Usage

```
x Xor y
```

Value of x	**Value of y**	**x Xor y**
True	True	False
True	False	True
False	True	True
False	False	False

Examples

The following example is applicable to both Basic and Crystal syntax:

```
{Customer.SELL} Xor {Customer.GOVT_APPROVE}
```

Returns True if one of {Customer.SELL} and {Customer.GOVT_APPROVE} is True, and the other is False.

Eqv (Logical equivalence)

Basic and Crystal syntax.

Usage

```
x Eqv y
```

Value of x	Value of y	X Eqv y
True	True	True
True	False	False
False	True	False
False	False	True

Examples

The following example is applicable to Basic and Crystal syntax:

```
{Customer.SELL} Eqv {Customer.GOVT_APPROVE}
```

Returns True if both {Customer.SELL} and {Customer.GOVT_APPROVE} are True, or both {Customer.SELLS} and {Customer.GOVT_APPROVE} are False.

Imp (Logical implication)

Basic and Crystal syntax.

Usage

```
x Imp y
```

Value of x	Value of y	X Imp y
True	True	True
True	False	False
False	True	True
False	False	True

Examples

The following example is applicable to both Basic and Crystal syntax:

```
{Customer.SELL} Imp {Customer.GOVT_APPROVE}
```

Returns True if both {Customer.SELL} and {Customer.GOVT_APPROVE} are in agreement (that is, both are True or both are False), or only {Customer.GOVT_APPROVE} agrees (that is, only {Customer.GOVT_APPROVE} is True) and overrides.

Array operators

Array operators are used to build a list of data fields, constants, or text string. These lists can then be used for checking to see if a field exists in the list, or for extracted elements by their position. Whereas the Range operators are used to see if an item exists in a range of values, these array operators allow you to see if an item exists in a set of non-contiguous values.

Make Array operator

In Array operator

Redim

Redim Preserve

Make Array operator

Crystal syntax.

The Make Array operator can be used with Crystal syntax only; the Array function can be used with Basic syntax only. See Arrays (Basic Syntax).

Usage

```
[x,y,z,...n]
```

Build an array containing the elements x, y, z,...n.

Examples

The following examples are applicable to Crystal syntax:

```
[100,200,300,400]

[{file.QTYA}, {file.QTYB}, {file.QTYC}]

[({file.AMT1} * .5),({file.AMT2} * .5), ({file.AMT3} * .25)]

[500, ({file.QTY} / 3)]
```

Note: You cannot have more than one data type in an array.

In Array operator

Basic and Crystal syntax.

Usage

```
x in [y]
```

Is x in the array y?

Examples

The following example is applicable to both Basic and Crystal syntax:

```
Rem Basic syntax

{customer.REGION} In Array ("CA", "HI", "AK")
```

```
//Crystal syntax

{customer.REGION} in ["CA", "HI", "AK"]
```

Is the value of the {customer.REGION} field in the array of state abbreviations?

The following examples are applicable to Crystal syntax:

```
{file.COLOR} in ["Red", "White", "Blue"]
```

Is the value of the {file.COLOR} field in the array of colors listed in the brackets?

```
DayofWeek({file.DATE}) in [2, 4, 6]
```

Is the value of the {file.DATE} field, converted to a number that represents the DayOfWeek in the array of numbers listed in the brackets? (Sunday = 1, Saturday = 7)

Related topics

Formula 2

Formula 3

Formula 8

Formula 11

Redim

Basic syntax x(n)

Use Redim x[n] (with the square brackets) only in Crystal syntax, and Redim x(n) (with the round brackets) only in Basic syntax.

Usage

```
Redim x(n)
```

Re-dimension the array x to size n, where x is an array and n is a positive whole number specifying the new size of x.

Examples

The following example is applicable to Basic syntax:

```
Dim x() As String

'Initialize first three array elements

x = Array ("a", "bb", "ccc")

'Re-size x to 4; old values are ignored

'x is now equal to Array ("", "", "", "")

Redim x(4)

'x is now equal to Array ("", "", "", "dddd")

x(4) = "dddd"

formula = x(4)
```

Comments

When an array is re-dimensioned with Redim, elements in the array are filled with default values for that type. See Default values for the simple types (Basic syntax).

Related topics

Array data types (Basic syntax)

Using array variables (Basic syntax)

Crystal syntax x[n]

- Use Redim x[n] (with the square brackets) only in Crystal syntax, and Redim x(n) (with the round brackets) only in Basic syntax.
- Unlike Redim x(n) in Basic syntax, a Redim statement in Crystal syntax cannot have multiple arrays.

Usage

```
Redim x[n]
```

Re-dimension the array x to size n, where x is an array and n is a positive whole number specifying the new size of n.

Examples

The following example is applicable to Crystal syntax:

```
Local StringVar array x:= ["a", "bb", "ccc"];

// resize the array to size 4; old values are ignored

// and filled with default values of empty strings

Redim x [4];
```

x [4] := "dddd"; // only x[4] is initialized

Comments

When an array is re-dimensioned with Redim, elements in the array are filled with default values for that type. See Default values for the simple types (Crystal syntax).

Related topics

Array data types (Crystal syntax)

Using array variables (Crystal syntax)

Redim Preserve

Basic syntax x(n)

Usage

```
Redim Preserve x(n)
```

Re-dimension the array x to size n, while preserving the initial values in x. x is an array and n is a positive whole number specifying the new size of x.

Examples

The following example is applicable to Basic syntax:

```
Dim x() As String

'Initialize first three array elements

x = Array ("a", "bb", "ccc")

'Re-size x to 4; old values are retained

Redim Preserve x(4)

'initialize x(4)

x(4) = "dddd"

'return "dddd"

formula = x(4)
```

Related topics

Array data types (Basic syntax)

Using array variables (Basic syntax)

Crystal syntax x[n]

- Use Redim Preserve x[n] (with the square brackets) only in Crystal syntax, and Redim Preserve x(n) (with the round brackets) only in Basic syntax.
- Unlike Redim Preserve x(n) in Basic syntax, you may not have multiple arrays in one Redim Preserve statement in Crystal syntax.

Usage

```
Redim Preserve x[n]
```

Re-dimension the array x to size n, while preserving the initial values in x. x is an array and n is a positive whole number specifying the new size of n.

Examples

The following example is applicable to Crystal syntax:

```
Local StringVar array x := ["a", "bb", "ccc"];

// resize the array to size 4 while preserving the old values.

Redim Preserve x [4];

// now x = ["a", "bb", "ccc", "dddd"]

x [4] := "dddd";

x[4] // returns "dddd"
```

Related topics

[Array data types (Crystal syntax)]

[Using array variables (Crystal syntax)]

Pattern operators

Pattern operators allow you to compare strings to a given pattern. They are useful operators for selecting records based on part of a string (Like pattern operator) or based on a partly unknown string (Starts with operator).

Starts with operator

Like pattern operator

Starts with operator

Basic and Crystal syntax.

The startsWith operator is useful for selecting records to include or exclude from your report.

Usage

```
x startsWith y
```

```
{fieldname} startsWith "abc"
```

This operator tests to see if the contents of {fieldname} start with a character string that you specify: "abc". If the contents of the field do start with "abc", then the formula returns the value True. If the field starts with anything else, the formula returns False.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
{customer.CUSTOMER NAME} startsWith "A"
```

TRUE, where {customer.CUSTOMER NAME} = ABC Ltd.

```
{customer.CUSTOMER NAME} startsWith "C"
```

FALSE, where {customer.CUSTOMER NAME} = ABC Ltd.

```
{customer.CUSTOMER NAME} startsWith "ABC"
```

TRUE, where {customer.CUSTOMER NAME} = ABC Inc. or ABC Ltd.

```
{customer.CUSTOMER NAME} startsWith "ABC"
```

FALSE, where {customer.CUSTOMER NAME} = XYZ Inc.

Like pattern operator

Basic and Crystal syntax.

The Like operator is useful for selecting records to include or exclude from your report.

Usage

```
x like y
```

```
{fieldname} like "cn*"
```

This operator tests to see if the contents of {fieldname} matches a pattern that you specify in a character string "c?n*". If the contents of the field do fit the pattern "c?n*", then the formula returns the value True. If the field starts with anything else, the formula returns False.

Use the wildcard symbols **?** and * to stand for variable characters. The **?** stands for a single character. The * symbol stands for any number of characters.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
{customer.FIRST NAME} like "Dn"
```

TRUE, where {customer.FIRST NAME} = Dan or Don.

```
{customer.FIRST NAME} like "Dn"
```

FALSE, where {customer.FIRST NAME} = Doug or Rob.

```
{customer.LAST NAME} like "*sn*"
```

TRUE, where {customer.LAST NAME} = Johnson or Olson or Olsen.

```
{customer.LAST NAME} like "*sn*"
```

FALSE, where {customer.LAST NAME} = Johnston or Smith.

Control structure operators

Control structures control the flow of logic in a formula. You can use them to build formula setting conditions that, if met, trigger specific consequences or repeat a sequence of actions under certain conditions.

If-then-else operators

Related topics

Basic syntax

If statements (Basic syntax)

Select statements (Basic syntax)

If-Then-Else operators

Basic Syntax

Usage

There are 2 kinds of If statements, the multi-line If, and the single-line If. Generally, it is better to use the multi-line If since it results in clearer formulas.

Multi-line If statements

```
If x Then
statementsA
End If
```

If x is True then evaluate stamentsA.

```
If x Then
statementsA
Else
statementB
End If
```

If x is True then evaluate statementsA otherwise evaluate statementsB.

```
If x Then
statementsA
ElseIf y Then
statementsB
Else
statementsC
End If
```

If x is True then evaluate statementsA; otherwise if y is True, evaluate statementsB; otherwise evaluate statementsC.

Note:

- There can be 0 or more ElseIf clauses and either 0 or 1 Else clause.
- The Else clause must be last.
- ElseIf is one word.
- Multi-line If statements must end in an End If.

Single-line If statements

```
If x Then y Else z
```

If x is true then do y. If x is not true (Else), do z.

Examples

The following example is applicable to Basic syntax:

```
If {Employee.Dept} = "Sales" Then

formula = {Employee.Salary} * 0.06

Else

formula = {Employee.Salary} * 0.04

End If
```

Assigns rate of salary increase based on an employee's department

Comments

The If part of the expression can include text, numbers, (Cust#<"10000"), and formulas ({@Formula}), where @Formula is Boolean.

Related topics

For full details on how to use the If statement, see:

If statements (Basic syntax)

Single-line and multi-line If statements (Basic syntax)

Crystal Syntax

Usage

```
If x Then y Else z
```

If x is true then do y. If x is not true (Else), do z.

Examples

The following example is applicable to Crystal syntax:

```
If {customer.POSTAL CODE} <= "49999" Then

"Blue Label"

Else
```

```
"Ground"
```

Assigns method of shipping based on distance from ship point.

```
If ToNumber ({file.ITEM}) >= 2500 and ToNumber({file.ITEM}) < 2600
Then

"Seasonal"

Else

""
```

If statement includes an **and** operator for the ToNumber (numeric), ToNumber (string), ToNumber (Boolean) function.

```
If {file.COUNT} >= 25 Then

{file.DISTRIBUTOR} * {file.COUNT}

Else

{file.DEALER} * {file.COUNT}
```

Quantity ordered determines price list used.

```
If {file.ONHAND}>10 Then

{file.ORDERED}

Else

If {file.ORDERED} < 5 Then

{file.ORDERED}

Else

2
```

Allocation based on quantity ordered using If-then-else nesting.

```
If PageNumber = 1 Then

PrintDate

Else

CDate(0,0,0)
```

Prints the print date (from the PrintDate function) on the first page, and prints nothing [as designated by the empty date CDate(0,0,0)] on the remaining pages.

Comments

- The If part of the expression can include text, numbers, (Cust#<"10000"), and formulas ({@Formula}), where @Formula is Boolean.
- The Then and the Else parts, however, must both be of the same data type: (Then text, Else text; Then number, Else number). Mixing text and number actions will result in an error message.

Related topics

For full details on how to use the If statement, see:

If expressions (Crystal syntax)

More details on If expressions (Crystal syntax)

Most formula examples use this operator. Select the formula of interest from the Formulas in action index.

Other operators

Other operators are used to assign values to variables, indicate an order of precedence in which calculations are to be performed, separate comments from formulas, define date constants, or call a function.

Parentheses operators

Assignment operators

Comments

Date-time literal (#...#)

Parentheses operators

Basic and Crystal syntax.

Usage

```
(x + y) * z
```

Performs the calculations inside the parentheses first.

Parentheses are used to control the order of precedence in which the Format Editor calculates a formula. In Basic syntax, they are also used as a string and array subscript.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
8 + 6 * 3 - 6 / 2
```

Returns 23.

```
(8 + 6) * 3 - 6 / 2
```

Returns 39.

```
(8 + 6) * (3 - 6 / 2)
```

Returns 0.

```
(8 + 6 * 3 - 6) / 2
```

Returns 10.

```
{file.SALES} - {file.COGS} - {file.T&E} * .8
```

Returns 1000, where SALES=25,000, COGS=12,000, and T&E=15,000.

```
{file.SALES} - (({file.COGS} - {file.T&E}) * .8)
```

Returns 27,400, where SALES=25,000, COGS=12,000, and T&E=15,000.

```
({file.SALES} - {file.COGS} - {file.T&E}) * .8
```

Returns ,600, where SALES=25,000, COGS=12,000, and T&E=15,000.

Related topics

Most formula examples use this operator. Select the formula of interest from the Formulas in action index.

Date-time literal (#...#)

Basic and Crystal syntax.

Usage

```
# alpha-numeric string #
```

This operator converts the enclosed alpha-numeric string to a DateTime value. The alpha-numeric should contain a date, a time, or a date and time representation that can be commonly recognized. Many formats are supported.

Examples

The following examples are applicable to both Basic and Crystal syntax:

```
#10:10am#
```

Returns a DateTime value of 10:10:00am.

```
#Oct. 19, 1999 #
```

Returns a DateTime value of October 19, 1999 12:00:00am.

```
#Oct. 19, 1999 10:10am#
```

Returns a DateTime value of October 19, 1999 10:10:00am.

Comments

If a formula returns a DateTime value, the format of the returned DateTime value is determined by the DateTime formatting applicable to that formula field.

Related topics

Date, Time, and DateTime (Basic syntax) under Simple data types (Basic syntax)

Date, Time, and DateTime (Crystal syntax) under Simple data types (Crystal syntax)

Assignment operators

Basic syntax (x = y)

You can use the assignment operator (=) and the assignment keyword "Let" (Let x=y) with only Basic syntax. Crystal syntax uses the other assignment operator (:=).

Usage

```
x = y
```

Assigns y to the variable x. x must have already been declared before the assignment. y must be of the same type as x, or of a type compatible with x.

```
Let x = y
```

Assigns y to the variable x. x must have already been declared before the assignment. y must be of the same type as x, or of a type compatible with x.

Examples

The following example is applicable to Basic syntax:

```
Dim n As Number

Dim b As Boolean

Dim c As Currency

Dim d As Date

Dim t As Time

Dim s As String

Dim numArray () As Number

'n is assigned 100

n = 100

'b is assigned the Boolean value True

b = True

'c is assigned the Currency value 5000

c = CCur (5000)

'd is assigned the Date value of Oct. 10, 1999

d = CDate ("10, 10, 1999")

't is assigned the Time value 12:50am

t = CTime ("12:50am")
```

' s is assigned a string which is a concatenation of 2 substrings and a field value

s = "The total sales is " & {Orders.Order Amount} & " billion."

' numArray is assigned an array of Number values

numArray = Array (10, 20, 30)

formula = s

Comments

- A variable must be declared in a separate statement before it is assigned any value in Basic syntax.
- Assignment only assigns a value to a variable. When a formula returns, the value assigned to the variable Formula is then formatted according to the format specified for that formula field of the corresponding type.

Crystal syntax (x := y)

You can use the assignment operator (:=) with only Crystal syntax. For more details, see Variable declarations (Crystal syntax).

Usage

```
x := n
```

Assigns the value n to the variable x. (x must have already been declared in the same formula.)

Examples

The following examples are applicable to Crystal syntax:

```
Amount:= 0
```

Initializes (sets to zero) the variable named Amount.

```
Amount:= 100
```

Assigns the value 100 to the variable named Amount.

```
Amount:= Amount + {file.QTY}
```

Assigns the result of a calculation to the variable named Amount. The calculation adds the value of the quantity field ({file.QTY}) to the current value of the Amount variable.

```
Amount:= {file.QTY1} + {file.QTY2} + {file.QTY3}
```

Totals the three quantity fields and assigns the total to the variable named Amount.

```
Customer:= "Westside Motors"
```

Assigns the string "Westside Motors" to the variable named Customer.

```
Customer:= {customer.FIRST NAME} + {customer.LAST NAME}
```

concatenates two fields and assigns the concatenated value of both fields to the variable named Customer.

```
Customer:= TrimRight({customer.FIRST NAME}) + {customer.LAST NAME}
```

Trims the trailing blanks from the first name field ({customer.FIRST NAME}), concatenates that field to the last name field ({customer.LAST NAME}), and assigns the concatenated value of both fields to the variable named Customer.

```
Customer:= "Mr. " + {customer.LAST NAME}
```

Concatenates the string "Mr. " with the value of the last name field {customer.LAST NAME} and assigns the concatenated value to the variable named Customer.

```
Amount:= 100; Customer:= "Westside Motors"
```

Assigns the constant 100 to the number variable named Amount and assigns the string "Westside Motors" to the string variable named Customer. You can assign values to multiple variables by separating the assignment statements with semicolons.

Comments

Formula comments are notes that are included with a formula to explain its design and operation. Comments do not print, and they do not affect the formula; they appear only in the Formula Editor.

Basic syntax (Rem)

Usage

```
Rem text
```

The text that follows the operator is a comment.

Examples

The following examples are applicable to Basic syntax:

```
Rem This formula returns the total number of days worked in a year.
```

Comment is on a new line on its own.

```
formula = TotalDays : Rem Total number of days worked in a year.
```

Comment follows a formula statement and a colon.

```
formula = TotalDays : Rem This formula returns the total number of
days _worked in a year.
```

Comment follows a formula statement and a colon, and it uses the line continuation character to continue onto the next line.

Comments

- The comment can be above or below the formula. It can begin on a new line on its own, or it can follow the formula on the same line if it is preceded by a colon (:).
- Can use a line continuation character ("_") within comment to carry on to next line.
- You can use the comment operator (//) only with Crystal syntax, and the other comment operators (') and (Rem) with Basic syntax.

Related topics

Comments (Basic syntax)

Basic syntax (')

Usage

```
' text
```

The text that follows the operator is a comment.

Examples

The following examples are applicable to Basic syntax:

```
' This formula returns the total number of days worked in a year.
```

Comment is on a new line on its own.

```
formula = TotalDays 'Total number of days worked in a year.
```

Comment follows a formula statement.

```
formula = TotalDays 'This formula returns the total number of days _
worked in a year.
```

Comment follows a formula statement, and it uses the line continuation character to continue onto the next line.

Comments

- The comment can be above or below the formula. It can begin on a new line on its own, or it can follow the formula on the same line if it is preceded by the apostrophe (').
- Can use a line continuation character ("_") within comment to carry on to next line.
- You can use the comment operator (//) only with Crystal syntax, and the other comment operators (') and (Rem) with Basic syntax.

Related topics

Comments (Basic syntax)

Crystal syntax (//)

Usage

```
// text
```

The text that follows the operator is a comment; it is not treated as part of a formula, and it does not print.

Examples

The following examples are applicable to Crystal syntax:

{file.SALES} - {file.COST}

// calculates the gross profit

{file.SALES} - {file.COST} is the formula; the Formula Editor ignores everything else.

({file.SALES} - {file.QUOTA}) *.06

// calculates sales commission

({file.SALES} - {file.QUOTA}) * .06 is the formula; the Formula Editor ignores everything else.

//what follows is

//a formula. Note that when

//we force the line break we have

//to begin each new line with a

//comment operator.

If {file.COST} > {file.SELL PRICE} Then "Loss" Else "";

//That was a formula

All commented lines that appear before or after the formula are ignored.

Comments

- The comment can be above or below the formula, or it can follow the formula on the same line if it is preceded by two slashes.

- You can use the comment operator (//) with only Crystal syntax, and the other comment operators (') and (Rem) with only Basic syntax.

Related topics

Comments (Crystal syntax)

Variable declaration operators

All variable names in the formula language must be declared before you can use them in a formula.

Variable declarators for Basic syntax

Variable declarators for Crystal syntax

Variable declarators for Basic syntax

Variable declarators let you define a variable by giving the variable a name. You can also assign a value to a variable when you declare it by using the Assignment operator. The following are variable declarators in Basic syntax.

Usage

- Dim x
- Dim x ()
- Dim x As Boolean
- Dim x As Number
- Dim x As Currency
- Dim x As Date
- Dim x As Time
- Dim x As DateTime
- Dim x As String
- Dim x As Number Range
- Dim x As Currency Range
- Dim x As Date Range
- Dim x As Time Range
- Dim x As DateTime Range
- Dim x As String Range
- Dim x () As Boolean
- Dim x () As Number
- Dim x () As Currency
- Dim x () As Date
- Dim x () As Time
- Dim x () As DateTime
- Dim x () As String

- Dim x () As Number Range
- Dim x () As Currency Range
- Dim x () As Date Range
- Dim x () As Time Range
- Dim x () As DateTime Range
- Dim x () As String Range.

Declares a variable x that can hold data of a type corresponding to the variable declarator used.

Comments

- A variable declarator must be used to declare a variable before the variable can be used in a formula.
- Dim specifies a variable with local scope. You can use the keywords Local, Global, and Shared instead of Dim to specify different scopes. Local and Dim are equivalent keywords. See Variable Scope (Basic syntax) and Variable scopes for Basic syntax.

Related topics

Variable declarations using Dim (Basic syntax)

Declaring array variables (Basic syntax)

Variable declarators for Crystal syntax

Variable declarators let you define a variable by giving the variable a name. Optionally, you can also assign a value to a variable when you declare it by using the Assignment operator. The following are variable declarators in Crystal syntax.

Usage

- Local BooleanVar x
- Local NumberVar x
- Local CurrencyVar x
- Local DateVar x
- Local TimeVar x
- Local DateTimeVar x
- Local StringVar x
- Local NumberVar range x
- Local CurrencyVar range x
- Local DateVar range x
- Local TimeVar range x
- Local DateTimeVar range x
- Local StringVar range x
- Local BooleanVar array x
- Local NumberVar array x
- Local CurrencyVar array x

- Local DateVar array x
- Local TimeVar array x
- Local DateTimeVar array x Local StringVar array x
- Local NumberVar range array x
- Local CurrencyVar range array x
- Local DateVar range array x
- Local TimeVar range array x
- Local DateTimeVar range array x
- Local StringVar range array x.

Declares a local variable x that can hold data of a type corresponding to the variable declarator used.

Local variables retain their value only for the given evaluation of the formula in which they occur.

Comments

- A variable declarator must be used to declare a variable before the variable can be used in a formula.
- Local specifies a variable with local scope. You can use the keywords Global, and Shared instead of Local to specify different scopes. You can also omit the scope keyword and start the variable declaration with the type name. This is the same as declaring a Global variable. See Variable Scope (Crystal syntax) and Variable scopes for Crystal syntax.

Related topics

Variable declarations (Crystal syntax)

Declaring array variables (Crystal syntax) (Crystal syntax)

Scope operators

The scope of a variable is an attribute of that variable specifying where you can use or reference it, e.g., within the same formula where it is defined, or within and outside of the formula where it is defined.

Variable scopes for Basic syntax

Variable scopes for Crystal syntax

Variable scopes for Basic syntax

Declaring a scope for your variable regulates its availability through different parts of your report. See Variable declarations using Dim (Basic syntax) and Variable Scope (Basic syntax).

The variable scopes available are:

- Dim
- Local

- Global
- Shared.

Dim specifies a variable with local scope. Local and Dim are equivalent keywords.

Related topics
Variable overview (Basic syntax)

Variable scopes for Crystal syntax

Declaring a scope for your variable regulates its availability through different parts of your report. See Variable Scope (Crystal syntax)

.

The variable scopes available are:

- Global
  The variable is available to formulas throughout the entire current report.
- Shared
  The variable can be shared with a subreport as well as the entire current report.
- Local
  The variable is specific and can only be used in the formula in which it is defined.

To use variable scopes you add the scope that you want to declare with your variable at the beginning of your variable declaration statement. You can also omit the scope keyword and start the variable declaration with the type name. This is the same as declaring a Global variable.

Examples

```
NumberVar Amount;
```

Declares a global variable named Amount that can hold any value of an integer or decimal data type.

```
Global NumberVar Amount;
```

Declares a global variable named Amount that can hold any value of an integer or decimal data type. This variable would be available to formulas throughout the current report

```
BooleanVar Outstanding;
```

Declares a global variable named Outstanding that can hold a TRUE or FALSE value.

```
Shared BooleanVar Outstanding;
```

Declares a shared variable named Outstanding that can hold a TRUE or FALSE value. This variable would be available in both the main report and any subreports contained within the main report.

```
StringVar LastName := "Adams";
```

Declares a global variable named LastName that can hold a string value and assigns the string "Adams" to that variable.

```
Local StringVar LastName : = "Adams";
```

Declares a local variable named LastName that can hold a string value and assigns the string "Adams" to that variable. This variable would be available only within the formula which it was declared.

Comments

- Your variable declaration statement is used to set aside a block of memory to hold the value assigned to the variable, and to assign a default value to that memory block. The default value assigned depends on the data type stored by the particular variable declarator used and on whether or not a value was assigned when the variable was declared.
- A variable declared in two formulas uses the same block of memory (that is, it refers to the same value.) For example:

```
//Formula #1

NumberVar x;

x:=5

//Formula #2

NumberVar x;

x:=x+15
```

x now has the value of 20.

Related topics
Variable declarations (Crystal syntax).

Expression separator

Crystal syntax.

Operator (Symbol/Word)

```
;
```

Usage

```
1+1; "abc"
```

1+1 and "abc" are two different formula expressions in a multiple expression formula. The semicolon between the expressions specifies where one expression ends and the next one begins. Without the semicolon, the two expressions would be treated as an individual expression.

Comments

This operator is used in only Crystal syntax to separate consecutive expressions. In Basic syntax, a line break or a colon is used to separate consecutive statements.

Related topics

Expressions (Crystal syntax)

Constants

These constants can be used in the formula language with:

- date functions
- conditional date formatting
- conditional font formatting
- text interpretation in the Report Designer.

Constants used with Date functions

Day of Week constants

First Week of Year constants

Constants used in conditional formatting formulas

Day of Week enclosure constants

Font-formatting constants

Text Interpretation constants

Mathematical constants

Text Interpretation constants

Day of Week constants

The following constants are defined with the corresponding numeric values. You may use them with date functions which accept firstDayOfWeek as a parameter:

Constant	Description
crUseSystem	0 (Use the NLS API setting.)
crSunday	1
crMonday	2
crTuesday	3
crWednesday	4
crThursday	5
crFriday	6
crSaturday	7

First Week of Year constants

The following constants are defined with the corresponding numeric values. You may use them with date functions that accept firstWeekOfYear as a parameter:

Constant	Value	Description
crUseSystem	0	Use the NLS API setting.
crFirstJan1	1	Start with week in which January 1 occurs (default).
crFirstFourDays	2	Start with the first week that has at least four days in the new year.
crFirstFullWeek	3	Start with first full week of the year.

Day of Week enclosure constants

The following constants are defined to support international formatting of day of the week, such as enclosing the day of the week in round or square brackets, or not enclosing it.

Constant	Description
crDayOfWeekNotEnclosed	Do not enclose the day of a week in a Date or DateTime field with brackets.
crDayOfWeekInParentheses	Enclose the day of a week in a Date or DateTime field with round brackets. This is the default with English fonts.
crDayOfWeekInFWParentheses	Enclose the day of a week in a Date or DateTime field with full-width round brackets which are available in Japanese fonts.
crDayOfWeekInSquareBrackets	Enclose the day of a week in a Date or DateTime field with square brackets. This is the default with English fonts.
crDayOfWeekInFWSquareBrackets	Enclose the day of a week in a Date or DateTime field with full-width square brackets which are available in Japanese fonts.

Font-formatting constants

Constant	Description
crRegular	Format font in regular type face.
crBold	Format font in bold type face.
crItalic	Format font in italic type face.
crBoldItalic	Format font in bold italic type face.

Text Interpretation constants

The following constants are defined to support importing data which is pre-formatted in Rich Text Format or HTML, and inserting it into a report as string or memo fields.

Constant	Description
crUninterpretedText	Interpret data as plain text.
crRTFText	Interpret data as Rich Text Format data.
crHTMLText	Interpret data as HTML data.

Mathematics constant

crPi - the Mathematical value *pi* which is 3.14 (if rounded to 2 decimal places).

Create Client Reports with Crystal Reports 2011

ViewPoint now uses the fully functional Crystal Reports 2011 to create Client Reports. These objects are still saved on the Power i so they can be accessed by any ViewPoint user, and managed in one central location. Once installed and licensed, Crystal Reports 2011 is launched externally anytime you create a new report from a ViewPoint view, or open an existing Client Report object from ViewPoint.

Crystal Reports 2011 includes its own fully documented built-in help system to assist you with all the features it provides. Access it by pressing the Help button on the Designer Toolbar.

Crystal Reports Workflow

Use this workflow to create a new Client Report with Crystal Reports 2011. The following four steps will create most of your report requests:

Start the Report Designer

Layout Your report

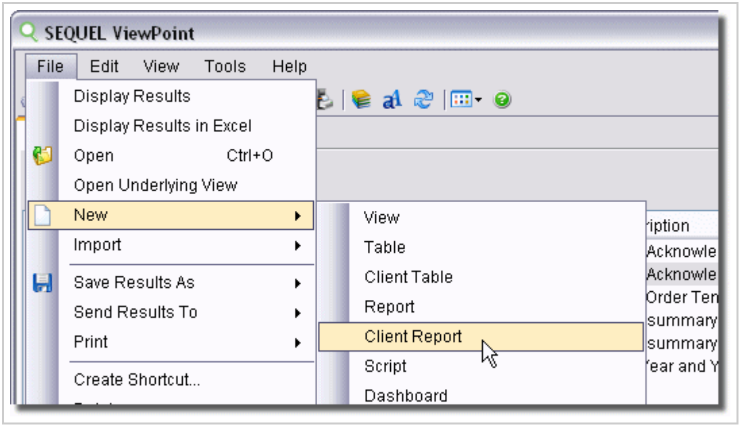
Add Optional Features and Functions

Save Your Client Report on the Power i (iSeries)

Start Crystal Reports from a View

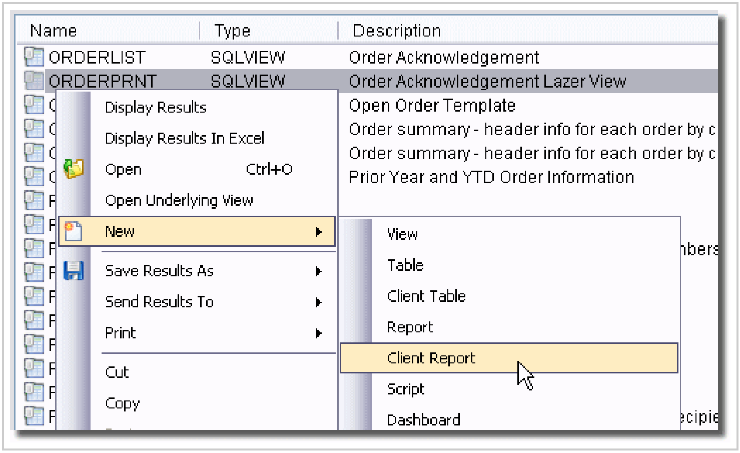
Crystal Reports 2011 is launched from the ViewPoint Explorer, or from the view designer as shown below:

From the ViewPoint Explorer, first highlight a view or table in the object list, and on the menu bar, select **File\New\Client Report.**



-or-

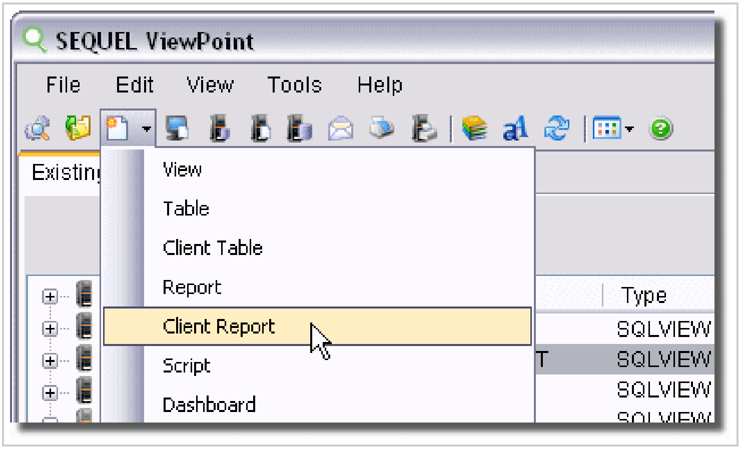
Right-click the desired view or table and select **New\Client Report** from the drop-down menu.



Name	Type	Description
ORDERLIST	SQLVIEW	Order Acknowledgement
ORDERPRNT	SQLVIEW	Order Acknowledgement Lazer View
		Open Order Template
		Order summary - header info for each order by c
		Order summary - header info for each order by c
		Prior Year and YTD Order Information

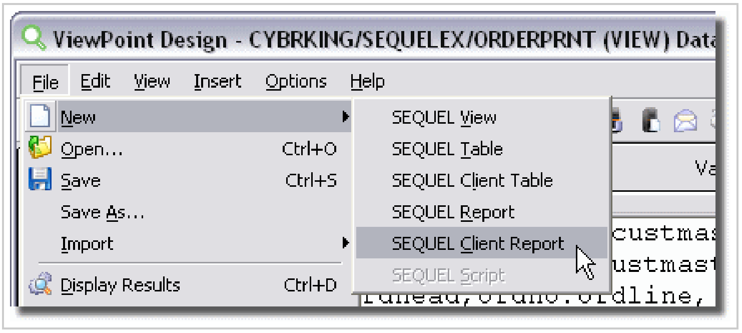
-or-

Highlight a view or table in the object list, press the **New** button on the toolbar, and select **Client Report.**



-or-

From the menu bar on the Design View display or the Table Designer, select **File\New\Client Report.**



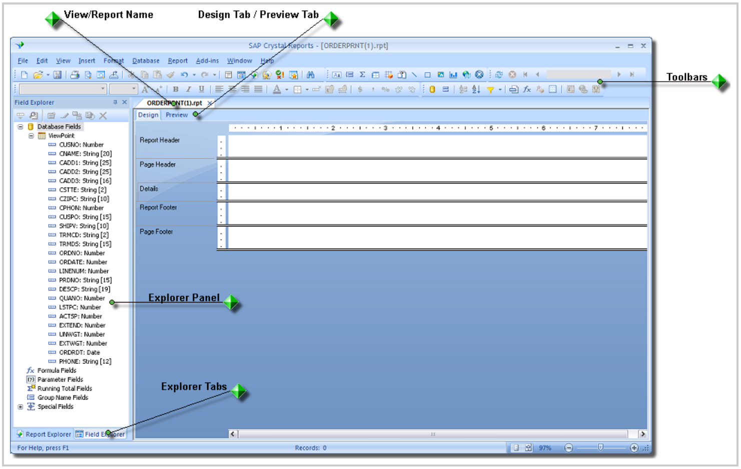
Crystal Reports 2011 will open in a separate window where you can layout your report.

Layout your Report

When launched from a ViewPoint view, the Crystal Reports 2011 will open with a display similar to the one below. The screen contains the following elements:

- The **Field Tab** in the **Explorer panel** (on the left) lists all the ViewPoint fields under the Database Fields node.
- The **View/Report Name tab** displays the name of the opened or saved client report. If you start Crystal from a view, and have not saved the report, the view name will appear in the tab.
- The **Design tab** is where you will layout your report.
- Use the **Preview tab** to see how your report will look as you go along with your design.
- All the Crystal functions and features are accessed using either the **Menu bar**, **Toolbar**, or right-click contextual menus.

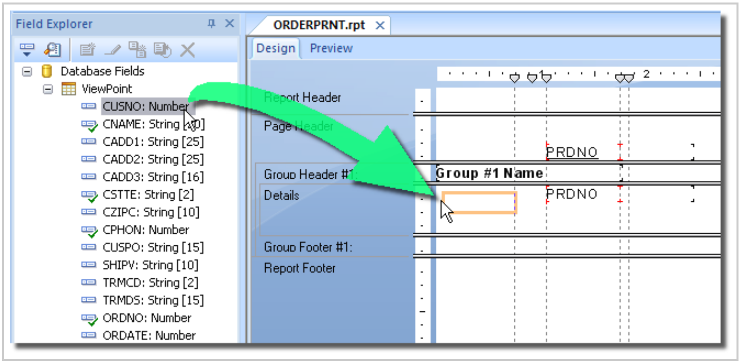
On this screen you will add fields from your view, create subtotal groups and sorting, add subtotal calculations, and add any optional items such as charts (graphs), crosstab charts, and more.



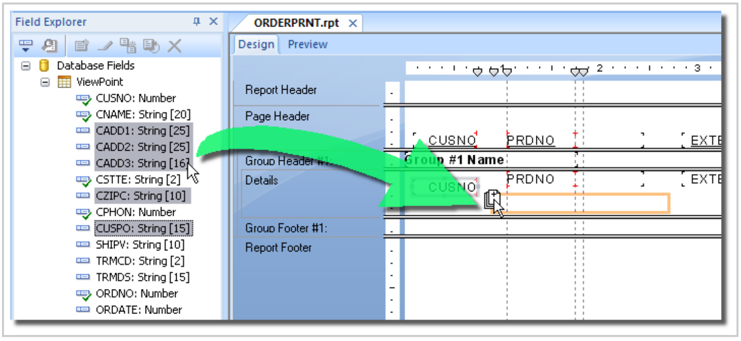
Add Fields from Your View

Adding fields to your report is very easy. Either method that follows will add the field and its corresponding column heading to the report.

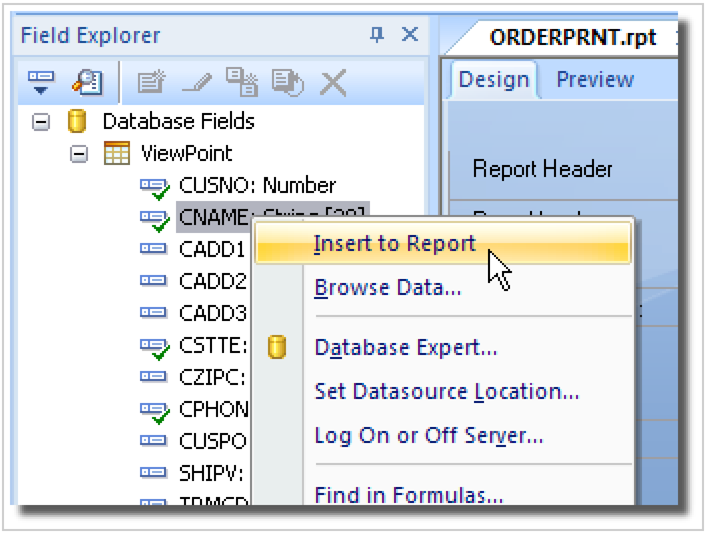
Click and drag a single view field from the **Field Explorer** and drop it on any group or format you like (usually the Details format).



You can also hold down the Shift or Control key while selecting multiple fields, and then drag them to the report like so.



Lastly, you can right-click a field and select **Insert to Report**.



Now you are ready to adjust your column headings and add any desired additional features and functions.

Add Optional Features and Functions

Below are brief instructions on how to add some optional, but common items to your report. For complete information on these features, please refer to the Crystal Reports 2011 online Help.

Subtotal Levels (groups)

Use Group Expert to add subtotal levels (groups) and select sorting options

If you want to create an aggregate (sum, min, max, etc.), you have to first create a group for the subject of the aggregate. This means if you want to 'sum an amount field by X', where 'X' is month, region, or product, then you have to first create a group for month, region, or product.

1. To create a new group, press the **Insert Group** button , or select **Insert\Group** from the menu.
2. Select the grouping field from the first drop-down list.
3. Set the sort order as desired.
4. (Optional) Set the **Group Name Field** on the **Options** tab.
5. (Optional) Set the group printing options (**Keep Group Together** and **Repeat Group Header on Each Page**) on the **Options** tab.
6. Press **OK** when finished.

- Crystal Reports groups items first, then sorts within the group.
- The Group Name Field is always a string.
- Date fields have additional grouping options based on time intervals.

Subtotal Calculations

Add summary calculations (subtotals and grand totals) to any group.

1. To create a summary calculation, select the field you wish to summarize.
2. Press the **Insert Summary** button .
3. Select the type of calculation from the **Calculate This Summary** drop-down list.
4. Select the **Summary Location** (Group Footer, Grand Total/Report Footer) and press **OK**.
   Note: You can also select **Insert\Summary** from there menu, or right click a field and select **Insert\Summary** to create totals. The default summary operation is Sum if a number field is selected and Maximum if a text field is selected.

To change the Summary Operation:

1. Right-click the existing Subtotal, Grand Total, or Summary field, and select **Edit Summary**.
2. Select the new desired Summary Operation.
3. Press **OK** when finished.
   Note: If no group exists, choose **Insert\Group** and create a group.

Selection Criteria

Set additional selection criteria (if not done in the view).

1. Select the field for which you wish to write criteria.
2. Press the **Select Expert** button Σ . (If a field was not selected in Step 1, Crystal Rreports presents a list of all available fields to create the formula.)
3. Select a comparison operator from the first list.
4. Select a value from the second list, or type your own.
5. Set other criteria as needed using the **NEW** button or the **NEW** tab.
6. Press **OK** when finished.
7. If the report is in preview mode when the selection criteria have been set, CR prompts to filter **Use Saved Data** or **Refresh Data** from the source. Press the **Refresh Data** button.

Charts (graphs)

Use charts to view your data visually. Charts are normally based on grouped informatioin and can be placed in the Report header/footer or the Group header/footer. Do the following to add a chart and set its options.

1. Press the **Insert Chart** button , or select **Insert\Chart** from the menu.
   An object frame appears as you drag the chart object into the report
2. Place the object frame in an empty area in the Report header/footer or Group header/footer and click to release it. The Chart Expert dialog display opens.
3. On the **Type** tab, in the **Chart type** list, select a chart type.
4. Click the chart subtype that best illustrates your data.
5. Click the **Data** tab.
6. In the Data area, in the **On change of** list, click the group field you want to base your chart on; then, in the **Show** list, click the summary field you want to display on your chart.
7. If the **Axes** and **Options** tabs appear, you can customize some of the chart's properties, such as the scaling of the axes, the legend, and the data points.
8. Click the **Text** tab.
9. Accept the default title information or add new titles to your chart.
10. Press **OK**.

    Note: When your chart is inserted, it may cover a portion of the report. Move and resize the chart so that it fits properly within the report

Crosstab Charts

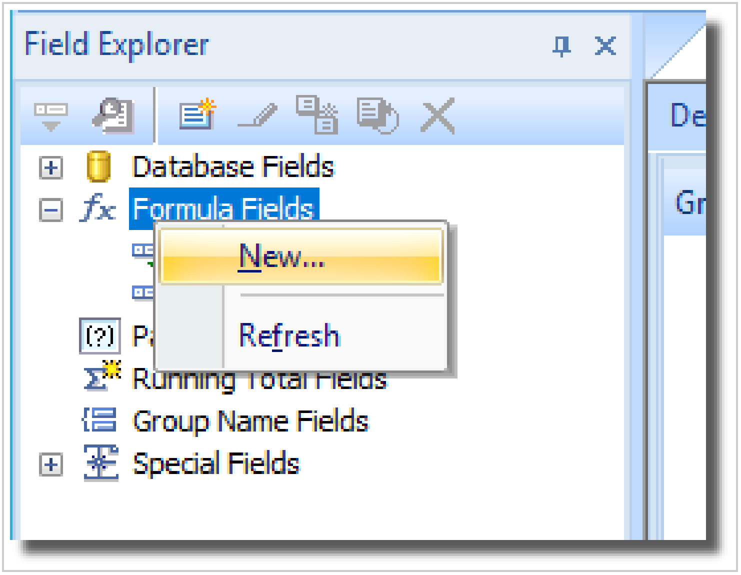
A crosstab chart displays data in rows and columns with a summarized field. Crosstab charts can be placed in the Report header/footer or the Group header/footer. Do the following to add a crosstab chart and set its options.

1. Press the **Insert Crosstab** button , or select **Insert\Crosstab** from the menu.
   An object frame appears with the Arrow cursor as you drag the Crosstab object into the report.
2. Place the object frame in an empty area in the Report header/footer or Group header/footer and click to release it.
   An empty crosstab object appears in your report. You can drag fields from your report or from the **Field Explorer** into your crosstab. You can also use the **Crosstab Expert** to build the crosstab chart.
3. Right-click crosstab and select **Crosstab Expert** from the drop-down menu.
4. In the Crosstab Expert display, select fields for the **Rows**, **Columns**, and **Summarized fields** sections.
5. Press **OK** when finished.

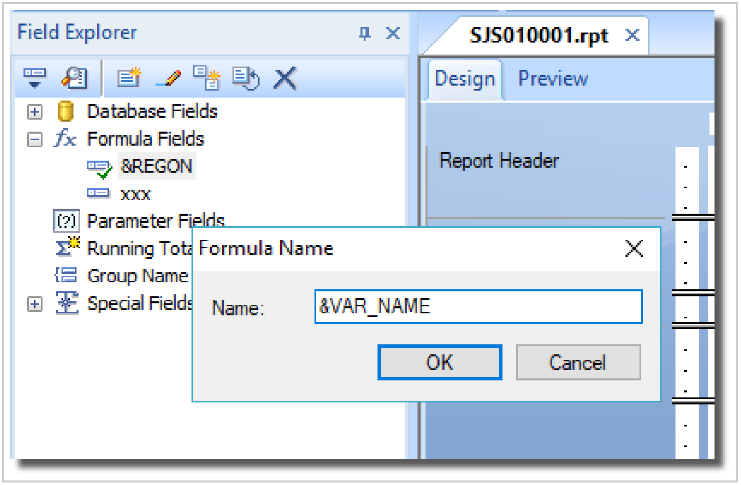
Add a View Variable to a Report

If your underlying view contains runtime promts, sometimes you might want to display the prompt value in your report. Follow the steps to create a formula field, and add it anywhere to the report.

1. In the Field Explorer, right-click Formula Fields and select **New**.

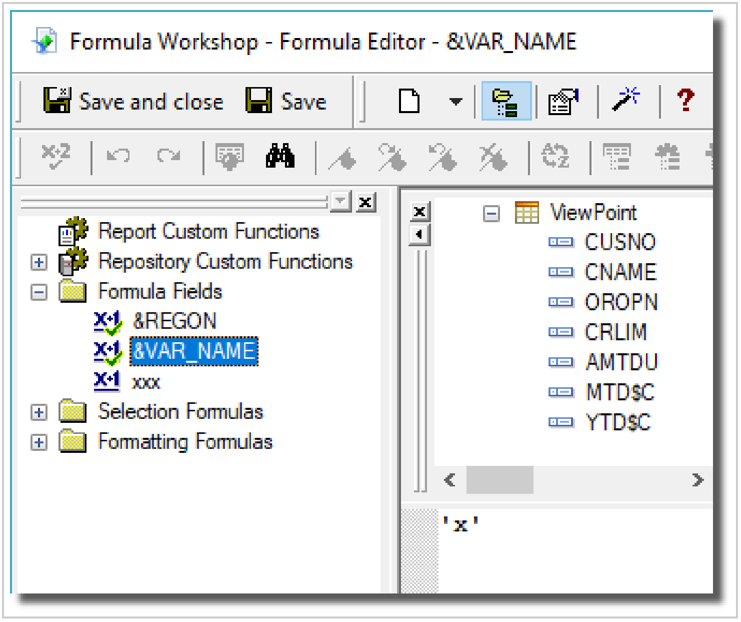


2. The Formula Name window displays. Give this new formula the same name as the variable from your view.



Be sure to include the ampersand (&), and use uppercase like so: &VAR_NAME. Press **OK**.

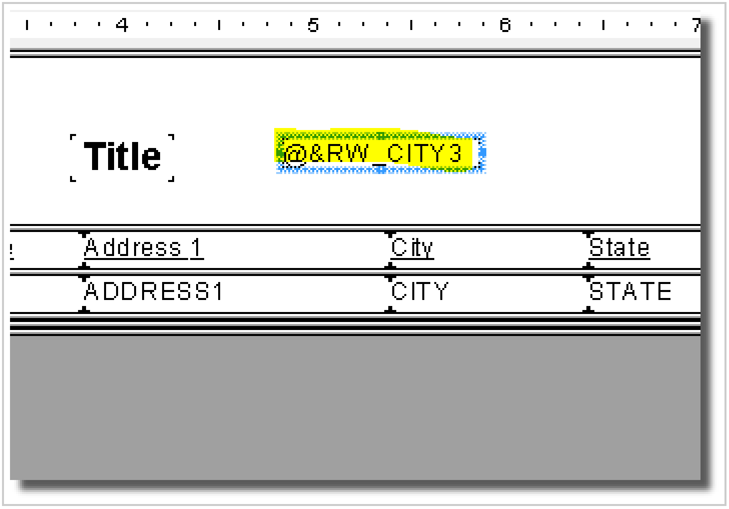
3. The Formula Editor window displays.



```
'x'
```

Enter a placeholder value for the formula. ('x' for character. '0' for numeric) Press the **Save and Close** button.

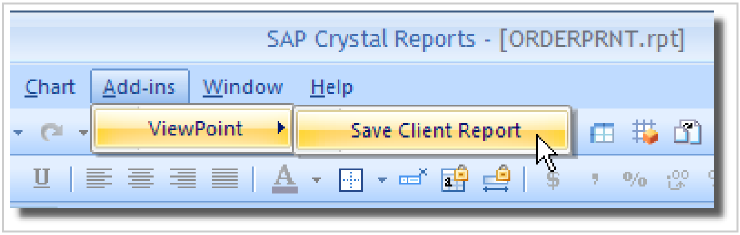
4. Back at the report designer click and drag the new formula field from the Field Explorer to any section of the report.



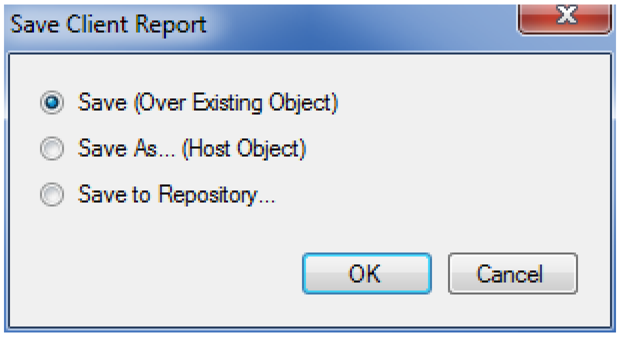
NOTE:
You wont see the prompt value in the Preview tab of the report editor. You must save the report, and run it from Viewpoint to see the value.

Save Your Report

To save your Client Report, select **Add-ins\ViewPoint\Save Client Report** on the menu.

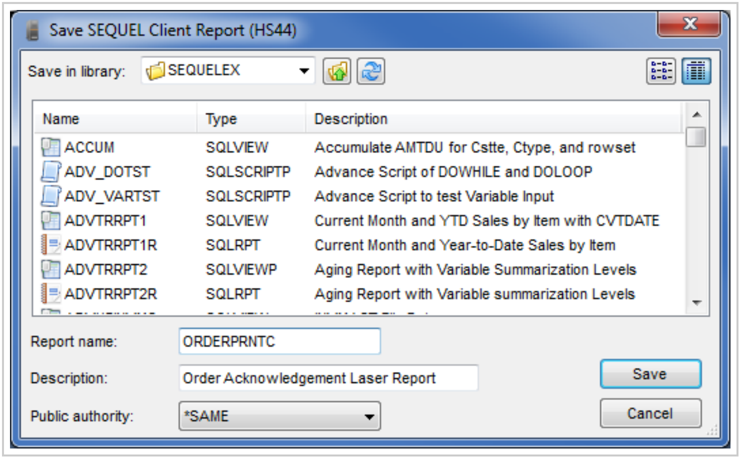


You are presented with three save options: Save (over existing), Save As (new), and Save to [ViewPoint] Repository. Make your selections and press **OK**.



The first two options, Save and Save As, use the same Save dialog.

Specify a name for the Report (or leave the name unchanged if saving an existing Report), a description or title, a library and press **Save**. A message will inform you when the Report is saved.



Save SEQUEL Client Report (HS44)

Save in library: SEQUELEX

Name	Type	Description
ACCUM	SQLVIEW	Accumulate AMTDU for Cstte, Ctype, and rowset
ADV_DOTST	SQLSCRIPTP	Advance Script of DOWHILE and DOLOOP
ADV_VARTST	SQLSCRIPTP	Advance Script to test Variable Input
ADVTRRPT1	SQLVIEW	Current Month and YTD Sales by Item with CVTDATE
ADVTRRPT1R	SQLRPT	Current Month and Year-to-Date Sales by Item
ADVTRRPT2	SQLVIEWP	Aging Report with Variable Summarization Levels
ADVTRRPT2R	SQLRPT	Aging Report with Variable summarization Levels

Report name: ORDERPRNTC

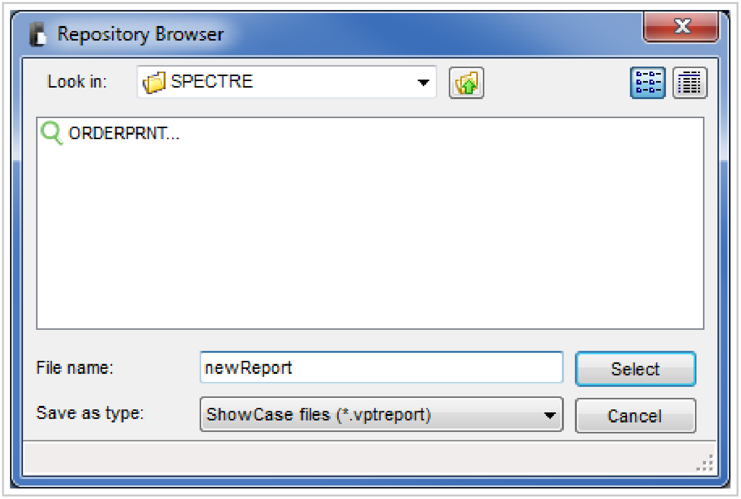
Description: Order Acknowledgement Laser Report

Public authority: *SAME

Remember: This report is saved on the Power i (iSeries) as a user space object (USRSPC). If you use the Crystal Reports 2011 option to save the report locally with the data (we don't recommend this), the report will become static. This local report has no way to connect back to the underlying view to return current data.

If you chose Save to Repository, use this dialog to navigate to a folder in the repository ‘root’ located at: \\IBM_i_name\root\sequel\swi\repository.

Select the root, or a folder listed under the root for your report. Enter a file name and press **Select** to save.



Repository Browser

Look in: SPECTRE

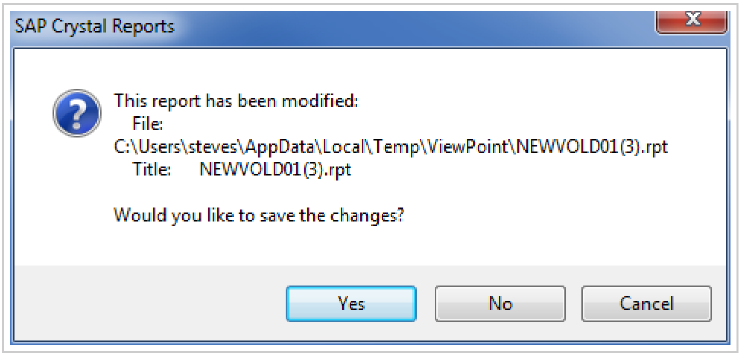
ORDERPRNT...

File name: newReport

Save as type: ShowCase files (*.vptreport)

Watch out for this panel (warning)

If you attempt to close the Crystal Reports designer after making a change or creating a new report, you will see a dialog (below) asking you to save the temporary work file created by Crystal Reports (notice the file path). This <u>does not</u> save your report on the host.



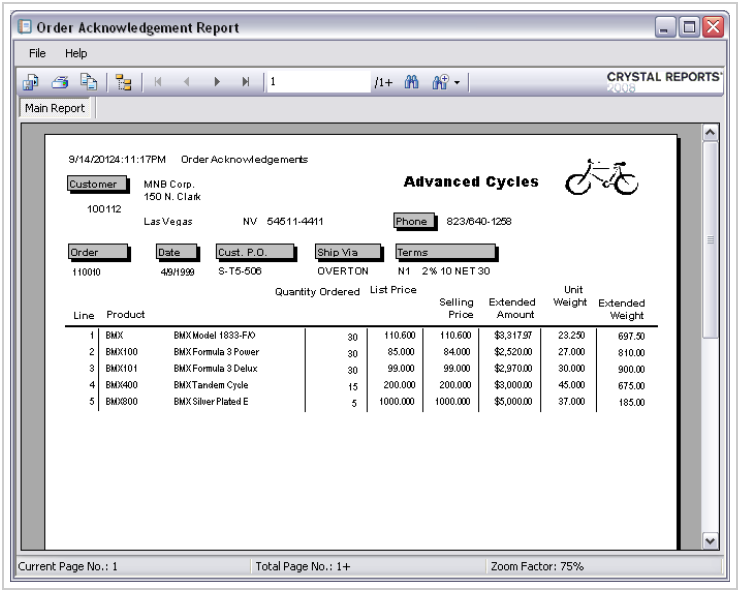
This report has been modified:
File:
C:\Users\steves\AppData\Local\Temp\ViewPoint\NEWVOLD01(3).rpt
Title: NEWVOLD01(3).rpt

Would you like to save the changes?

Press the **Cancel** button and make sure you have saved the report on the host as described in the 'Save Your Report' section above. Then when you close the designer you can reply to this dialog with the **No** button.

Displaying Report Results

Double-click a Client Report in the ViewPoint Explorer to display the report on your desktop.



9/14/20124:11:17PM Order Acknowledgements

Advanced Cycles

Customer: MNB Corp.
150 N. Clark
100112
Las Vegas NV 54511-4411
Phone: 823/640-1258

Order	Date	Cust. P.O.	Ship Via	Terms
110010	4/9/1999	S-T5-506	OVERTON	N1 2% 10 NET 30

Line	Product		Quantity Ordered	List Price	Selling Price	Extended Amount	Unit Weight	Extended Weight
1	BMX	BMX Model 1833-F/O	30	110.600	110.600	$3,317.97	23.250	697.50
2	BMX100	BMX Formula 3 Power	30	85.000	84.000	$2,520.00	27.000	810.00
3	BMX101	BMX Formula 3 Delux	30	99.000	99.000	$2,970.00	30.000	900.00
4	BMX400	BMX Tandem Cycle	15	200.000	200.000	$3,000.00	45.000	675.00
5	BMX800	BMX Silver Plated E	5	1000.000	1000.000	$5,000.00	37.000	185.00

Current Page No.: 1 Total Page No.: 1+ Zoom Factor: 75%

Index