Second Edition

MASTERING CUSTOM FIELD FORMULAS

in Microsoft® Office Project

For the Users of All Desktop Versions/Editions Supporting the Feature

Ismet Kocaman

Mastering Custom Field

FORMULAS

in Microsoft® Office Project

Ismet Kocaman

About the Author

Ismet Kocaman is a Management Consultant, Project Management Consultant, Technical Project Manager and a Mechanical Engineer with over 20 years of experience in the manufacturing sector.

He is currently providing management consultancy to the companies in the manufacturing sector on system improvement projects and technical projects. He also conducts training seminars for engineers on Project Management and MS Project with focus on the project management process in the manufacturing environment.

He is a Project Management Professional (PMP) and holds several Microsoft certifications on MS Project.

Visit the author’s website for more information at http://www.ismetkocaman.com

Contents

Introduction ____ 1

Getting Started ____ 5

Creating a Formula in Project 5
Steps to Follow While Developing Formulas 10
Step #1: Determine whether you really need a formula 10
Step #2: Define output from the formula 11
Step #3: Define input to the formula 11
Step #4: Determine which operations the expressions in the formula should perform on the information inputted (or passed) 11
Step #5: Test the formula's syntax and logic 12
Step #6: Document the formula 12
Fields in Project 12
What is a field ? 12
What is a field reference in a formula ? 13
Properties of a Field 13
Field Names 13
Field Categories 14
Field Types 14
Entry Types 18

Formulas in Project ____ 21

What is a Formula ? 21
Literals 22
Operators 24
Arithmetic Operators 24
Comparison Operators 27
Logical Operators 28
Concatenation Operators 29
Operator Precedence 30

Built-in Functions ____ 33

Built-in functions for the custom fields 33
How to use a built-in function in a formula 34
Function descriptions 37
How to interpret a function description 38
Data Types 39
How to use data type information 41
How to handle the returned values based on the data type information 41
How to handle the parameters based on the data type information 43
Working with String parameters 44
Working with logical values 45
Functions accepting logical values: iif and Switch functions 45
Functions returning logical values 49
Using the Choose function 53
Constants Recognized in Formulas 55

Settings for the Custom Fields ____ 57

How to Delete or Disable a Formula, How to Clear or Protect a Custom Field's Data 57

How to Import a Custom Field 58
Custom Fields in Summary Rows 58
Rollup Methods 62
Graphical Indicators 64

Special Operators: Like and Between...And ________ 69

Inserted Projects and the Custom Fields ________ 73

Errors in Formulas ________ 79
Logical Errors 79
Syntax Errors 84
#ERROR Errors 86
How to filter for #ERROR lines in custom fields 88

Working with Custom Flag and Text Fields ________ 91
Returning Numeric, Date or Logical Values to the Custom Text Fields 94
The built-in functions for handling text information 98
The StrConv function 98
The StrComp function 99
The InStr function 100
Using Str, CStr and Format functions in formulas 101
Format expressions used in the Format function 103

Working with Custom Number and Cost Fields ________ 111
How to enter values to the custom number and cost fields by typing in 111
How do the custom number and cost fields interpret the values returned from the formulas 113
Rounding Numbers in Formulas 115
How does Project round off the numeric data in the custom fields 115
Using the built-in functions to round off the numeric data 118
Using the Int or Fix function to truncate the numbers 118
Using the Int function to round up the numbers in formulas 119
Using the Int and Fix functions to arithmetically round the numbers off 120
Some examples of custom rounding on monetary values 121
Rounding off to the nearest even integer 122
Implementing Excel's worksheet function Mod in formulas 123
Implementing the round-half-to-even algorithm in formulas 123

Working with Duration and Work Data ________ 127
Duration Type Fields in Project 127
How to enter duration and work values in Project 129
Customizing Project's default time units 129
Customizing the time unit labels 130
How does Project display values in the duration type fields ? 131
How does Project determine the conversion factors between the units ? 131
How can we change the conversion factors between the units ? 132
How are the duration and work data stored in the duration type fields ? 133
How to pass the conversion factors to formulas 135
Using ProjDurConv or ProjDurValue functions to access Days per month setting in a formula 135
How does it affect the schedule if we change "Hours per day" value ? 136
Changing the default units in a project plan containing duration and work data 138
Limits for the Duration type fields 139
Built-in Functions used in Duration and Work Calculations 142

Some Examples with Duration Type Fields 142

Working with Dates and Times 145

Customizing Project's default date display format 145
Default time values 147
How to enter date and time in Project 148
The Date Type fields in Project 149
Referencing the date type task fields in formulas 150
Referencing the date type resource fields in formulas 151
Using the fields listed under the Project / Date category in formulas 152
How does Project store date and time values ? 153
Date field limits 156
Date and Time Functions 158
Date, Time, Now and Timer Functions 159
Year, Month and Day Functions 160
Hour, Minute and Second Functions 161
Passing Dates to Functions As Parameters 161
Using Date Strings to Pass Dates to Functions 161
Using Date Literals to Pass Dates to Functions 163
Date Separators 165
DateSerial Function 165
CDate Function 167
DateValue Function 169
TimeValue Function 170
TimeSerial Function 171
DateDiff and DatePart Functions 172
Week and Weekday Number Calculations with DatePart and DateDiff 175
Using date arithmetic in week calculations 179
Calculating Quarters 179
Weekday, WeekdayName and MonthName Functions 180
What is the First Day of the Week ? 181
Calculating the start and end dates of a week based on a date 183
Using a formula to display the system's first day of week setting 185
What is the First Week of the Year ? 185
Calculating Week Number of a Date 187
DateAdd Function 189
Using Today, Tomorrow and Yesterday in Formulas 190
Comparisons and arithmetic operations with dates and times 190
Using Hardcoded Dates and Times in Formulas 192
Setting a Date Field's Value to NA 194
Handling the Initial Value "NA" of a Date Field in Formulas 194
Numeric Comparisons to Test for NA 195
Using IsDate Function To Test for NA 196
Other Methods to Test for NA 196
Working with times 197
How to compare times in Formulas 197
Referencing the default start time in a formula 197
How much precision do we need in the time serial ? 197
Comparing times 198
Other methods to compare times 198

Working with Manually Scheduled Tasks ________ 201
What is a placeholder task ? 201
Formulas performing operations based on task mode........ 203
Formulas working only for manually scheduled tasks........ 203
Processing text information (including blank) of a placeholder task........ 203
Processing valid duration or start/finish date values of a manual task 204
Formulas working only for auto-scheduled tasks........ 206
Formulas performing operations on tasks in any task mode 207
The New Baseline Fields........ 207
Calculating the variances for manually scheduled tasks 208
Formulas to see how the text information in the fields Duration, Start or Finish changes 214

The Built-in Functions in Project Category ________ 219
Date Separators Used in Date Strings Passed to Functions........ 219
Pj Enumeration Constants Recognized in Formulas 220
ProjDurValue function 225
ProjDateConv Function 225
ProjDurConv function 227
ProjDateAdd and ProjDateSub Functions 230
ProjDateDiff Function 232
ProjDateValue Function 235
Using ProjDateValue to Test for NA 238

Index ________ 239

Introduction

This book has been prepared for readers, who currently have skills to use Project as a planning and scheduling tool in managing projects; and who also want to learn how to make effective use of the custom field formulas (also referred to as user-defined formulas or just as formulas) while working on the project data.

The content of this book is only related to processing a project's task and resource data by using custom field formulas, therefore no part of this book covers the topics related to managing projects with Project. The author assumes that the reader has some basic math knowledge and skills to write formulas or expressions for other purposes, and is able to extend that knowledge and skills to write formulas in Project by using the built-in functions mentioned and the techniques explained in this book.

Even though Project contains plenty of the data fields of various types and categories presenting the planned, scheduled and actual project data interpreted in many different ways to the users, one might still need some custom information that is not directly available in the existing fields. Project is a highly customizable application such that it enables us to build our own formulas in the custom fields in order to perform further operations on the existing data to produce some new data needed. This feature was added to the product with the version, Project 2000.

In Project, there are, of course, several other ways to calculate custom data. For example, the existing project data can be processed by using VBA code, or exported to Excel for processing further by using Excel's tools, in order to generate the new data needed. On the other hand, none of these methods, if not chosen by personal preference, would be more practical to implement than using formulas, provided that a solution by using the custom field formulas already exists in a particular case. It is not required to have programming skills in order to be able to develop and maintain the custom field formulas. Besides we do not need to leave the current Project view to input data to and test the formulas. No further action of the user is required to recalculate the formulas when the data referenced change in a project plan, since Project automatically re-evaluates the custom field formulas and refreshes the associated fields with the resulting data accordingly.

As mentioned above, the custom field formulas are mainly used to calculate some custom schedule or cost data that is not available in a project plan. There are many other useful applications of the formulas, such as displaying a field's data stored in the background when we want to understand how Project performs certain calculations. Other than that, there may be cases where the data calculated are not used literally, but instead, the results of the formulas are used to filter, group or sort the project data, based on one or more criteria. And also multiple conditions in one or more filters can be combined into a single formula which helps us to immediately identify the tasks or resources satisfying all these conditions. In a manner, they work like always-active filters.

We must constantly monitor the schedule, work and cost performance of a project during the implementation phase otherwise we cannot take timely actions to avoid problems that have potential to cause delays and/or budget overruns. Formulas may be created to evaluate the schedule, work and cost performance of a project. Then the results, combined with the other progress related information collected, can be interpreted further to determine the overall status of the project. As a result, a performance check formula is the one created to constantly watch whether a certain schedule or cost data in a task or resource field of the current project plan meets a condition. The condition may be composed of a single comparison expression testing a field's value against a certain threshold value that we specify or may be a combination of several other conditions testing whether a field's value falls into a range that we specify. In both cases, the formula produces a single result. In the former one, the result represents either one of a two-state information, such as Yes and No, where use of a flag field suffices. But in the latter case, the result represents multiple-state information, therefore, a non-flag type custom field should be used to display the returned values. In both cases, the custom field with formula can be set to display graphical indicators since visual representation of the data enables us to immediately see the items requiring attention, that would otherwise be difficult without a filter, for example, while working on a long task or resource table. We will cover how to define graphical indicators by using the **Custom Fields** dialog box later.

The purpose of this book is not to discuss when a project is considered as well-performing, and then how performance check criteria (or conditions) can be defined in a formula, based on that information. Instead, we will focus on how to use field references and functions to build custom field formulas processing project's task or resource data to produce accurate results. Once you learn how to build formulas correctly by using this book, you can easily develop your own set of formulas performing certain checks on any schedule.

Although it is also outside the scope of this book, formulas can be created to perform the schedule quality assessments. These formulas are used to check to see whether the structure and elements of a schedule are in conformance with certain standards or guidelines. Then if required, the schedule is further fine-tuned to meet the criteria specified by the standard, based on the results of the formulas. The schedule quality assessment is continuously conducted on a schedule at all phases of a project, as part of the schedule development and maintenance process. Note that it may not be always possible to convert a schedule quality assessment criterion to a condition tested by a custom field formula. For example, parsing predecessors or successors field for link types, lags or leads for some checks is not possible by using formulas since a formula cannot loop through the delimited list of the items in these fields. Although formulas offer a limited capability on this area, they prove useful for many checks.

Project's backward compatibility and multi-language support features handle the conversions for the components of the formulas while saving the current file to the file format of the previous versions or opening a file from the file format of the earlier versions, or while opening a project plan file created by using a version of the product installed in a system with a different locale. But it is strongly recommended that you always check the custom field formulas to ensure that they return the results as expected after these operations. Note that Project does not change the strings, that is, any text enclosed in double quotation marks, in formulas.

The name "Project" will be used throughout the book in place of the complete product name. The term "formulas" is to be used throughout the book as the plural form of "formula", instead of "formulae". The fields other than the custom fields are referred to as "non-custom fields" or "regular fields" (which are not technical terms) when we need to differentiate them from the custom fields in the context.

All the formulas have been tested with the latest standalone desktop version of the product that exists at the time of producing this book; it is the version 1910 (Build 12130.20410 C2R) of MS Project Professional 2019. Beware that the results from the formulas may change as the product is updated with the new public updates or service packs released, or when the formulas are used in the future versions of the product. The formulas presented in this book can also be used in the previous versions of the product such as 2003 and 2007; especially, the users of the versions prior to 2013 may find some formulas (that is, the ones working similar to some built-in interactive filters of the version 2013 and later) quite useful. But note that displaying some of the formula results in the graphical reports is only possible in the versions 2013 and later, and the formulas handling data of the manually-scheduled tasks can be used only in the versions 2010 and later.

Getting Started

In this first section of the book, we will explore the basics of a formula in Project. Let us get started with creating a simple formula as an example and then discuss some basic concepts as we proceed through the steps in the example.

Creating a Formula in Project

In Project, we cannot type in a formula directly into a custom field, or into the entry bar. We have to use commands and dialog boxes for this purpose. A custom field that does not contain any formula is, of course, available to enter information by typing in. Once we have entered a formula to a custom field, the field is then closed for entries by typing in, and it only displays the results of the formula.

In the example below, we will create a simple formula in order to demonstrate how to enter formulas to the custom fields and to explain how the formulas work in Project, so follow the steps:

- Start Project. Project automatically opens a blank project file and displays the default view; that is, the Gantt and with Timeline view, in versions 2010 and later, which contains the Gantt Chart view in the bottom pane. We will now work in the default task table displayed (i.e., the Entry table) on the Gantt Chart view.

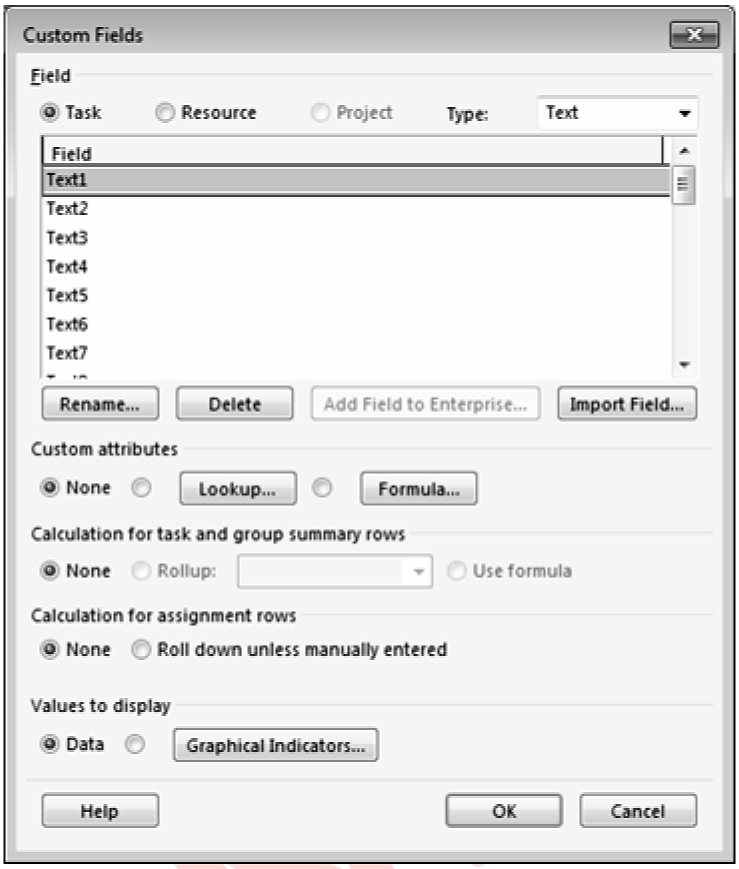
- Insert the custom text fields, **Text1** and **Text2** into the blank task table to any column position that you like. And enter the text **Hello, World !** into the **Text1** field in the first row of the task table; Project will automatically create the first task entry in the task table.

  Note that it is not required to display a custom field in a table to enter a formula for that custom field. Project will evaluate the formulas in the hidden custom fields too.

  Sometimes, a long formula in a custom field is divided into small formulas to overcome a limitation; at other times, a complex formula is divided into small formulas to manipulate it easily during the tests. In either case, the other custom fields used to enter the small formulas are referred to as intermediate fields. You can hide the intermediate custom fields when you are done with testing the formula.

- The **Custom Fields** dialog box is the only place where we enter the formulas and adjust all the settings related to the custom fields (we will explore the sections of this dialog box in detail, later). Therefore, we need to open this dialog box in order to enter the formula for the **Text2** field. There are four alternative ways to open the **Custom Fields** dialog box, as follows:
  - Clicking the **Custom Fields** button in the Properties group of the PROJECT tab on the Ribbon
  - Selecting the **Custom Fields** command in the shortcut menu opened by right-clicking any column header in a task or resource table
  - Hitting the keys **Alt**, **r** and **f** successively
  - The **Custom Fields** command button can be added to the Quick Access Toolbar

If you open the dialog box while Project's focus is set to a custom field in a task or resource table (that is, while a cell in a custom field's column is selected), then the dialog box is opened for that custom field; otherwise, it is opened for **Text1**.



Field
[✓ Task] Resource Project Type: Text

Field
Text1
Text2
Text3
Text4
Text5
Text6
Text7

Custom attributes
[✓ None] Lookup... Formula...

Calculation for task and group summary rows
[✓ None] Rollup: Use formula

Calculation for assignment rows
[✓ None] Roll down unless manually entered

Values to display
[✓ Data] Graphical Indicators...

- In the dialog box opened, you will see two option buttons representing two categories of the custom fields, in the **Field** section on the top of the dialog box; namely, **Task** and **Resource** (the **Project** option button has no use in the standalone desktop products).

  Project automatically selects the option button, according to the category of the project data focused on, in the active view; therefore, the **Task** option button will be automatically selected now.

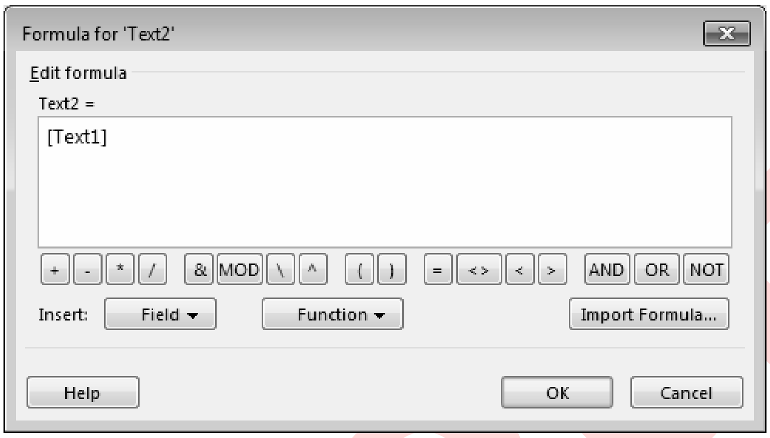
  Always verify that the currently selected option button corresponds to the category of the custom field for which you want to enter the formula, and if not, change it to the other one. In this example, we will work with the task custom fields, therefore we can keep the current selection of the option button.

  In the dialog box opened, you will also see that the **Text2** field has already been highlighted in the Field box of the **Field** section, for your convenience.

- The option **None** is selected by default in the **Custom attributes** section. Now click the **Formula...** button to open the **Formula** dialog box (there is no need to select the related option button first). The title of the dialog box shows Formula for 'Text2'. Here, any expression that you enter into the Edit formula box is referred to as a custom field formula.

Note that holding down the **Alt** key while pressing **r**, **f** and **u** successively opens the **Formula** dialog box quickly. The entry bar does not show the formula entered. We need to open the **Formula** dialog box again to review a formula entered.

- Enter the text **[Text1]** into the entry box as it is shown below. The field name enclosed in square brackets is called a field reference. You have just created a custom field formula containing a reference to a field now.



The **Field** button can also be used to insert a field reference into a formula. Project automatically attaches the enclosing brackets to the field name when the field reference is inserted through the **Field** button. And doing so also avoids the typing mistakes in the field name, and additionally, you will have a chance to cross-check the type of data which this field is expected to return to the formula by looking at the category name which the name of the field is placed under. For example, you will find the **Type** field under the Number category of the **Field** button while creating a resource custom field formula, since the **Type** field returns a numeric value representing the resource type. As a result, any reference to the **Type** field in a formula should be handled as a numeric value.

Note that the commands to cut, copy and paste the information can be used in the Edit formula box. While entering long formulas by typing in, do not hit <Enter> to continue with the next line in the Edit formula box. Instead, just keep typing in so that the entry automatically proceeds with the next line. A formula will constitute a single line, no matter how long it is. Sometimes, you may paste a long formula copied from a text editor into the Edit formula box. In this case, you do not need to worry about the spaces or the extra empty lines in the formula since Project automatically converts the entry to a single line formula and removes the empty lines and the extra spaces unless they are inside the double quotation marks.

Although it is not possible to change the font and the font size setting for the Edit formula box, we can always create the formula in a text editor and then copy/paste it.

- Click **OK** when you are done with entering the formula. Project now displays a dialog box with the warning message shown below:

  Existing data in the "Text2" field will be deleted because all values will now be calculated by the formula.
  To replace all data in the "Text2" field with the new calculated values, click OK.
  To return to the Formula dialog box, click Cancel.

The purpose of this message is to prevent you from accidentally overwriting the existing data in the custom field by the results from the formula. If this is the case, or if you decide to keep the existing data in the custom field, then clicking **Cancel** returns you to the **Formula** dialog box, so that you can click **Cancel** again in order to close the **Formula** dialog box. This action automatically empties the Edit formula box and keep the existing data in the custom field. Otherwise you can proceed with clicking **OK**.

It is important to note that the **Undo** command does not work, once the formula overwrites the existing data.

Project automatically turns the option button of the **Formula…** button on (i.e., the radio button) as soon as the **Formula** dialog box is closed. You can now select **None** to disable the formula but keep it in the Edit formula box; and then selecting the option button without directly clicking the **Formula…** button in order to enable the formula will trigger a dialog box with the following message:

Any existing data in the "Text2" field will be discarded as all values will now be calculated by the formula.
To enable the formula and replace all data in the "Text2" field with the calculated values, click OK.

Clicking **OK** in the dialog box will simply turn the option button on, thus enabling the formula entered previously. Note that clicking the **Formula…** button will bypass this warning message and open the **Formula** dialog box with the formula.

Project is not sensitive to case in formulas, except for the text enclosed in double quotation marks. Therefore entering, for example **[tEXt1]**, does not create a syntax error. Project automatically changes it to **[Text1]** while evaluating the formula, so you will see **[Text1]** when you re-open the formula box.

Note Be careful while editing formulas since the **Undo** command will not undo any changes that you have done in the formula. Therefore, always keep a copy of the previous formula somewhere else while making a major editing in the current one.

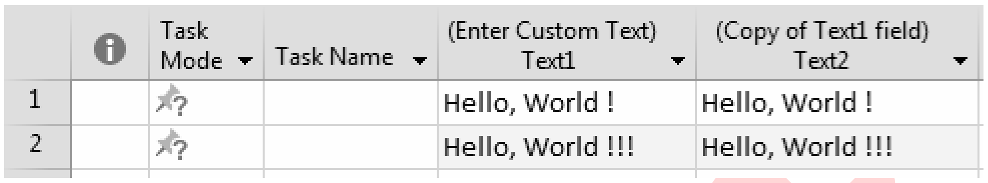
- Keep all the default settings in other parts of the dialog box. We will discuss the function of each control in detail, later. Click **OK** again to close the **Custom Fields** dialog box.

  Note that saving a formula is a two step process; first, closing the **Formula** dialog box, and then closing the **Custom Fields** dialog box. Project performs a syntax check in between those two steps; if the formula passes the syntax check, then Project accepts the formula.

- You are now back to the task table. The **Text2** field now displays the message that you have previously entered to the **Text1** field in the first row of the table; if so, then the formula is working.

  Note that Project does not show any indicator to tell us which custom fields in the table have formulas. When you click any cell in a custom field's column, the entry bar will be closed for entry, if the field contains a formula. And also the entry bar will display the information that it gets from the cell in gray color. Therefore, it will make working on a schedule easy, if you rename the custom fields in a way to give information on the calculated content. Keeping the mouse pointer over the column header of the renamed custom field opens a ScreenTip that shows the actual custom field name in parentheses (See the **ScreenTip style** box in **General** tab of the **Project Options** dialog box)

- In order to test the formula further, let us create another task in the project plan. Instead of entering a task name, just enter the text **Hello, World !!!** into the **Text1** field in the second row of the task table. Project will automatically create another task entry and the **Text2** field will also display the text **Hello, World !!!** as shown below:



	ⓘ	Task Mode	Task Name	(Enter Custom Text) Text1	(Copy of Text1 field) Text2
1				Hello, World !	Hello, World !
2				Hello, World !!!	Hello, World !!!

In demonstrations, instead of screen captures like the one above, we will use the following representation which hides the irrelevant details of the tasks or the resources:

Task table

Text1	Text2
Hello, World !	Hello, World !
Hello, World !!	Hello, World !!

Text1 field : *<enter text by typing in>*
Formula:
Text2 field : **[Text1]**

Note in the screen capture above that the column headers of the custom text fields have been renamed by using the **Field Settings** dialog box. These new titles will be visible only in that particular table.

You can rename the custom fields by using the **Rename** button in the **Custom Fields** dialog box. Renaming the custom fields by descriptive names provides some benefits as follows: the new name is recognized in the formulas (Project replaces the new name with the actual custom field name as soon as you save the formula), and it is easy to find the custom field name while scrolling through the names in the Field box of the **Custom Fields** dialog box, and also a custom name will remind you the purpose of the formula.

On the other hand, Project displays the name entered to the **Title** box of the **Field Settings** dialog box in the column header, even when you have already renamed the custom field with a different name in the **Rename Field** dialog box.

We will discuss all the controls in the **Custom Dialog** box later in detail.

Important Note In order to avoid destroying the project data accidentally while going back and forth between the dialog boxes, create new formulas in the backup copy of the original project plan file, and transfer them to the original file after having done all the tests.

Almost any long formula composed of nested expressions looks complicated without proper formatting. Creating, editing or reviewing a formula can be performed outside the **Formula** dialog box, for example, in a word processor, and then the formula can be copy/pasted back to the Edit formula box. Project automatically removes extra spaces and lines among the elements of the formula while saving by clicking OK if its syntax is correct. But there is an exception; the strings (that is, the characters enclosed in the double quotation marks) divided into multiple lines while editing a formula outside the **Formula** dialog box must be combined into a single line before copy/pasting the formula to the Edit formula box since it is not done automatically. And also it may sometimes be required to overwrite the quotes and dashes with the same characters in the Edit formula box in order to fix the character code problems causing syntax errors in the copy-pasted formulas.

Steps to Follow While Developing Formulas

The following six step procedure outlines the phases of developing a custom field formula. We have just discussed how to enter a simple formula in the previous section, but we have not yet covered any detail topic related to developing formulas. Therefore, if you want, you can skip this section for now and continue reading from the next section (that is, Fields in Project) and then return here after studying the whole book. This book presents all the information that you need to create custom field formulas by following these six steps.

Step #1: Determine whether you really need a formula

Before attempting to develop and use a formula, first perform several checks in order to decide whether you really need a formula in your particular case. Read the possible scenarios listed below:

- You may need to produce some new data from the current project data by using a custom field formula; in this case, review task and resource fields available in Project since the data that you want to produce may already exist in one of these fields. For example, you do not need to calculate the difference between the baseline start date and the current start date of a task to find the slippage amount in working days (based on the project calendar or task calendar) since there is a start variance field, **Start Variance** which contains this information. But you must use a formula to see the variance in calendars days.

- You may need to interpret the project data in a different way by reformatting the existing data; for this purpose, a conversion formula may be used, but first check Project's formatting capabilities since Project may already have a feature to format the data in the way that you want. For example, you may want to display the dates in a table in a format different than the default one. Before creating a formula to format dates, check all the date display formats available in Project (see the Date format box on **General** tab of the **Project Options** dialog box) since the format that you need might be one of the formats available.

- You may use a formula to perform tests on tasks or resources in order to identify the tasks or resources whose data satisfies certain conditions; then the results from the formula can be used for filtering or grouping purposes. Before doing this, always review the built-in task/resource filters and groups (and also see AutoFilter's filters and groups), since a filter or group that suits your needs may already be included in Project.

  But suppose that you want some graphical indicators that will alert you immediately when a certain task or resource data exceeds a predefined threshold value in a project plan. This cannot be achieved by a highlight filter since filters require user's action to trigger filtering operation. In this case, you must create a custom field formula which constantly watches and evaluates the related task or resource data and then displays the results by the graphical indicators defined.

Also note that, most of the time, you do not need a complex formula since graphical indicators can be defined for a custom field referencing the target field; in this case, conditions tested against the value of the target field in the indicator table do the same job as the test expressions in the formula.

At this point, suppose that you cannot find what you need in Project after having read all the paragraphs above, then proceed with Step #2.

Step #2: Define output from the formula

- What output do you want from the formula ? Is it a numeric value, text information, date and time information, a duration or work value, or a logical value such as Yes and No ?
- Then decide which custom field can properly hold and display the data returned from the formula. Always check whether the data returned from the formula needs to be converted to the type of data expected by the custom field. Most of the time, you want to apply some sort of conversion to the data returned from the formula in order to display it in a particular format in the custom field. You may also decide to return the output to a custom text field since it can display any type of data even without a data type conversion.
- Also check whether the data returned from the formula falls into the range of valid values accepted by the custom field selected.

  At this point, you are expected to know which custom field is going to be used to enter the formula.

Step #3: Define input to the formula

- Determine what data you need to process in the formula in order to obtain the result that you want; that is, which fields to reference, functions to reference, and literal data to use.

  At this point, you are expected to know all the elements of the formula.

Step #4: Determine which operations the expressions in the formula should perform on the information inputted (or passed)

- Decide whether you need to verify the data hold in the fields referenced before attempting to access the fields' data from the formulas.
- Decide whether you need to apply conversion to the data referenced; that is, literal data or the data returned from the fields or the functions. If you are not sure what type of data the field stores on the background or a function returns, then you can create a temporary formula referencing to that field or to the function in another custom field and see the result displayed, before using it in the formula.
- Determine how to combine all the elements of the formula in expressions. That is, determine what operations you need to perform on these elements that provide the data input to the formula in order to obtain the output desired.

  At this point, you can create the expressions required to built the formula. Note that you can use a pencil and paper or an application such as Notepad until you reach at the point where you are ready to enter the formula to the custom field.

Step #5: Test the formula's syntax and logic

- Enter the formula to the custom field. At this point, you will know whether the formula has been entered correctly or not (that is, there is no syntax error), if Project allows you to save the formula.
- Insert the custom field containing formula, and all the fields referenced in the formula to a test table.
- Test the formula by manually populating the fields that the formula references with the data representing all possible scenarios (you can temporarily substitute intermediate custom fields in place of the fields referenced). And verify the results obtained by comparing them with the expected ones.

  After having tested the formula with all possible data input scenarios, you now know that the formula logic produces accurate results.

Step #6: Document the formula

- Even if you are the only one using the formula, document how it works for future reference and maintain a version history describing all the modifications done on the formula.

  Documenting the formulas and keeping a record of the formula revisions will help you to remember your previous work on the formula when you need to modify it or fix an error occurred after having used the formula for a while. Also other users of the project plan file containing formulas may benefit from such documents when trying to understand how the formulas in the project plan file work.

Fields in Project

As it is seen in the example above, the custom field formulas operate on the project data that the fields hold. For this reason, first of all, we must understand what a field is in Project, in order to build accurate formulas.

What is a field ?

In Project, a field is a cell at the intersection of a row and a column in a table, a named box in a form, or any other location in a view, which contains the project data that is either input by the user or calculated by Project's scheduling engine for a task, resource or assignment.

> **Note** In this book, as it is seen in the paragraph above, a field has been defined as a project data entry and display area in the user interface (that is, a location in a view). Depending on the context, a field may have a different definition in terms of databases.

All the complex data management operations on the fields are handled by Project in the background, therefore a user does not need to worry about the operating system and the application level details regarding how the project data that the fields hold are stored into or retrieved from an mpp file residing in computer's hard disk or any other storage device.

What is a field reference in a formula ?

In the example above, you have just entered your first formula for a custom text field **Text2**, which copies the content of the **Text1** field to the **Text2** field. In Project, we use field references in the formulas when we need to pass the field values into the custom field formulas.

As required by Project's custom field formula syntax, a field name enclosed in square brackets in a formula, for example **[Text1]**, is called a field reference; that is, the **Text2** field's formula references the field **Text1**. Project evaluates the **Text2** field's formula for each task in the schedule, as soon as we click **OK** in the **Formula** dialog box. This means that the formula gets the data stored in the **Text1** field and returns to the **Text2** field for each task line in the project plan. Project keeps recalculating the formula for all the tasks as we continue to create new tasks and/or change the existing data in the **Text1** field.

Project recognizes a field reference in a formula when a single-word field name is not enclosed in brackets but it would be a good practice to use brackets for consistency and readability of the formula.

You cannot enter any information to the **Text2** field by typing in since the custom field now contains a formula. This feature can be used to implement a simple protection for the data entered. For example, in a task table, enter a reference to the **Name** field in a formula for a custom text field, then rename the custom text field as "Task Name"; and insert its column before or after the **Name** field. Finally, hide the **Name** column in the task table. The table will now display the task names in the custom text field but the custom text field will be closed for entry by typing in. This simple method prevents us from changing the content of the actual **Name** field accidentally. Note that we cannot lock the fields in Project for the same purpose.

Properties of a Field

In Project, a field is characterized by some properties such as its name, category, type and entry type. Each property is covered in detail in the following sections.

> **Note** Product help documentation provides a list and detailed descriptions of all the fields available in Project. Search the topic "available fields" on the website for the product help. Field description pages include category, field type and entry type information.
>
> Always review the list of available fields before attempting to create a custom field formula in order to produce custom data since the information needed might already be available in Project.

Field Names

A field's name is an identifier which is composed of one or more descriptive words associated with the project data that the field holds. Project allows us to rename the fields in the user interface. Renaming fields is a very useful feature, especially while working with the custom fields, as discussed earlier.

Field Categories

In Project, the fields are organized based on the six categories of the project data available as follows:

Task, Resource, Assignment,
Task-timephased, Resource-timephased, Assignment-timephased

A non-custom field may have one or more categories, depending on which categories of the related project data exists. For instance, the duration data is available only for the tasks in a project. Therefore, the **Duration** field has only task category and it is referred to as a task field. The **Cost** field has all six categories since the related cost data are available in all categories. As another example, there are only two categories of the **Percent (%) Complete** field, task and task-timephased categories, since the percent complete data does not exist in other categories.

A custom field may have only task, resource and assignment categories. It is important to note that, as it is clear from the option buttons available on the top of the **Custom Fields** dialog box, we can enter formulas only for the task and resource categories of the custom fields.

In order to have access to a custom field of the assignment category, we need to insert the custom field into a table in the Task or Resource Usage view; therefore, there are two subcategories of the assignment custom fields: task assignment and resource assignment categories. The custom fields in the assignment category are available to input data by either direct user entry or roll-down method (this topic will be discussed later) in Usage views, but we cannot define formulas for the custom fields of this category.

A custom field formula can only reference the fields (that is, the non-custom fields or the other custom fields) of its own category since cross-category field access is not possible from within a custom field formula. For instance, a task custom field formula cannot contain a field reference to a resource field or a resource custom field, although a few task fields contain resource information, such as **Resource Names**, **Resource Group** and **Resource Initials**.

Field Types

Project information is stored in the fields, in various forms of data such as date and time, number, text, currency and so on. A field type (or field's data type) identifies the type of the project data that the field contains, as the name implies. The content of a field may be different but the field type is identical across all categories of a field.

In order to create a formula that gives accurate results, we need to know both the type of the fields (non-custom or custom) referenced in the formula and the type of the custom field for which we create the formula. We will discuss why we need to know the field types while creating formulas in detail, later.

The field types for the non-custom fields are listed in the table below, along with the description of the field content and some examples of the fields for each type. As mentioned before, formulas can only be created for the task and resource custom fields, therefore a formula can only reference the task or resource fields. Consequently, we will focus on the non-custom fields of the task and resource categories.

Field Types for Non-Custom Fields

Field Type	Field's Content	Examples
Currency	Cost value	Actual Cost, Fixed Cost
Currency Rate	Rate of pay value	Standard Rate, Overtime Rate
Date	Date and time value	Scheduled Start, Deadline Scheduled Finish
Duration	A value representing a duration (span) of time	Work, Scheduled Duration, Leveling Delay, Start Variance, Free Slack, Total Slack
Enumerated	An item selected from a drop-down list of the predefined choices that the field contains	Accrue At, Task Calendar, Constraint Type, Status
Integer	A whole number	ID, Unique ID
Integer List	A list of whole numbers separated by the list separator character	Predecessors, Successors
Percentage	A percentage value	% Complete, % Work Complete
Percentage/ Number	A percentage value or a decimal number (depending on the setting)	Max Units, Peak
Text	Text information	Task Name, Duration, Start, Finish, Group, Code, Outline Number
Text List	A list of text items separated by the list separator character	Resource Names, Resource Initials, Resource Group
RTF	A text in Rich Text Format which allows formatting such as bulleted lists, bold typeface and so on	Notes
Yes/No	Yes or No value	Summary, Milestone

The fields display the project data in various formats as specified by the field type. For example, the **Cost** field adds a currency symbol to the value displayed, or the **Max Units** field includes a percentage symbol along with the value displayed, if the format is set to percentage. As another example, a duration value is displayed with a time unit.

On the other hand, it is important to understand how the values are stored internally in a field, rather than how they are displayed since we design our formulas based on what kind of information a reference to that particular field evaluates to, in a formula. Despite the fields display a project's data in many different formats, the references to these fields in formulas evaluate to either numeric or text (non-numeric data) as follows:

- References to the duration, integer, percentage and some enumerated type fields (for example, the **Status** field, the **Constraint Type** field) evaluate to whole numbers in formulas.
- References to text, text list, RTF, integer list and some enumerated type fields (for example, the **Task Calendar** field) evaluate to text data in formulas.
- References to the currency, currency rate and percentage/number (for example, the resource fields, **Max Units** and **Peak**) type fields evaluate to decimal numbers in formulas, with a precision up to two decimal places.

- The date field references evaluate to numeric values (decimal numbers with a precision up to 10 decimal places) in formulas but they are recognized as date and time values depending on the context in a formula.
- The Yes/No type fields (for example, the **Summary** and **Milestone** fields) are also referred to as flag type fields. References to flag fields in formulas evaluate to -1 for a flag field displaying Yes and 0 for a flag field displaying No, when converted to numeric values.

Note that some fields may store negative numeric values. In Project, there is a range of valid values associated with each field type. Project does not allow us to enter data that falls outside the range of valid values to any of the fields. Field type also gives us information on the valid format of the entries to a field. For example, we cannot enter a duration value with a time unit label which is not recognized by Project.

Note on the table above that, the **Duration**, **Start** and **Finish** fields are listed as text type fields since they can accept text information. Therefore, we cannot create filters with criteria which contain tests comparing the **Duration** field with the other duration type fields; and also the **Start** or **Finish** fields with the other date type fields, so we must use "scheduled" versions of those fields in the Field Name column of such filters. We may, of course, create a filter comparing the **Scheduled Duration** field with the **Duration** field, but not the one comparing the **Duration** field with the **Scheduled Duration** field.

In formulas, the **Duration**, **Start** and **Finish** fields will evaluate to the same values as the "scheduled" versions for automatically scheduled tasks; and the same also applies to the manually scheduled tasks only when they contain valid duration and date values. In this book, we will always work with automatically scheduled tasks, unless otherwise stated; therefore, we will use the **Duration**, **Start** and **Finish** fields. Also note that the field type specification for those fields are the same as the "scheduled" versions in the product documentation, so do not get confused over their field types. The entry type for the **Scheduled Duration**, **Scheduled Start** and **Scheduled Finish** fields of a manually scheduled task is "calculated", so they do not allow any input by typing in. We will discuss entry types in the next section.

It is important to note that the operations that can be performed on a field reference in a formula are determined based on the type of the value that the field reference evaluates to. For example, arithmetic operations cannot be performed on the text data unless it contains numeric characters that can be converted to numbers.

> **Note** In this book, we will discuss only the types of the fields that can be referenced in the custom field formulas; therefore, the field type **Indicator** is not listed above. Search for the topic "field types" on the product help pages, in order to see a complete list of the field types and the fields available for each type.

The custom field types available in Project are listed in the table below. Note that the **Custom Fields** dialog box groups all of the task custom fields in sets of 10, 20 or 30 fields under a type name; that is, each set is identified with a custom field type name (see the first column in the table below). The same grouping method applies to the resource custom fields as well. Therefore, in the **Custom Fields** dialog box, we have to select the associated custom field type in the **Type** box in order to get access to a particular custom field for entering a formula.

Field Types for Custom Fields

Type of the Custom Field (Type box)	Custom Fields available (Field box)	Field Type	Field's Content
Cost	Cost1 through Cost10	Currency	Cost value
Date	Date1 through Date10	Date	Date and time value
Start	Start1 through Start10	Date	A start or other date and time value
Finish	Finish1 through Finish10	Date	A finish or other date and time value
Duration	Duration1 through Duration10	Duration	Duration or work value
Number	Number1 through Number20	Number	A numeric value
Text	Text1 through Text30	Text	Any text information
Flag	Flag1 through Flag20	Yes/No	Yes or No value
Outline Code	Outline Code1 through Outline Code10	Outline code	A custom tag showing a hierarchy similar to WBS codes or outline numbers

We have already discussed the field types in the previous paragraphs. Note on the field type number in the table above; references to the number type fields (that is, the custom number fields) in formulas evaluate to decimal numbers, with a precision up to two decimal places. The ranges of valid values are also specified for the custom fields. Project does not allow us to enter or return data that falls outside the range of valid values to the custom fields. We will discuss the range and format of the valid values for each custom field type, later.

> **Note** Always check the limits of the custom fields while working with large values. Search the topic "Specifications" on the product support website for detailed information regarding the limits of the custom fields. A custom field may display #ERROR at the summary level, when the rolled-up sum of the subtask values exceeds the maximum value allowed in the range of valid values. In this case, consider changing the related option setting in the **Custom Fields** dialog box to **None**. If it suits your needs, you can also select another roll-up method or you can modify the formula for the summary rows and then select **Use formula**.

When we need to interpret a project's data in an alternative way, a custom outline code field can be used to create a custom hierarchical structure which can then be used to sort, filter or group the project's task or resource data in the way we want. Note that we cannot enter formulas to custom outline code fields, so the information presented on the table above would be helpful only when we reference a task or resource custom outline code field in a custom field formula. The custom outline code field references evaluate to text data in formulas. All the controls except for the **Lookup...** command will be grayed out in the other sections of the **Custom Fields** dialog box as soon as you select the "Outline Code" in the **Type** box of the **Field** section of the **Custom Fields** dialog box.

A field's name usually gives you an idea of field's data type (or its content). For example, it is obvious that the **Finish1** and the **Start** fields are both date type fields containing date and time information. On the other hand, it is better to verify data types by reviewing the field descriptions in the help articles, for both the fields referenced in a formula and the custom field for which the formula is created. For instance, the **Start Variance** field's data type might be confused with date. It is indeed a duration type field. As another example, the **Status** field and the **Task Calendar** field are both enumerated type fields but the **Status** field evaluates to a numeric value while the **Task Calendar** field gives text data in a formula.

What happens if a formula returns a value whose type does not match the custom field's type ? This is what we will intentionally do a lot in the examples. Most of the time, Project handles type conversions implicitly. At other times, we get a #ERROR in the custom field. In this case, performing an explicit type conversion on the return value may solve the problem. All these cases will be discussed later, in related sections.

In Project, the last column of any table is the Add New Column column by default. Project detects the data type of any information that you enter into any cell in this column by typing in and then automatically replaces the column with the first available custom field of the proper type as soon as you approve the entry. This helps you to see what the data type of the information entered is and which custom field is available to use. If you have already used up all the custom fields of a particular type and category, then Project will display the **Delete Custom Fields** dialog box. You can select some fields listed to delete; or click **Cancel** to close the dialog box and then either use a different type of field or reuse an existing field of the same type.

Entry Types

A field's entry type in a task or resource category can be one of the following types as mentioned in the field descriptions: **entered**, **calculated**, **calculated or entered** and **null**. Details for each type are as follows:

- Only the user can enter a value to or modify the current value in an **entered** field.
  Example: the task field **Physical % Complete**
- In a calculated field, only Project calculates the value, based on the values in the other fields. Project recalculates the value automatically when the schedule changes.
  Examples: the task field **Early Start**, the resource field **Actual Cost**
- The value of a **calculated or entered** field is calculated by Project based on the values in the other fields. Any user entry overrides the calculated value.
  Examples: the task fields **Duration** and **Actual Cost**
- A task or resource category of a field with entry type null, actually displays no information but it makes the assignment category of the field available in a Usage view. There are just a few fields with entry type null, so it is easy to remember those fields and avoid using them in formulas. They do not cause #ERROR but the formula will not get any data from these fields. And they are as follows: the resource category of the **Baseline Start**, **Baseline Finish**, **Actual Start**, **Actual Finish** and **Leveling Delay** fields, the task and resource categories of the **Cost Rate Table** field.

  The entry type null just indicates that the field has no value (i.e., empty) in that particular category. Therefore, for example, the numeric field **Leveling Delay** will always contain zero (0), the non-numeric field **Cost Rate Table** will always contain a zero-length string ("") and the date type field **Baseline Start** will always contain NA, in the categories with null entry type.

This might create a confusion, but note that, some null fields (e.g., the resource fields **Baseline Start** and **Cost Rate Table**) given above can still be found in the lists of the **Field** button. Therefore, we should always review the field descriptions and verify the entry types for the fields referenced in formulas.

As seen above, the field type is identical across all categories of a field but the entry type may vary. For instance, the entry type for the resource category of the **Baseline Start** field is null, therefore it always displays NA (NA means empty date field in Project) in the Resource Usage view. It is just a placeholder in the field's column and makes the assignment category of the field, which has the entry type **calculated or entered**, available in the Resource Usage view.

As another example, the task field **Resource Type** is just a placeholder, providing no information and the resource assignment category of this field displays information in the Task Usage view. In Resource Usage view, the resource field **Type** displays the type of the resource; on the other hand, its task assignment category automatically gets the rolled-down value from the resource category but its entry type is null so the dropdown button is grayed out.

The task field **Actual Cost** is calculated by Project by default but we can change the related settings in order to enter the values manually; see the help page describing the field for more information.

The entry type for all the task and resource custom fields is **calculated or entered** but a custom field containing a formula is closed for entering values by typing in; that is, the entry type becomes calculated.

As an exception, the entry type of the custom outline code fields is always **entered** since we cannot define formulas for them. A custom outline code field has no task assignment category; that is, the associated cells are blank and grayed out in a Resource Usage view. On the other hand, resource assignment category automatically gets the rolled-down lookup list from the task category, even though the option buttons below the section **Calculation for assignment rows** is inactive in the **Custom Fields** dialog box for the task custom outline code fields.

As explained above, the entry type of a field gives us information on how the field is populated. For example, Project always populates the fields with the entry type calculated with an initial value by default but it may not be a proper value to use in calculations. It is also possible that the field may later become empty when the schedule changes. An **entered** field will be empty initially and the initial value representing the "empty" depends on the field type and but it may not be used in calculations.

When the fields referenced in the formula are empty (i.e. blank) or do not contain proper values, the formula may not return accurate results or even worse, it may not generate an error message. Therefore, a formula should verify the condition of the fields referenced before using their values and should also warn us if the verification fails. In order to develop such a formula, we have to know a field's entry type and initial value.

Formulas in Project

In this section, we will start with a definition of a formula and then continue our discussion of the topic with the elements of a formula.

What is a Formula ?

In Project, any expression, entered into the formula box for a custom field, which evaluates to a valid value that can be displayed in the custom field is referred to as a formula. An expression evaluates to a single value whose type also designates type of the expression. As we will discuss later, both logical expressions (also referred to as Boolean expressions) and comparison expressions evaluate to logical values corresponding to either true or false; a string expression evaluates to a string value (that is, text information); a numeric expression evaluates to a numeric value that can be used in arithmetic calculations and a date expression returns a date value.

An expression consists of one or more of the following elements: field references, references to the built-in functions, literals and operators. For example, the simple formula created in the first example, is composed of a single expression; that is, a field reference. Field references have already been discussed in the previous section. We will now explore the other elements of an expression and the expression syntax.

Expression syntax is a predefined set of rules that you must follow while creating expressions. If you do not use the correct syntax, then Project cannot recognize the elements of a formula while evaluating the formula. A syntax error is generated if Project does not recognize an element of the formula; and Project does not proceed with the formula any further until you fix the problem. Syntax errors in formulas are easy to handle since they are detected immediately while saving the formula, that is, at the moment when Project evaluates the formula.

For example, consider a formula referencing a field which has a two word name such as **Actual Start**. Expression syntax requires that you must enclose the field names in square brackets in the field references. It will work with the fields having a single word name but you will get the following syntax error if you fail to enclose the field name **Actual Start** with square brackets in a formula:

The formula contains a syntax error or contains a reference to an unrecognized field or function name.
To return to the Formula dialog box and highlight the error, click OK.

When you click OK, Project will return you to the formula box and highlight the word "Actual" since it is not recognized as a field name. You will not be allowed to save the formula until you fix the syntax errors generated. A syntax error will, of course, be generated when you enter a field name that does not exist. Some other examples of formulas that would generate a syntax error:

Enter this:	**Project highlights this:**
Finish Duration	Duration
Finish, Duration	, (comma)

In the first formula, Project starts to evaluate the formula from left to right and recognizes the **Finish** field, but its relation to the **Duration** field is not defined; and as such, Project cannot determine how to evaluate the expression, therefore, highlights "Duration" which is not expected at this position. In the second one, the comma character is highlighted since it is not an operator, which is expected to define the relation between the field references.

It is obvious that the proper way to reference both fields in a formula is to build an expression performing some operations on the values that the fields evaluate to, in order to produce a result. We will discuss syntax errors and other type of errors that the custom field formulas generate in detail, later.

Literals

A literal refers to a value typed directly into an expression. Literal values of the different data types are used in expressions such as numeric literals, string literals, date literals and Boolean literals.

Numeric literals are the values of any numeric data type. For instance, the numbers 2 and 3.7 used in the arithmetic expression 2 + 3.7, are both numeric literals. Project recognizes scientific notation (i.e., a numeric value formatted as a combination of the numeric value, the letter "E" or "e", the positive or negative sign for the exponent and the exponent itself). For example, enter 5.0e-1 into the formula box for a custom number field; it will be displayed as 0.5 in a custom number field. When you open the **Formula** dialog box again, you will see that Project has already replaced the entry with 0.5 in the formula.

A string literal is any sequence of contiguous characters enclosed in double quotation marks. All the letters, numbers, spaces, or punctuation within the double quotation marks are interpreted as characters; e.g., "123abc". Note that a string literal is sometimes referred to as a string or a text string. Text information entered into a formula must be enclosed in double quotation marks, so that Project recognizes it as a literal value (i.e., a string literal). As an example; enter the string literal "this is a ""string"" value" into a custom text field's formula box. The field will display the message this is a "string" value as shown below.

Task table

Text1	Number1
this is a "string" value	24

Formulas:
Text1 field : **"this is a ""string"" value"**
Number1 field : **Len([Text1])**

Note that two double-quotation marks within the string literal are interpreted as single double-quotation mark. This is how we must use a double quotation mark within a string literal, otherwise we will get a syntax error while saving the formula. The **Len** function accepts a string as the parameter and returns its length; note that the **Len** function counts two double-quotation marks within the string literal as one while calculating the length of the string literal.

A date and/or time literal is any sequence of characters, representing a date and/or time in a valid format, enclosed in the number signs (#). Date and/or time literals can be entered in various formats, such as the formats specified by the system's locale setting, ISO 8601 format which is the international standard and so on. As an example, the date and time literal #02-Jan-2014 13:00# represents January 02, 2014 13:00:00. The missing time part will be 00:00:00 (midnight) by default in a date literal. The default is 00 for the seconds part, if not specified in a time literal, or in the time part of a date and time literal. The 24-hour notation is the default time format when there is no AM or PM designator in a time or a date and time literal.

Logical operations represented by the logical operators (see the next section) are performed according to the Boolean algebra which was introduced in 1854 by George Boole; and the result of these operations is either true or false, that is, a Boolean value. The words **True** and **False** are the Boolean literals representing these two Boolean values (that is, true and false) in the custom field formulas. The words **Yes** and **No** are also available to use in formulas and they are equivalent to **True** and **False**, respectively. In formulas, all these words are used without double quotation marks and case-insensitively.

The flag fields display Yes or No for logical results. Therefore, in formulas, we will use the literals **Yes** and **No** instead of **True** and **False** for a consistent look. Note that both **True** and **Yes** evaluate to -1, and both **No** and **False** evaluate to 0 (zero) when converted to numeric values in formulas, just like the references to the flag fields. On the other hand, it is best to use logical literals instead of their numeric equivalents in formulas in order to make formulas more readable.

Note The terms logical and Boolean are used interchangeably. In this book, we will use the terms logical literals and logical values in place of Boolean literals and Boolean values, respectively.

Operators

As you may have already noticed while entering an expression to the formula box, the **Formula** dialog box has buttons that can be used to insert the elements of an expression, such as the **Field** button and the **Function** button. And also there are buttons for the operators, that are placed just below the formula box, as shown below:

+
-
*
/
&
MOD
\
^
=
<>
<
>
AND
OR
NOT

An operator is a word, or a symbol composed of one or more characters, that denotes the operations to be performed on the expressions (that is, the operands). Operators are categorized based on the operations that they represent as follows: arithmetic, comparison, logical and concatenation operators.

Arithmetic Operators

The table below lists the arithmetic operators. In the table, the operands **expr**, **expr1** and **expr2**, each represents a numeric value or a numeric expression (that is, a subexpression).

Arithmetic Operators			
Symbol	**Name**	**Syntax**	**Operation represented**
+	Addition	expr1 + expr2	Sum the expressions
–	Subtraction	expr1 - expr2	Find the difference between the expressions
	Negation	-expr	Change the sign of expr from positive to negative
*	Multiplication	expr1 * expr2	Multiply the expressions
/	Division	expr1 / expr2	Divide expr1 by expr2
\	Integer division	expr1 \ expr2	Round both expressions to integers and then divide expr1 by expr2, truncate the result to an integer and return the integer quotient
Mod	Modulo or modulo division	expr1 mod expr2	Round both expressions to integers and then divide expr1 by expr2, return the remainder. The result will have the same sign as expr1
^	Exponentiation	expr1 ^ expr2	Raise the expression expr1 to the power of the exponent expr2

We are all familiar with the arithmetic operators such as the addition, subtraction, multiplication and division operator, and also the arithmetic operations denoted by those operators, but how are they implemented in the custom field formulas ? This is what we will discuss now, starting with the familiar one which is the addition operator.

The expression **expr1 + expr2** shown in the Syntax column of the table represents the expression syntax (that is, the format recognized by Project) for the addition operation. The plus sign ("+") is the addition operator, which is an arithmetic operator, that requires two operands (that is, **expr1** and **expr2**).

Operands to the addition operator can be literal values as in the arithmetic expression **6 + 9.2**, or numeric expressions, each of which evaluates to a numeric value. We will discuss various types of expressions in detail, later.

The plus sign specifies that an addition operation is to be performed on both expressions; namely, **expr1** and **expr2**. Project, or the related component of the application, evaluates the numeric expressions on both sides of the addition operator in a certain order, and then adds the two resulting values together, and returns the sum as the numeric result of the addition operation to the main expression (that is, to the point where the main expression contains the expression **expr1 + expr2** as a subexpression) or to the custom field containing only the expression **expr1 + expr2** as a formula.

Note that, in Project, a leading sign is not used while entering a formula. If you enter, for example, the expression **=6+9.2** into the formula box, then you will get a syntax error; Project will return you to the **Formula** dialog box and highlight the equal sign ("=") in the expression entered. Project will not allow you to save the formula until you remove the preceding equal sign.

The basic concepts and the mechanism described above, regarding the implementation of the addition operator in formulas, apply to the other operators as well. On the other hand, the details of the operations represented vary depending on the operator.

If an operand to an arithmetic operator in an expression is a string containing numbers as characters or a date value (or any expression evaluating to those values), then Project attempts to convert it to a numeric value before performing the arithmetic operation. If the conversion fails, then the formula containing the expression returns #ERROR. For example, the formula **"123"*2** returns 246 to a custom number field. This implicit conversion feature enables us to perform arithmetic operations on dates and times, as we will discuss later in the related section; for example, the formula **[Start]+1** adds 1 day to the start date. As we will discuss later, a number can be added to or subtracted from a date; and a date can be subtracted from a date but adding dates does not produce a meaningful result.

Some of the arithmetic operators are not used often in daily arithmetic, such as the integer division operator (that is, the backward slash character, "\"), the modulo operator, the negation operator and the exponentiation operator. The following table shows examples of the integer division operation:

Number1 field's formula		**Number1** field's value
31.5 \ 4.7	→	6
31.5 \ 4.5	→	8

The steps in evaluation of the integer division expressions **31.50\4.7** and **31.50\4.5** are as follows:

$$\begin{array}{lccccccc} & & (1) & & (2) & & (3) \\ 31.5 \backslash 4.7 & \rightarrow & 32 / 5 & \rightarrow & 6.4 & \rightarrow & 6 \\ 31.5 \backslash 4.5 & \rightarrow & 32 / 4 & \rightarrow & 8 & \rightarrow & 8 \end{array}$$

In these operations, first the operands (i.e., the dividend and the divisor) are rounded off to the nearest integers (1), then a regular division operation occurs (2), next the quotient is truncated to an integer by removing the fractional part (no rounding off is done); thus, the formula returns the integer quotient of the division to the **Number1** field (3).

The results shown reveal that rounding the operands to integers in an integer division operation is based on the round half to even algorithm. The algorithm is applied only when the fractional part is halfway between the two integers. Therefore, 4.5 is rounded down to 4 since 0.5 is the halfway value between two integers, 4 and 5, and 4 being even. And 4.7 is rounded up to 5, since it is the nearest integer. An integer division by 1 can also be used to verify the result of the rounding operation as in 4.7\1 or 4.5\1. We will discuss the round-half-to-even algorithm in detail, later.

The following table shows examples of the modulo division operation:

Number1 field's formula		**Number1** field's value
31.5 **Mod** 4.7	→	2
31.5 **Mod** 4.5	→	0

The steps for the arithmetic operation represented by the modulo operator are as follows:

		(1)		(2)		(3)
31.5 **Mod** 4.7	→	32 / 5	→	6.4	→	32 – 6 * 5 = 2
31.5 **Mod** 4.5	→	32 / 4	→	8	→	32 – 8 * 4 = 0

Project rounds the operands to integers by using the same algorithm as the integer division operator (1), performs the division operation as usual (2); and then calculates the remainder (3). Note that the calculation rounded dividend – (integer quotient) * rounded divisor yields the remainder. A modulo division operation performed on the whole numbers used as operands involves no rounding off. We will use the modulo operator in examples, as we progress through the topics.

The table below includes examples with the exponentiation operator:

Number1 field's formula		**Number1** field's value
2^-1	→	0.5
2^2.2	→	4.59

The expression 2^-1 evaluates to 0.5. This is an example of the negation operation as well. Here, 2 raised to the power of negative 1 is equal to 1/2, that is, 0.5. The exponent can be a decimal number as in 2^2.2 which yields 4.59.

Comparison Operators

As mentioned in the previous paragraphs, we use arithmetic operators to perform arithmetic operations on any two numeric expressions, in order to produce a numeric result that we need. Similarly, we use comparison operators to compare expressions in order to determine the relation between the two expressions (or operands). The following table lists the comparison operators:

Comparison Operators			
Symbol	**Name**	**Syntax**	**Value returned**
=	equal to	expr1 = expr2	True if expr1 is equal to expr2
<>	not equal to	expr1 <> expr2	True if expr1 is not equal to expr2
<	less than	expr1 < expr2	True if expr1 is less than expr2
>	greater than	expr1 > expr2	True if expr1 is greater than expr2
<=	less than or equal to	expr1 <= expr2	True if expr1 is less than or equal to expr2
>=	greater than or equal to	expr1 >= expr2	True if expr1 is greater than or equal to expr2

A comparison expression created by using a comparison operator (see the Syntax column in the table above) always evaluates to one of the two logical values, true and false. For example, the comparison expression **expr1 <> expr2**, testing for inequality returns true if **expr1** is not equal to **expr2**, as shown in the last column of the table, and returns false if they are equal.

In the example below, the **Flag2** field will display Yes only if there is an exact match between the text entered to the **Text1** field and the string literal "Hello, World !!". This is a typical string comparison:

Task table

Text1	Flag2
Hello, World !	No
Hello, World !!	Yes

Text1 field : *<enter text by typing in>*
Formula:
Flag2 field : **[Text1] = “Hello, World !!”**

And more examples:

Condition tested		Flag1 field’s formula		Flag1 field’s value
Is “test” equal to “test” ?	→	**“test” = “test”**	→	Yes
Is 99 greater than or equal to 2 ?	→	**99 >= 2**	→	Yes
Is -1 greater than 0 ?	→	**-1 > 0**	→	No

As it is seen in the examples above, the operands, **expr1** and **expr2**, can be literal values (i.e., numbers, strings or dates and/or times), numeric expressions, string expressions or date expressions. The question is now: how does Project evaluate the comparison expression, if the types of the operands do not match ?

We will discuss comparing different types of fields and using the type conversion functions to convert types explicitly, later. Here, let us first explore how the comparisons with different types of literals work. There is of course no practical use of comparisons, where both operands are literals, but here, we will compare literals of different types in order to understand how the comparisons work. Common scenarios are as follows:

- Comparing a string literal with a numeric literal in a formula, as in **“9” = 9**, produces #ERROR even when the string literal contains only numbers as characters.
- A date literal can be compared with a numeric literal. In this case, a numeric comparison is performed. For example, both comparisons below return true since the numeric literals correspond to the serial numbers for the date literals (system’s short date format is M/d/yyyy):

  #11/11/2014#=41954 and **#12/11/2014#=41954+1**
- While comparing a date literal to a string literal, a string comparison is performed. This means that if the string operand does not contain the exact representation of the date operand’s date and time value according to the system’s locale (that is, the short date and long time format specified in the system), the comparison returns false.

  For example, the comparison **#11/11/14#="11/11/14"** returns false. Project changes the date literal **#11/11/14#** to **#11/11/2014#** (that is, the current system’s short date display format: M/d/yyyy) in the formula. Using this hint, we can modify the comparison as **#11/11/14#="11/11/2014"** and then the result becomes true. As a result, a string representing a date can be used as a date string if it contains a date in the system’s short date format. As we will discuss later, many functions accepting dates as parameters recognize date strings in various formats as long as the order of the elements is correct according to the system’s locale.

Logical Operators

Most of the time, we create formulas to find out whether the values in a field meet multiple conditions; for example, searching for the dates in a date type field that fall into a particular range of project dates. In this case, we use logical operators (or Boolean operators) to combine comparison expressions in order to create multiple conditions. The resulting expression representing multiple conditions is referred to as a logical expression.

Logical operators operate on logical values as operands. Therefore, the operands **expr**, **expr1**, **expr2** can be any expression that evaluates to a logical value, such as comparison expressions.

The **Formula** dialog box contains buttons for the logical **AND**, **OR**, and **NOT** operators. The table below lists the logical operators.

Logical Operators

Symbol	Name	Syntax	Value returned
And	Logical conjunction	expr1 and expr2	True when both expr1 and expr2 are true
Or	Logical disjunction	expr1 or expr2	True when either expr1 or expr2 is true
Not	Logical negation	not expr	True when expr is not true (that is, reversing the result of expr)

The references to the flag type fields (i.e., Yes/No fields) such as **Summary**, **Milestone** and **Flag1** evaluate to the logical values in formulas. Therefore, comparing a flag field's value with a logical literal for equality or inequality is redundant, as in **[Summary] = Yes** and **[Summary] <> Yes**; instead, the field reference **[Summary]** and the logical expression **Not [Summary]** can be used, respectively.

We will continue to discuss comparisons and logical operations in other sections as well, since they are used a lot in the custom field formulas.

Concatenation Operators

The concatenation operators **&** and **+** (that is, the ampersand sign and the plus sign) are used to combine string expressions. See the examples below:

Text1 field's formula		**Text1** field's value
“Hello, ” & “World !”	→	Hello, World !
123 & “.” & 33	→	123.33
123 & "4" & 33	→	123433
123 & 4 & 33	→	123433

The concatenation operator **&** (the ampersand sign) combines two strings to form one string; and note that any numeric operand is converted to a string in the operation. Thus, a numeric value (or any expression returning a numeric value) can be concatenated with an empty string (“”) in order to convert it to a string as in **123 & “”**.

The concatenation operator **+** (the plus sign) also combines the string operands successfully. But note that if any of the operands is a numeric expression, then the other operand is automatically converted to a number and an arithmetic addition operation is performed. Therefore, the other operand must be a string literal containing only numbers as characters, such as “123”, otherwise the result will be an error instead of the arithmetic sum of the two operands. For example, the expression 123+".567" returns 123.57 to a custom number field (that is, the arithmetic sum of 123 and 0.567 rounded to the field's allowed precision) while the expression 123+",567" generates #ERROR, since the string operand ",567" cannot be converted to a number during evaluation. Here, the comma is not recognized as part of a numeric value, based on the decimal symbol setting of the computer system used.

Note that Project does not have a delimiter symbol setting, so it is obtained from the operating system. Therefore, the delimiter character that appears between any two predecessor ids (that is, the task identification numbers) on the **Predecessors** field in a task table reveals your system's delimiter symbol setting. In a system where the decimal separator symbol is comma (,) and the delimiter symbol is the semicolon ";", for example, both **3,9 & ""** and **3.9 & ""** return **3,9**; Project replaces period in the decimal literal 3.9 with comma, in the formula. On the other hand, in a system where the decimal separator symbol is period (.) and the delimiter symbol is the semi colon ";", the concatenation operation **3,9 & ""** will cause syntax error, as it is seen above; in this case, we must use the string literal **“3,9”** if we need to use the value in a string expression, such as **“the value is “ & “3,9”**.

Operator Precedence

In previous paragraphs, we have used examples of single operator expressions while discussing various categories of operators, but formulas usually contain multiple expressions combined with operators. How does Project determine the order of evaluation in complex expressions ? It is determined according to the precedence and associativity of the operators, as follows:

- Expressions with operators of higher precedence are evaluated first.
- If the operators have the equal precedence, then the order of evaluation is determined by the associativity of the operators.

The table below lists operators in descending order of precedence; that is, the arithmetic exponentiation operator (**^**) has the highest precedence and the logical disjunction operator (**Or**) has the lowest precedence.

Precedence and Associativity	
Precedence (descending)	**Associativity**
Arithmetic Operators	
^	-
– (negation)	-
*, /	left to right
\	-
Mod	-
+, –	left to right
Concatenation Operator	
&	-
Comparison Operators	
All comparison operators	left to right
Logical Operators	
Not	-
And	-
Or	-

Note that some rows in the table contain more than one operator. These are the operators of equal precedence; namely, multiplication and division, addition and subtraction, and all comparison operators. For example, if an expression contains more than one comparison operator, then Project evaluates the expression from left to right as determined by the associativity of the comparison operators.

Here are some examples with numeric literals that show how order of evaluation is determined based on the precedence and associativity of the operators:

1 + 4 * 3 - 2 / 2 → 1 + **12** – 2 / 2 → **13** – 2 / 2 → 13 - **1** → **12**

1 * 2 / 1 * 3 → **2** / 1 * 3 → **2** * 3 → **6**

Note that the multiplication and division operators have the same precedence but it is higher than that of the addition and subtraction operators.

The **Formula** dialog box has buttons for parentheses. The parentheses override the default order of evaluation, therefore they can be used to control the order of evaluation in expressions.

+	**-**	*****	**/**	**&**	**MOD**	\	**^**	**(**	**)**	**=**	**<>**	**<**	**>**	**AND**	**OR**	**NOT**		

An expression enclosed in parentheses is given the highest precedence. The precedence and associativity rules also apply to the expressions enclosed in parentheses.

1 + 4 * (3 - 2 / 2) → 1 + 4 * (3 – **1**) → 1 + 4 * **2** → 1 + **8** → **9**
1 * (2 / 1) * 3 → 1 * **2** * 3 → **2** * 3 → **6**

Note that in the second example, there is no effect of using parentheses since multiplication and division have equal precedence.

As another example; the expression **"1"+"1"+1** is evaluated from left to right, so the concatenation **"1"+"1"** yields **"11"**, then the final expression **"11" + 1** is an arithmetic addition operation yielding the numeric value 12. Here, using parentheses changes the order of evaluation; for example, in the expression **"1"+("1"+1)**, the subexpression **("1"+1)** evaluates to 2 since the string literal "1" is converted to the numeric value 1, and then the final expression **"1" + 2** is an arithmetic addition expression where the numeric value 2 and the string literal "1" (which is automatically converted to number) are added together, thus giving the result 3. In order to make things clear, use the ampersand (&) to perform a string concatenation instead of the plus (+).

We sometimes use parentheses to emphasize the order of evaluation. Consider the arithmetic expression, 2^-1. In this expression, the precedence rules are in place to provide the proper order of evaluation, so we do not need parentheses, but we may want to use them in order to improve readability, as in 2^(-1).

Expressions can be nested by using parentheses. In case there are multiple levels of nesting, the evaluation starts with the innermost expression and then continues outward. There is a certain limit in how deep we can nest the expressions by using the parentheses, and exceeding this limit in the nesting level will result in a syntax error. In Project, this limit is probably high enough not to pose a problem in formulas. Nesting deeper than a few level makes a formula difficult to read, especially when the expressions wrap in the **Formula** dialog box. Project will warn us with a syntax error if there are mismatched parentheses in a formula.

Settings for the Custom Fields

The **Custom Fields** dialog box contains all the settings and commands related to the custom fields. We will now explore these settings and commands grouped under 5 different sections of the dialog box.

We have already discussed some parts of the **Field** section such as the **Task** and **Resource** options, the **Type** selection box, the Field box and the **Rename** command. The **Add Field to Enterprise…** command is not active in the desktop version of Project, just like the Project option at the top of the section. Also the **Formula** dialog box in the **Custom attributes** section has already been covered in the previous parts of the book.

> **Note** Use caution while working on the **Custom Fields** dialog box since changing some settings or applying some commands might result in loss of the existing custom field data.

How to Delete or Disable a Formula, How to Clear or Protect a Custom Field's Data

Selecting a field in the Field box and clicking the **Delete** button will delete the field's custom name and its lookup table, formula or graphical indicators defined; all the sections will be returned to the defaults. Note that there will be no Undo for this operation. The same operation can also be performed on the **Fields** tab of the **Organizer** dialog box (the custom field will not appear in the pane of its project plan file if it does not have a lookup table or formula, and/or graphical indicators defined). But this operation does not delete the data that the field currently holds.

The data in a custom field that has no formula can be deleted in a table, as follows: select and right-click the cells containing the data and then select **Clear Contents** command in the shortcut menu opened. Another method is to enter **Null** as the formula (that will generate #ERROR), and select **None** (that will clear #ERROR) in the **Custom attributes** section and then click **Delete** to remove **Null** in the Edit formula box.

Note that selecting the **Data Type** command on the shortcut menu opened by right-clicking the column header of a custom field shows a list of custom field types which is identical to the list of the **Type** button in the **Custom Fields** dialog box, except for the Outline Code type. The list shows the type of the current custom field highlighted and check-marked. Now selecting a different custom field type from the list, hides the column of the current custom field, deletes all its data if it does not have a formula and inserts a new custom field of the selected type to the table. The **Undo** command will undo the data type change command.

The **None** option is the default selection in the **Custom attributes** section, meaning the custom field is available for data entry by typing in. The **Lookup…** button in the same row is used to define a lookup table for a custom field. In this book, we will not discuss how to create lookup tables for the custom fields, but we may use custom fields with lookup tables in the examples.

In order to disable a formula temporarily, just select **None** option in the **Custom attributes** section; the custom field will now allow data entry. Then all the existing data in the field will be overwritten by the output from the formula, if you select the **Formula...** option back. Project will warn you before doing this.

If you want to protect the data entered to a custom field from being overwritten accidentally by typing in, then select the **Formula...** option and click OK in the dialog box that Project displays for warning, but do not click the **Formula...** button. Project will not allow any data entry as if the custom field has a formula.

How to Import a Custom Field

Copy-pasting the field data does not bring in the custom field's attributes and/or graphical indicators. The **Import Custom Field** dialog box is used for this purpose.

The **Import Field...** button in the **Field** section opens the **Import Custom Field** dialog box. This dialog box imports lookup table or formula, and/or graphical indicators of the custom field (selected in the Field box) of any project plan or template (selected in the Project box) to the custom field highlighted in the Field box of the **Custom Fields** dialog box. Note that the **Field type** option buttons allows us to select either Task or Resource field, independent of the **Task** or **Resource** category selected on the top part of the **Field** section in the **Custom Fields** dialog box.

We cannot select a project plan that is not currently open since it will not appear in the drop-down list of the Project box. Note that the import operation may be performed between the custom fields of the same project plan file; for example, the lookup table or formula and/or graphical indicators of the **Text1** field can be imported to the **Text2** field in the same project plan. If you copy the custom field that has a lookup table or formula, and/or graphical indicators to the Global.mpt, then the corresponding custom field of all new project plan files that does not have any of these elements defined will automatically get the attributes and/or graphical indicators defined, even without you noticing.

The **Import Formula** dialog box opened from the **Formula** dialog box can be used to import only the formulas; and the **Import Indicator Criteria** dialog box opened from the **Graphical Indicators** dialog box can be used to import only the indicator criteria. Both import operations can be performed between the custom fields of the current project plan file, or between the custom fields of the current project plan file and another file (or template) currently open.

Custom Fields in Summary Rows

We will now explore how the custom fields are populated in the task and the group summary rows (that is, the group headers), but let us first see how it works with the regular fields (i.e., non-custom fields).

At the task summary level, some regular fields contain the calculated data (e.g., the **Duration** field) while the others contain the data rolled-up from the non-summary task rows based on a predefined method (e.g., the **Start** and **Cost** fields), or the data entered by the user only (e.g., the **Fixed Cost** field). Although some regular fields such as **Duration** and **Start**, also accept user input at the summary level, others may be closed for entering data such as the **Cost** field.

Project calculates the data shown in the group headers for the regular fields at the time of grouping the data and we cannot customize how Project calculates that data. The data shown for the regular fields in the group header disappear as soon as the grouping is turned off. The group headers are not the user input areas that we can enter data to the regular fields. Other than that, Project may not calculate the values for the regular fields at the group summary level in the same way as it calculates them at the task summary level. For example, the **Fixed Cost** field shows the rolled-up sum of the group row values in the group header. And as we would expect, some regular fields show no data at the group summary level, such as the **ID** field (unlike the summary tasks, the group headers have no ID number) and the **Standard Rate** field. Let us now see how Project handles custom fields in summary rows.

The custom fields can be filled in with data by three methods at the task summary level (including the project summary level); user input, rolling up the non-summary task values (that is, values of subtasks including normal tasks, milestones and recurring tasks) and using a formula. The following table lists the related settings in the **Custom Fields** dialog box for the task summary rows.

	Options available to select in Calculation for task and group summary rows		
Option selected in Custom attributes	**None (default)**	**Rollup**	**Use formula**
None (default) *User enters non-summary values*	*User enters summary values*	*Non-summary values are rolled up*	**N/A**
Formula *Formula calculates non-summary values*			*Formula calculates summary values*
Lookup *User selects non-summary values from a lookup table*	*User selects summary values from a lookup table*	**N/A**	**N/A**

Note that the option selected in the **Custom attributes** section specifies how the custom field gets its data at the non-summary level and also determines what options will be available in the next section **Calculation for task and group summary rows**. For example, as shown in the table, if there is a lookup table defined, then both the **Rollup** and the **Use formula** options will be grayed out. And also, the **Use formula** option will be grayed out, if there is no formula defined (that is, **None** selected in the **Custom attributes** section). Project locks the custom field at the summary level when the **Rollup** or the **Use formula** options is selected.

The task (that is, non-summary and summary task data if the group definition includes) or resource data in a table can be grouped based on a regular or custom field, or any combination of those fields in a grouping hierarchy (the **Group Definition** dialog box allows us to enter up to 10 rows of field names). The table below summarizes how the custom fields are populated in the group summary rows.

	Options available to select in Calculation for task and group summary rows		
Option selected in Custom attributes	**None (default)**	**Rollup**	**Use formula**
None (default)	*Custom field is blank at group summary rows. Custom flag field shows No by default.*	*Custom field's values in group rows are rolled up to group summary rows*	**N/A**
Formula			*Formula calculates custom field's values at group summary rows*
Lookup		**N/A**	**N/A**

Like the regular fields, Project calculates the data shown in the group headers for the custom fields at the time of grouping the data and we can specify how Project calculates that data. The data shown for the custom fields in the group header disappear as soon as the grouping is turned off. The group headers are not the user input areas that we can enter data to the custom fields.

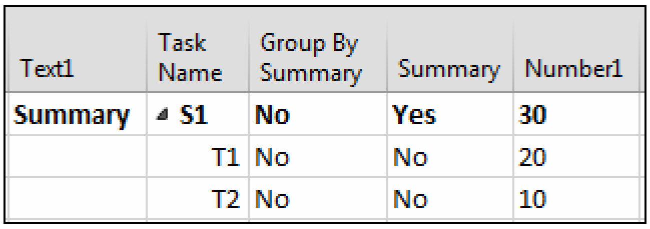
Group rows under the group headers contain grouped task or resource custom field data which is entered by the user, or calculated by a formula, or selected by the user from a lookup table defined, as it is specified by any of the options **None**, **Formula** and **Lookup** selected in **Custom attributes** section of the **Custom Fields** dialog box (see the first column of the table above). If a custom field's value in any group row is changed by entering a new value or selecting another item from the lookup or modifying the formula entered, then Project updates the rolled-up group summary row value of the custom field, but it does not rebuild the grouping based on the new data. It is required to turn off and on the grouping in order to refresh the grouping.

As mentioned above, we can specify how Project calculates a custom field's data in the group headers. In the grouping displayed, we can keep a custom field blank in the group summary rows or we can select a rollup method that specifies how the grouped values of the customs field to be rolled up to the group summary rows. These two actions correspond to the **None** and the **Rollup** options, respectively, in the section **Calculation for task and group summary rows** (see the second and the third columns of the table above).

Suppose that a custom field has a formula, and we want to use the same formula to calculate the custom field's value in the group headers, instead of rolling up the values of the custom field in the group rows to the group headers. Then we can select the option **Use formula** for this custom field in the section **Calculation for task and group summary rows**. In this case, the custom field formula can access the data calculated for the regular fields and the other custom fields in the group summary rows.

Consider the simple task table below where the **Number1** field displays 30 at the summary level which is the rolled-up sum of the values 10 and 20 entered for the tasks, and the **Text1** field contains the formula (select **Use formula**);

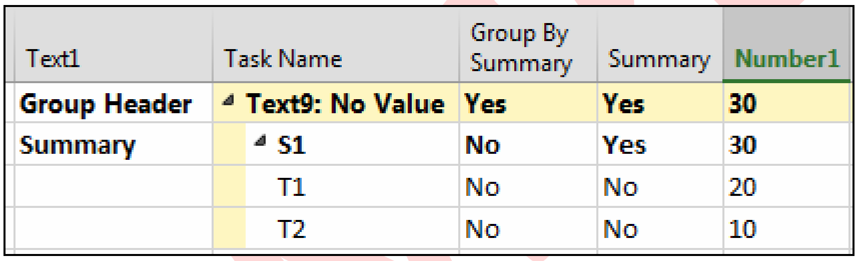
Switch([Group By Summary],"Group Header",[Summary],"Summary",True,"")



Text1	Task Name	Group By Summary	Summary	Number1
Summary	◢ **S1**	**No**	**Yes**	**30**
	T1	No	No	20
	T2	No	No	10

In the formula, the flag field **Group By Summary** returns true to the formula (and displays Yes when inserted to the table) in the group summary rows. The **Summary** field cannot distinguish the group summary row from the task summary row, and it returns true for both, therefore, it is placed as the second pair of the expressions in the **Switch** function.

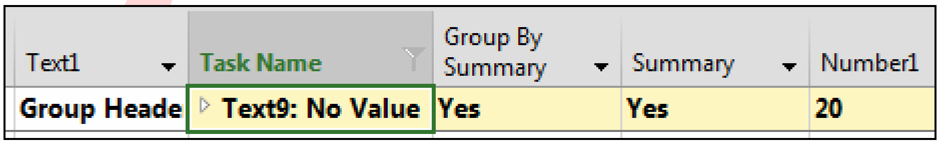
Let us now group the task data based on the blank field **Text9** and also turn on the checkbox **Show summary tasks** while defining the group. The picture below shows the task table with grouping.



Text1	Task Name	Group By Summary	Summary	Number1
Group Header	◢ **Text9: No Value**	**Yes**	**Yes**	**30**
Summary	◢ **S1**	**No**	**Yes**	**30**
	T1	No	No	20
	T2	No	No	10

As we would expect, the **Number1** field's value in the group summary row is the rolled-up sum of the values of the subtasks T1 and T2 (that is, 20 + 10 = 30) even though the group rows include the summary task S1. This simple example demonstrates how grouping handles rolling-up the values in the group rows to the group headers.

It is important to note that Project does not include the data of the rows filtered out while calculating the values at the group headers, as we would expect. For example, try this; collapse the group to the header row, **Number1** shows 30 and then apply AutoFilter to the **Name** column to filter out T2; **Number1** will now display 20 as it is seen below:



Text1	Task Name	Group By Summary	Summary	Number1
Group Heade	▷ **Text9: No Value**	**Yes**	**Yes**	**20**

Note that grouping also works in some other views with no table such as the Network Diagram view and the Team Planner view. The tables given above, which list the options available for the task and the group summary rows also apply to the groups created in those views.

The AutoFilter menu opened by right-clicking any field's column header in a table, provides **Group on this field** or **Group by** commands. Other than that, the **Group Definition** dialog box provides several other options while defining a group.

Rollup Methods

We will now discuss how Project rolls up the custom field data to the task and group summary rows, in detail. No rollup method is available for the custom text fields. The rollup methods available for the other custom fields are as follows:

- There are two rollup methods for the custom flag fields: **And** and **Or**:

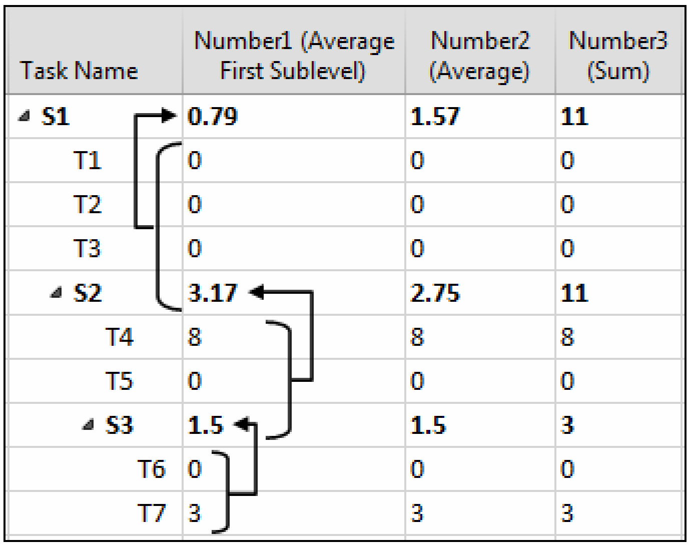
  If the **And** method is selected, then we will not get a Yes at the summary level unless all the subtask level data are set to Yes in the custom flag field (that is, the result will be the same as if the **And** operator has been applied to all subtask data in a formula).

  If the **Or** method is selected, then it is sufficient to have just one Yes among the rolled-up data (that is, at the subtask level) in order to have Yes at the summary level (i.e., the result will be the same as if the **Or** operator has been applied to all subtask data in a formula).

- There are two rollup methods for the custom date fields: **Maximum** and **Minimum**. The **Maximum** method is used to roll-up the latest date among the subtask dates to the summary level. And the **Minimum** method is used to roll-up the earliest date among the subtask dates to the summary level.

  These two methods are also available to select for the other custom fields holding numeric data, that is, the custom cost, number and duration fields.

- The rollup methods **Average**, **Average First Sublevel** and **Sum** can be applied to the custom cost, number and duration fields. The example below shows how these methods work with the summary tasks. Note that the behavior is the same with the grouped tasks too.

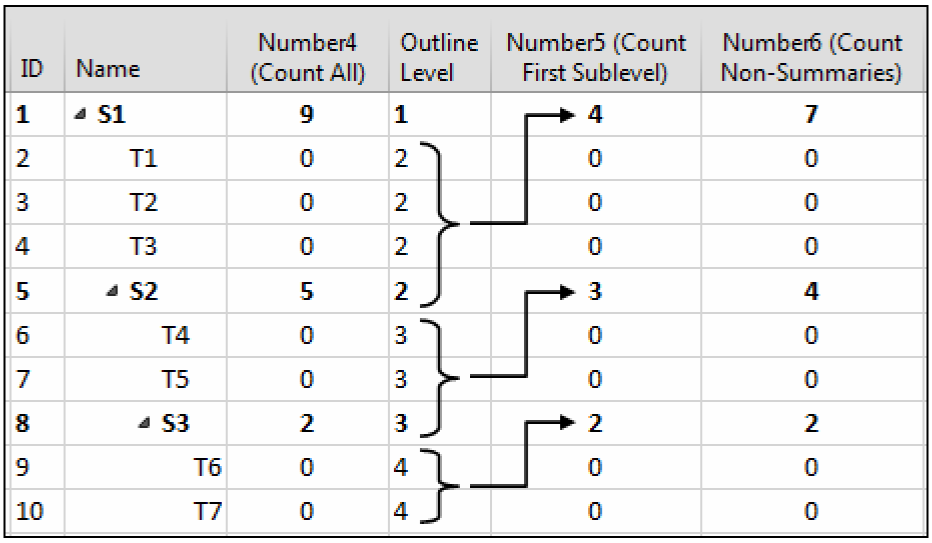


Task Name	Number1 (Average First Sublevel)	Number2 (Average)	Number3 (Sum)
S1	0.79	1.57	11
T1	0	0	0
T2	0	0	0
T3	0	0	0
S2	3.17	2.75	11
T4	8	8	8
T5	0	0	0
S3	1.5	1.5	3
T6	0	0	0
T7	3	3	3

How are they calculated:

Number1:	S3: 3 / 2	→ 1.5	(average of T6 and T7)
	S2: (8 + 1.5) / 3	→ 3.17	(average of T4, T5 and S3)
	Here, S3 is on the first level of the subtasks.		
	S1: 3.17 / 4	→ 0.79	(average of T1, T2, T3 and S2)
	Here, S2 is on the first level of the subtasks.		
Number2:	S3: 3 / 2	→ 1.5	(average of T6 and T7)
	S2: (8 + 3) / 4	→ 2.75	(average of T4, T5, T6 and T7)
	S1: (8 + 3) / 7	→ 1.57	(average of T1, T2, T3, T4, T5, T6 and T7)
	The summary level result is the average of all subtasks.		
Number2:	S3: 3	→ 3	(sum of T6 and T7)
	S2: 8 + 3	→ 11	(sum of T4, T5, T6 and T7)
	S1: 8 + 3	→ 11	(sum of T1, T2, T3, T4, T5, T6 and T7)
	The summary level result is the sum of all subtasks.		

- The rollup methods **Count All**, **Count First Sublevel** and **Count Non-Summaries** can be applied to the custom number fields. The example below shows how these methods work with the summary tasks. Note that the behavior is the same with the grouped tasks too.



ID	Name	Number4 (Count All)	Outline Level	Number5 (Count First Sublevel)	Number6 (Count Non-Summaries)
1	**S1**	**9**	**1**	**4**	**7**
2	T1	0	2	0	0
3	T2	0	2	0	0
4	T3	0	2	0	0
5	**S2**	**5**	**2**	**3**	**4**
6	T4	0	3	0	0
7	T5	0	3	0	0
8	**S3**	**2**	**3**	**2**	**2**
9	T6	0	4	0	0
10	T7	0	4	0	0

Note that the method with First Sublevel is used to count all the sub-items on the first sublevel including the non-summary and summary tasks.

The table below lists the custom field types and the corresponding rollup methods available:

	Custom Field Type				
Rollup Method	**Number**	**Cost**	**Duration**	**Date**	**Flag**
Average	●	●	●		
Average First Sublevel	●	●	●		
Sum	●	●	●		
Maximum	●	●	●	●	
Minimum	●	●	●	●	
Count All	●				
Count First Sublevel	●				
Count Non-Summaries	●				
And					●
Or					●

When you click the **Rollup** option button, the rollup box initially shows **Or** for the custom flag fields and **Maximum** for the other numeric custom fields. Note that there is no rollup options available for the custom text and outline code fields.

Graphical Indicators

In the **Custom Fields** dialog box, all the sections initially have the **None** option selected by default, except for the **Values to display** section which has the **Data** option selected by default. If you select the option **Graphical Indicators…** and click its button, Project opens the **Graphical Indicators** dialog box for the custom field highlighted in the Field box. If you want to display graphical symbols instead of the literal data in a custom field (except for the **Outline Code** field), then follow the steps below in order to specify indicator criteria:

- The dialog box opens with **Nonsummary rows** selected by default in the Indicator criteria for section, so you can now specify the indicator criteria for subtasks and group rows in the the Indicator criteria table. Otherwise, leave the table blank and proceed with selecting the next item, which is either **Summary rows** or **Project summary**.
- Select **Summary rows** and then enter the criteria for task summary and group summary rows to the table.
  If you do not need a different set of criteria for the summary level, then you can turn on the checkbox just below **Summary rows**. Project copies the criteria from **Nonsummary rows** as soon as you click Yes in the warning dialog box displayed. The table shows the criteria inherited in gray.
- Select **Project summary** and then enter the indicator criteria to the table for the project summary task. You can turn on the checkbox below **Project summary** in order to use the same criteria as specified for **Summary rows**.

In the Indicator criteria table, criterion defined in each row represents the condition to be tested on the custom field's data and the image to be displayed when the condition is met. The first column specifies the test, that is, how the custom field's data is to be compared against the value(s) entered to or the field reference selected in the Value(s) column. If the criterion entered is not valid, then Project alerts us and does not allow us to close the dialog box; for example, text information cannot be entered to the Value(s) column if the indicators are defined for a custom number field.

Project evaluates the formula and then starts evaluating the conditions in the Indicator criteria table, starting from the top row. If a test fails, then Project proceeds with the next row below, and keeps evaluating the condition on every row until it finds the condition that is satisfied. When it finds the true condition, then the corresponding indicator image is displayed in the custom field. But if all the tests fail, then the custom field shows blank (which is the default image being the first item in the image drop-down list) instead of the data. It is possible to handle the "no match" situation by using the test "is any value" as the last criterion line.

Test expressions are the same as the ones available to use in the **Filter Definition** dialog box, except for the last item, "is any value". We will discuss the tests in detail later. Here, we will focus on how to establish the logic while defining indicator criteria.

Consider a scenario where we have some numbers in the fields **Number1** and **Number2** and we want to monitor how the **Number2** field's data changes relative to the data stored in the **Number1** field. Suppose that the indicator criteria are given as follows:

- Based on the difference **Number2** - **Number1**;

Use green indicator,	if the difference is less than or equal to 7 percent of **Number1**
Use yellow indicator,	if it is greater than 7 percent of **Number1** and if it is less than or equal to 15 percent of **Number1**
Use red indicator,	if it is greater than 15 percent of **Number1**

- **Number2** is always greater or equal to **Number1**, so there is no need to verify that the difference is positive.

We can create an **iif** formula to represent the logic given above, as shown below:

```
iif ( ( ([N2] - [N1]) <= ([N1] * 0.07) ) ; "Green" ;
iif ( ( ([N1] * 0.07) < ([N2] - [N1]) ) And
( ([N2] – [N1]) <= ([N1] * 0.15) ); "Yellow"; "Red"))
```

The field names have been renamed from **Number1** and **Number2** to N1 and N2, respectively. The construct above is an example of inefficient formula which includes an explicit definition of each range specified, with a lot of parentheses. The following simple and easy to understand custom text field formulas will do the same work;

using the **iif** function:

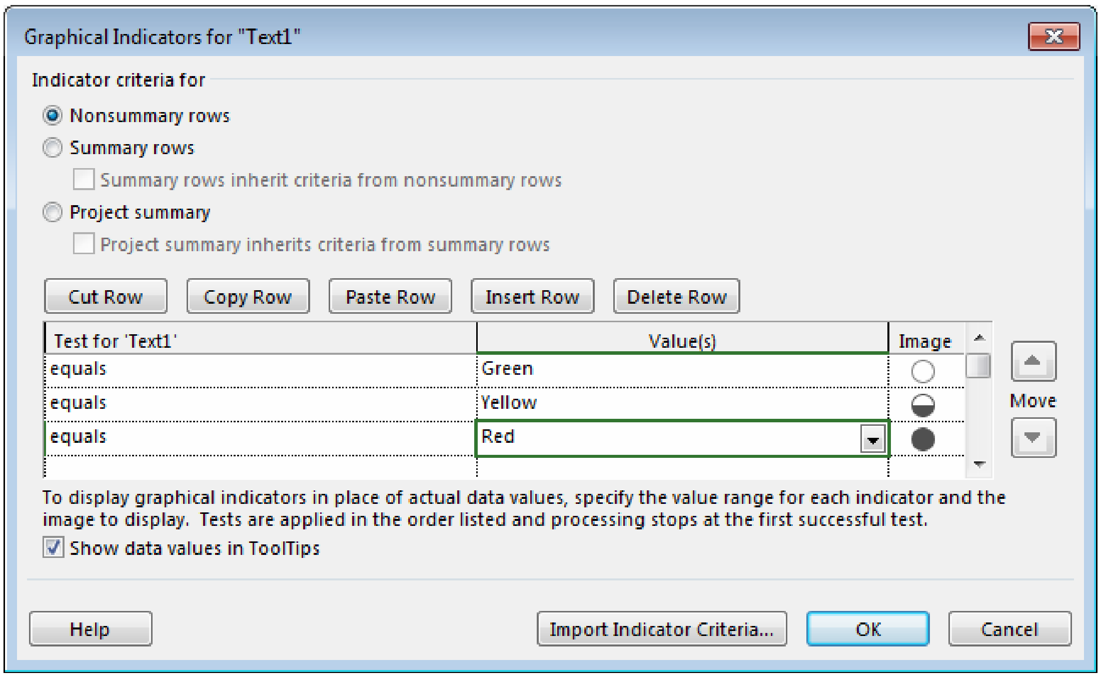
```
iif([N2 <= [N1] * 1.07, "Green",  iif([N2] <= [N1] * 1.15, "Yellow", "Red" ) )
```

or using the **Switch** function:

```
Switch( [N2] <= [N1] * 1.07, "Green", [N2] <= [N1] * 1.15, "Yellow", True, "Red" )
```

Note that the order of the conditions testing to see what range the difference value falls into is important. In the second condition (or the second pair in the **Switch** function) we do not need to check for the lower limit of the range since it has already been covered by the first condition (or the first pair in the **Switch** function).

We can now enter the formula to the **Text1** field and specify the indicator criteria by entering to the table as shown below:



Graphical Indicators for "Text1"

Indicator criteria for

[✓ Nonsummary rows]
Summary rows
☐ Summary rows inherit criteria from nonsummary rows
Project summary
☐ Project summary inherits criteria from summary rows

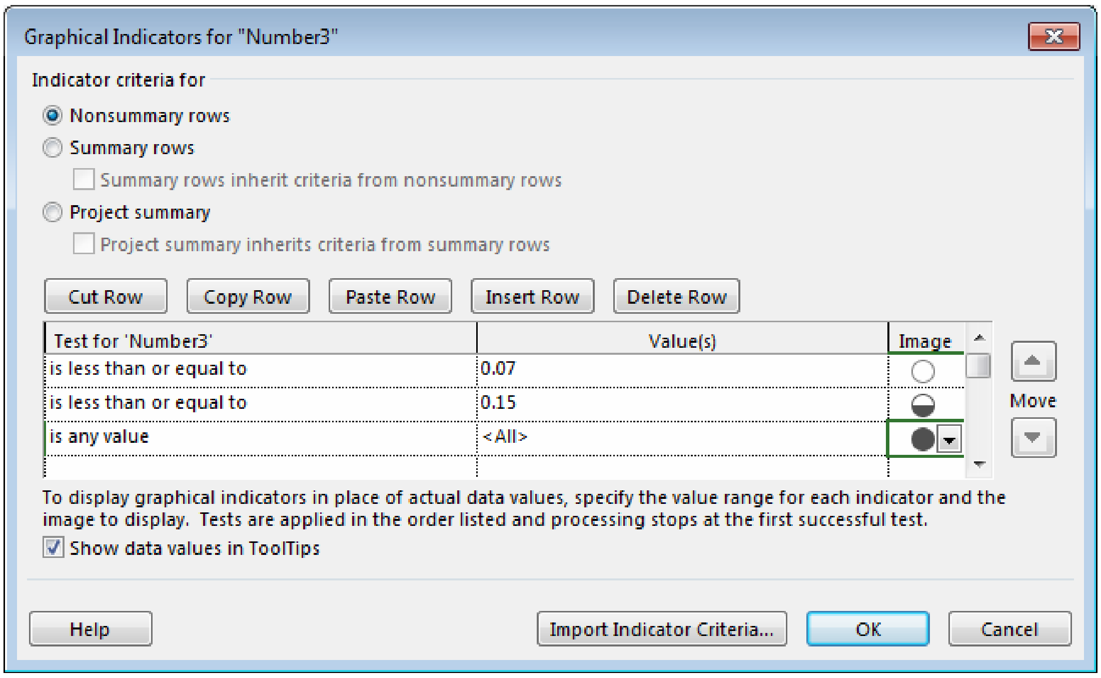
Test for 'Text1'	Value(s)	Image
equals	Green	○
equals	Yellow	◒
equals	Red	●

To display graphical indicators in place of actual data values, specify the value range for each indicator and the image to display. Tests are applied in the order listed and processing stops at the first successful test.

☑ Show data values in ToolTips

A more simple solution to the case would be as such:

- Enter the formula **(Number2 - Number1) / Number1** to the **Number3** field. This formula calculates the percentage of the difference.
- Specify the indicator criteria in the table as shown below:



Graphical Indicators for "Number3"

Indicator criteria for

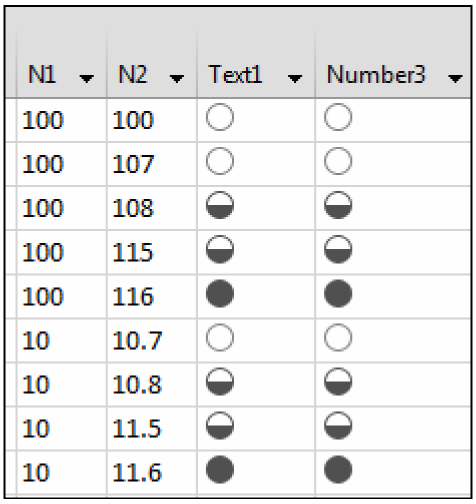
[✓ Nonsummary rows]
Summary rows
☐ Summary rows inherit criteria from nonsummary rows
Project summary
☐ Project summary inherits criteria from summary rows

Test for 'Number3'	Value(s)	Image
is less than or equal to	0.07	○
is less than or equal to	0.15	◒
is any value	<All>	●

To display graphical indicators in place of actual data values, specify the value range for each indicator and the image to display. Tests are applied in the order listed and processing stops at the first successful test.

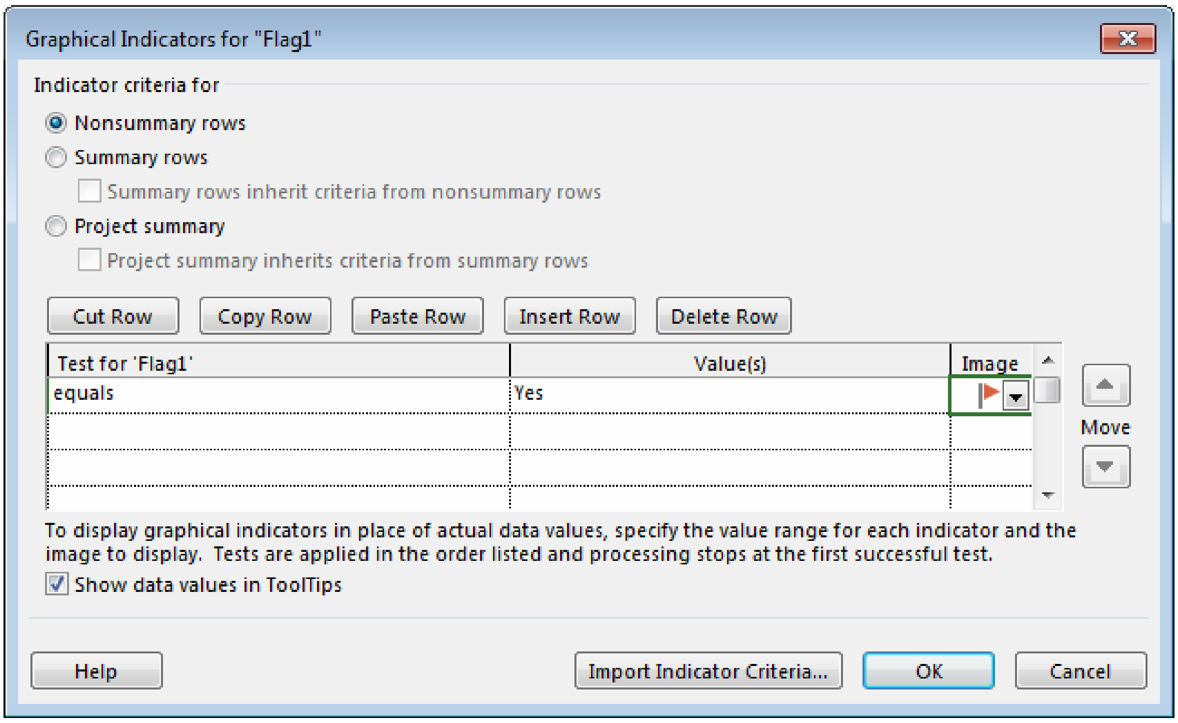
☑ Show data values in ToolTips

In this method, changing the ranges when needed would be obviously a lot easier than editing the formula. See a sample task table below:



N1	N2	Text1	Number3
100	100	○	○
100	107	○	○
100	108	◒	◒
100	115	◒	◒
100	116	●	●
10	10.7	○	○
10	10.8	◒	◒
10	11.5	◒	◒
10	11.6	●	●

Project inserts <All> to the Value(s) column automatically as soon as we select “is any value” in the Test for ‘Number3’ column. This criterion row is used only when all the tests above fails. If not used, then Project uses the blank indicator image by default, for the items that has no indicator specification. For example, while defining indicators for the flag fields, we can enter a single row of criterion. Then Project uses the blank indicator image for the other item(s) by default, as shown below:



Graphical Indicators for "Flag1"

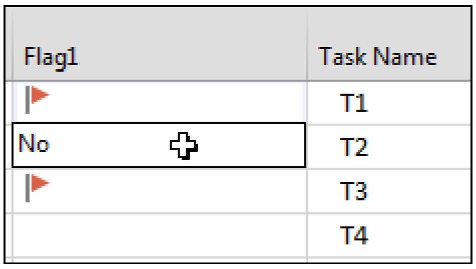
Indicator criteria for

[✓ Nonsummary rows]
Summary rows
Summary rows inherit criteria from nonsummary rows
Project summary
Project summary inherits criteria from summary rows

Test for 'Flag1'	Value(s)	Image
equals	Yes	🚩

To display graphical indicators in place of actual data values, specify the value range for each indicator and the image to display. Tests are applied in the order listed and processing stops at the first successful test.

[✓] Show data values in ToolTips



Flag1	Task Name
🚩	T1
No	T2
🚩	T3
	T4

Note that the checkbox **Show data values in ToolTips** is turned on in the **Graphical Indicators** dialog box, therefore we can see the content of a cell by keeping the mouse pointer over it.

As a final note on the graphical indicators; if a test (or criterion) that we are trying to enter to a row in the Indicator criteria table is not suitable for the custom field, then Project does not allow us to complete the entry and displays a dialog box with the following message:

This type of test does not apply to the field you selected.
Click a different test or a different field in the Field Name column.

Clicking **OK** in the dialog box returns us to the same row, then we can enter a proper test and complete the operation.

Special Operators: Like and Between...And

A criterion defined in a task or resource filter (or custom AutoFilter) or for a graphical indicator can be converted to a condition to be tested in a formula, by using a logical operator corresponding to the test expression selected. Test expression is the item selected from the drop-down list of the Test column in the **Filter Definition** dialog box, or in the **Custom AutoFilter** dialog box or in the Indicator criteria table of the **Graphical Indicators** dialog box. The following table lists test expressions and the corresponding comparison operators:

Test expression	Logical operator
equals	=
does not equal	<>
is greater than	>
is greater than or equal to	>=
is less than	<
is less than or equal to	=<

The table above does not include the tests for "within" and "contain". The "within" tests can be converted into multiple conditions composed of comparison expressions combined by logical operators. For example, consider the filtering criterion **Number1 is within 1,9**. Any value in the **Number1** field that falls within the range specified by 1 and 9 (inclusive) satisfies that criterion. The filtering criterion should be changed to **Number1 is not within 1,9** (inclusive) in order to find the values that falls outside the same range. So how can we create conditions representing these two criteria ? The answer is shown below:

Filter applied to the **Number1** field:		Formula used in the **Text1** field:
Number1 is within 1,9	→	**iif (1 <= [Number1] And [Number1] <= 9; [Number1]; "")**
Number1 is not within 1,9	→	**iif (Not (1 <= [Number1] And [Number1] <= 9); [Number1]; "")**

Using a custom text field avoids display of zeros in the **Text1** column for the values that do not satisfy the filtering criterion. There is a more practical way to perform the tests above in the formulas; using a special operator that does not have a button in the **Formula** dialog box: the **Between...And** operator. So we can construct the formulas as follows:

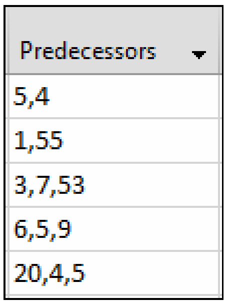
iif ([Number1] Between 1 And 9; [Number1]; "")

and

iif ([Number1] Not Between 1 And 9; [Number1]; "")

Suppose that we need to see the tasks whose predecessor is the task with identification number 5. We can quickly list those tasks by using AutoFilter, but here we will create custom filters (or custom AutoFilters) in order to demonstrate how “contain” tests work as shown below:

Task table:



Predecessors
5,4
1,55
3,7,53
6,5,9
20,4,5

→

Filtering criterion used and the task lines displayed:		
Predecessors contains 5	**Predecessors does not contain 5**	**Predecessors contains exactly 5**
5,4 1,**55** 3,7,**5**3 6,**5**,9 20,4,**5**	No result	**5**,4 6,**5**,9 20,4,**5**

As it is seen in the table above, the tests “contains” and “does not contain” list the predecessor lists containing any number of 5s and the predecessor lists that does not contain any 5s, respectively. The test “contains exactly” recognizes the delimiter and finds only 5s standing alone. The “contain” tests of the filters used for string search in the fields like the one shown above can be performed by using the **Instr** or the **StrComp** functions in formulas.

These formulas will be relatively complex formulas, especially when we need to find the multiple occurrences of a string in the field. Instead, we can use some other special operator, which do not have a button in the **Formula** box; the **Like** operator.

The **Like** operator requires two operands; the left operand is any string expression (here, the field **Predecessors**) and the right operand is the pattern string. The **Like** operator returns true if each character of the string expression matches the character at the corresponding position in the pattern string; for example, the condition **[Text1] Like “abc”** returns true if the **Text1** field contains the string **abc** exactly.

The pattern string may contain any combination of the following elements:

- A specific character.
- A character list: for example, **[abc]** represents any of the characters **a**, **b** or **c**, **[129]** represents any of the numeric characters **1**, **2** or **9**.
- A character or digit range: for example, **[a-z]** represents any character from **a** to **z**, inclusive; **[0-9]** represents any digit from **0** to **9**, inclusive.
  [!a-z] represents outside a range of characters, **[!0-9]** represents no digit.
- A wildcard character: ***** represents any number of characters, **?** represents a single character, **#** represents a single digit. A wildcard character is enclosed in square brackets in order to use as literal character, such as **[*]**, **[?]** and **[#]**.

As a result, the **Like** operator helps us to find the text information in the fields that satisfies any pattern that we define. Let us now create custom flag field formulas doing the same string searches by using the **Like** operator:

Filter applied to the **Predecessors** field:		Formula used in the **Flag1** field:
Predecessors contains 5	→	**[Predecessors] Like "*5*"**
Predecessors does not contain 5	→	**[Predecessors] Not Like "*5*"**
Predecessors contains exactly 5	→	**[Predecessors] Like "5,*" OR [Predecessors] Like "*,5,*" OR [Predecessors] Like "*,5"**

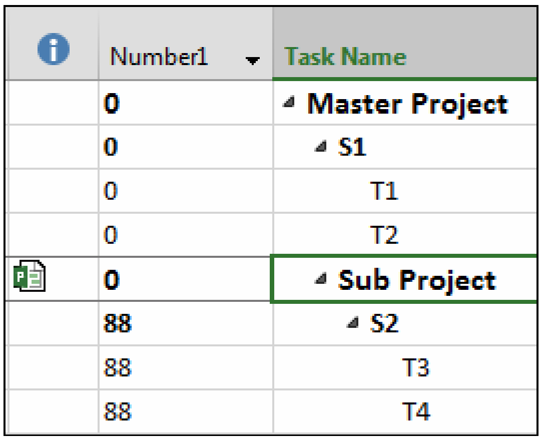
We do not need to use the **iif** function in the formulas entered to the custom flag fields since the **Like** operator returns logical values.

Regarding operator precedence and associativity; both **Like** and **Between…And** operators have the same precedence and associativity as the comparison operators. Also note that the wildcard characters (*****, **?** and **#**) are treated as literal characters when used in the operands of **Between…And** operator.

Inserted Projects and the Custom Fields

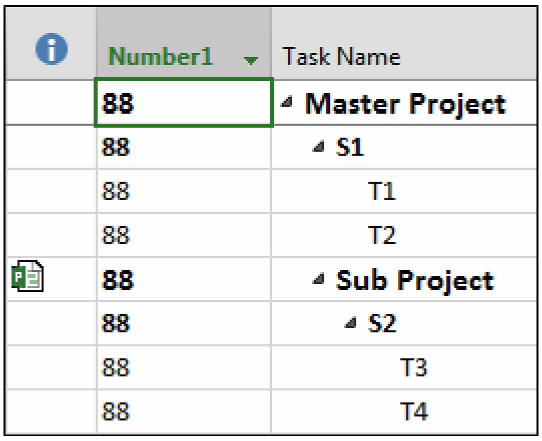
We will know explore how Project handles the custom field formulas while working with inserted projects. Consider two project plan files, Master Project.mpp and Sub Project.mpp. Let us now discuss several custom field formula scenarios with those files in order to understand how it works with subprojects:

- Sub Project.mpp's **Number1** field contains 88 as the formula, and the Calculation for task and group summary rows for this field is set to **Use formula**. Master Project.mpp's **Number1** field is blank. This is how Master Project.mpp looks after inserting Sub Project.mpp:



	Number1	Task Name
	0	**Master Project**
	0	**S1**
	0	T1
	0	T2
	0	**Sub Project**
	88	**S2**
	88	T3
	88	T4

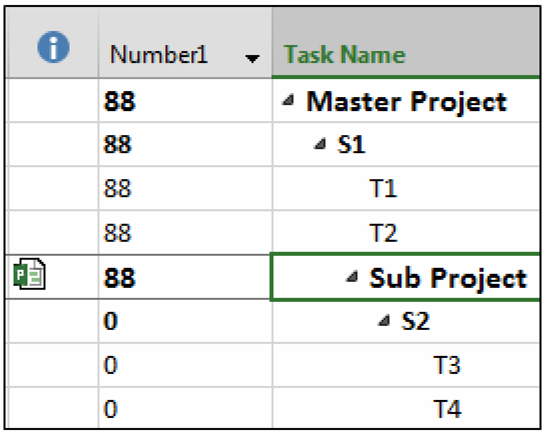
In order to use the formula and the other settings of the **Number1** field in Master Project.mpp, we need to copy the field to Master Project.mpp by using either the **Organizer** dialog box or the **Import Field** command in the **Custom Fields** dialog box. See the results of importing **Number1** to Master Project.mpp below:



	Number1	Task Name
	88	**Master Project**
	88	**S1**
	88	T1
	88	T2
	88	**Sub Project**
	88	**S2**
	88	T3
	88	T4

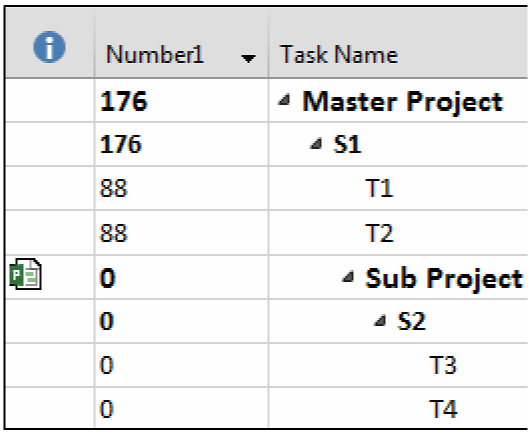
How does it work if we do just the opposite ?; that is, the case where Sub Project.mpp's **Number1** is blank, but Master Project.mpp's **Number1** has a formula (i.e., 88). See the results with different Calculation for task and group summary rows settings for **Number1** in Master Project.mpp below:

Use formula



ℹ	Number1	Task Name
	88	**Master Project**
	88	**S1**
	88	T1
	88	T2
	88	**Sub Project**
	0	**S2**
	0	T3
	0	T4

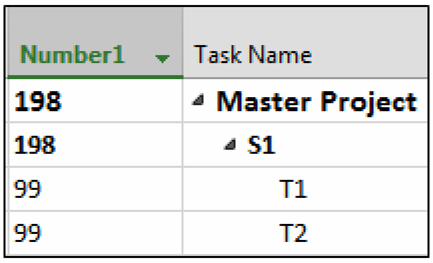
Rollup: Sum



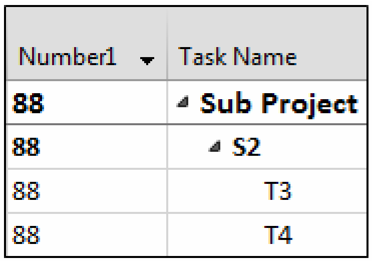
ℹ	Number1	Task Name
	176	**Master Project**
	176	**S1**
	88	T1
	88	T2
	0	**Sub Project**
	0	**S2**
	0	T3
	0	T4

Here, outdenting Sub Project, to the same outline level as S1, does not change the results on the summary rows in both pictures.

- The examples above are quite straightforward. Let us now experiment with a different scenario. Consider the project files below:



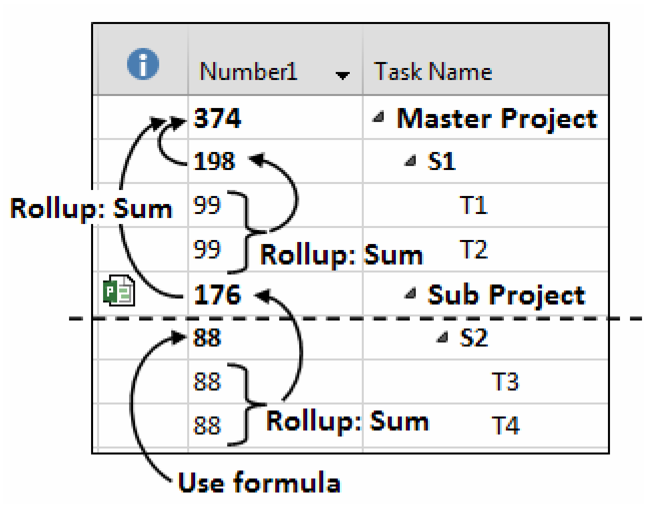
Number1	Task Name
198	**Master Project**
198	**S1**
99	T1
99	T2



Number1	Task Name
88	**Sub Project**
88	**S2**
88	T3
88	T4

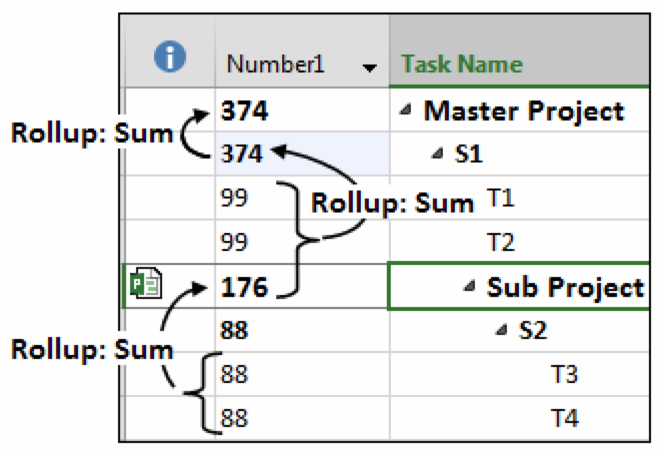
This time, Master Project.mpp has the formula 99 in the **Number1** field, and the Calculation for task and group summary rows for this field is set to **Rollup: Sum**. Sub Project.mpp's **Number1** field contains 88 as the formula, and the Calculation for task and group summary rows for this field is set to **Use formula**.

This is how Master Project.mpp looks after inserting Sub Project.mpp:



	Number1	Task Name
	374	Master Project
	198	S1
	99	T1
	99	T2
	176	Sub Project
	88	S2
	88	T3
	88	T4

Note that both Master Project and Sub Project use their own formulas and the Calculation for task and group summary rows settings for **Number1**; see below how indenting Sub Project changes the rolled-up data of Master Project's S1:

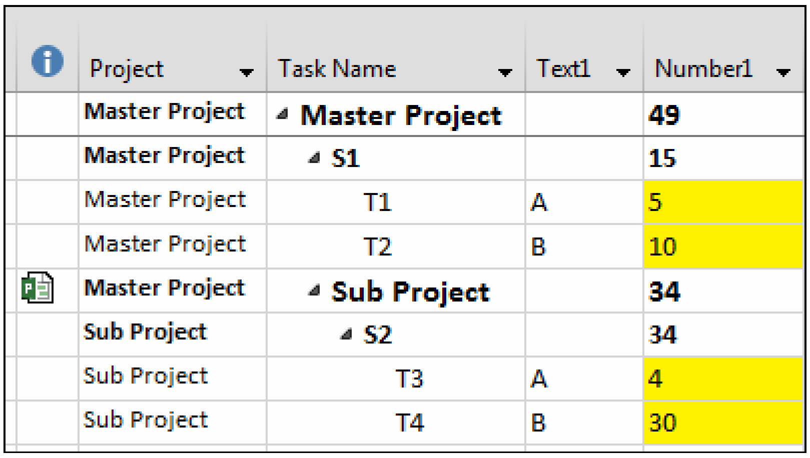


	Number1	Task Name
	374	Master Project
	374	S1
	99	T1
	99	T2
	176	Sub Project
	88	S2
	88	T3
	88	T4

As it is seen from the results in **Number1** at the task level, the master project and the inserted projects (that is, linked subprojects) may have different formulas, but how inserted projects contribute to the custom field's summary level results in the master project is determined according to the Calculation for task and group summary rows setting of the custom field in the master file. If the custom field is renamed differently in both files, then the name in the master project file will be effective.

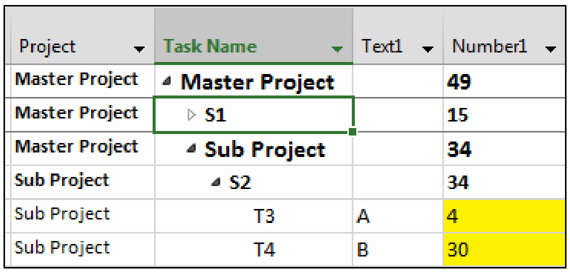
As shown in the examples, Project does not complain if a custom field's formula and settings in inserted projects differ from those in the master project; therefore, in order to obtain accurate results from the formulas we need to ensure that the inserted projects and the master project have identical formulas and settings for the custom fields used.

Consider the simple schedule below; **Number1** has some values corresponding to As and Bs in **Text1**, entered by typing in, with rolledup sums at the summary level.

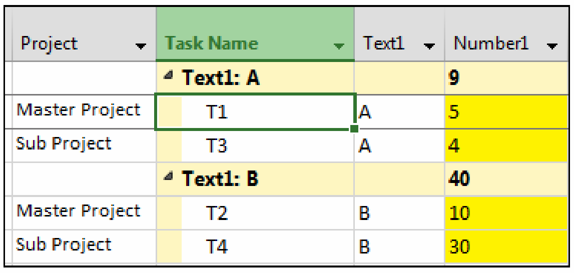


ℹ	Project	Task Name	Text1	Number1
	Master Project	◢ Master Project		49
	Master Project	◢ S1		15
	Master Project	T1	A	5
	Master Project	T2	B	10
	Master Project	◢ Sub Project		34
	Sub Project	◢ S2		34
	Sub Project	T3	A	4
	Sub Project	T4	B	30

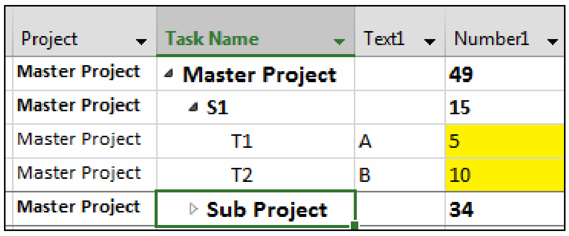
Suppose that you want to see the total values for As and Bs and therefore, you apply an AutoFilter group to **Text1**. While collapsing S1 does not affect the resulting group, collapsing the subproject (not S2) before applying the group changes the results, as shown below:



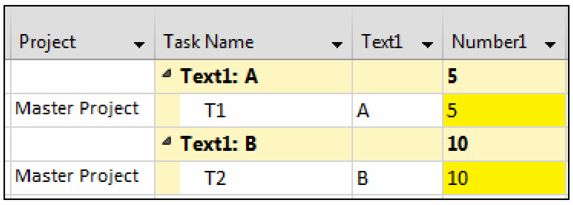
Project	Task Name	Text1	Number1
Master Project	◢ Master Project		49
Master Project	▷ S1		15
Master Project	◢ Sub Project		34
Sub Project	◢ S2		34
Sub Project	T3	A	4
Sub Project	T4	B	30



Project	Task Name	Text1	Number1
	◢ Text1: A		9
Master Project	T1	A	5
Sub Project	T3	A	4
	◢ Text1: B		40
Master Project	T2	B	10
Sub Project	T4	B	30



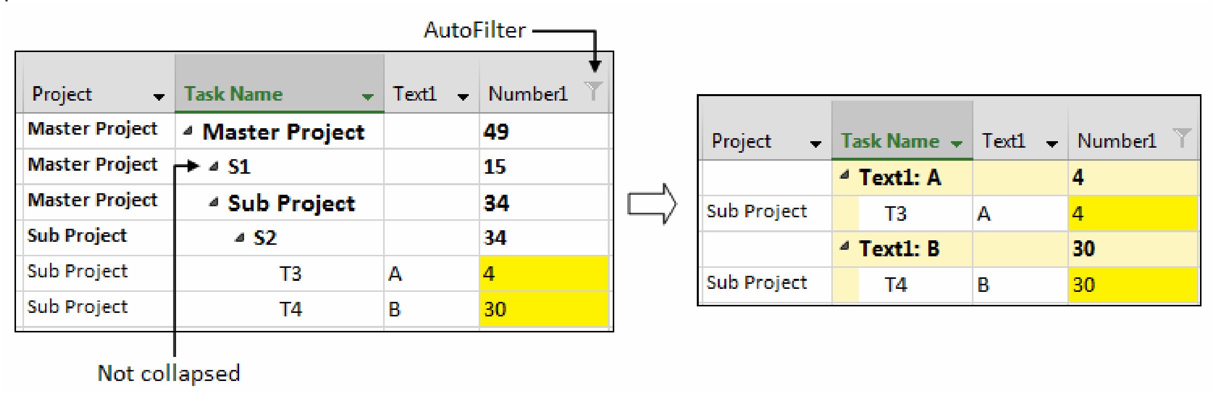
Project	Task Name	Text1	Number1
Master Project	◢ Master Project		49
Master Project	◢ S1		15
Master Project	T1	A	5
Master Project	T2	B	10
Master Project	▷ Sub Project		34



Project	Task Name	Text1	Number1
	◢ Text1: A		5
Master Project	T1	A	5
	◢ Text1: B		10
Master Project	T2	B	10

As a result, always make sure to expand the task outline completely by selecting **All Subtasks** in **Show Outline** command on the VIEW tab before applying the group in a schedule containing linked subprojects.

Suppose that you do not want to include T1 and T2 in the group, then you can filter them out by applying the AutoFilter to **Number1** (that is, clear the checkboxes for 5 and 10 in the AutoFilter pane). The result will be as follows:



Project	Task Name	Text1	Number1
Master Project	Master Project		49
Master Project	S1		15
Master Project	Sub Project		34
Sub Project	S2		34
Sub Project	T3	A	4
Sub Project	T4	B	30

Project	Task Name	Text1	Number1
	Text1: A		4
Sub Project	T3	A	4
	Text1: B		30
Sub Project	T4	B	30

The resource custom fields defined in a project plan used as a resource pool overwrite all the corresponding resource custom fields in the project plan files sharing the resource pool.

Index

Add New Column, 18
Boolean literal, 23
Built-in Functions
category, 34
data types, 39
definition, 33
function description, 38
name, 36
nesting, 44
parameters, 36
reference syntax, 34
return types, 39
Comparisons
date comparison, 83, 204, 205
numeric comparison, 28, 51, 83, 87, 93, 111, 191, 193, 195, 196, 199, 204, 220, 237
string comparison, 27, 28, 51, 83, 93, 96, 97, 99, 192, 193, 195, 198, 199, 203, 204
Conversion and Date/Time functions
CDate, 34, 96, 97, 155, 158, 161, 162, 163, 164, 165, 167, 168, 169, 171, 172, 189, 197, 226, 227, 231, 237, 238
DateSerial, 34, 44, 157, 158, 160, 165, 166, 170, 179, 193
DateValue, 34, 35, 83, 86, 87, 158, 162, 163, 164, 165, 169, 170, 171, 184, 190, 193, 194, 226, 238
Day, 34, 50, 82, 158, 160
Hour, 34, 55, 134, 158, 161, 172, 198, 223
Minute, 34, 158, 161, 198
Month, 34, 158, 160, 161, 166, 179, 180
Second, 34, 158, 161, 198
TimeSerial, 34, 158, 171, 172, 184, 193, 197, 198, 199, 200
TimeValue, 34, 158, 170, 171, 197, 198, 199, 200
Weekday, 34, 39, 55, 158, 180, 181, 183, 184, 185
Year, 34, 158, 160, 166, 179
Conversion and Text functions
Asc, 34, 97, 98
Chr, 34, 98, 229
StrConv, 34, 55, 98, 104
Conversion functions
CBool, 34, 49, 52, 53, 213, 214
CByte, 34, 44, 123
CCur, 34, 41, 123, 124
CDbl, 34, 41, 42, 43, 51, 52, 83, 94, 95, 101, 107, 195, 229
CDec, 34
CInt, 34, 44, 123
CLng, 34, 44, 123, 124
CSng, 34, 41, 44, 199
CStr, 34, 42, 83, 91, 94, 95, 96, 97, 101, 102, 108, 109, 155, 156, 191, 192, 228, 229
CVar, 34
Hex, 34
Oct, 34
Str, 34, 101, 108, 109
Val, 34, 35, 120, 135, 188, 229, 230
Custom field formula. *See* Formula

Custom fields
cross-category field access, 14
Date, Start and Finish, 145
comparisons and arithmetic operations, 190
customizing date display format, 145
date related project level fields, 152
date type resource fields, 151
date type task fields, 150
default times, 147
entering values, 148
functions handling date and time data, 158
hard-coded date and times, 192
how stored, 153
limits, 156
passing dates to functions, 161
setting field value to NA, 194
testing for NA, 194
Duration, 127
customizing default time units, 129
customizing time unit labels, 130
entering values, 129
functions handling duration data, 142
holding duration data, 128
holding work data, 128
limits, 139
time unit conversion factors, 131
Flag and Text, 91
handling text by using functions, 98
returning values to text fields, 94
inserted projects, 73
intermediate, 5
limits, 17
locking, 13
Number and Cost, 111
entering values, 111
returning values, 113
protecting data, 13
renaming
Field Settings dialog box, 9
Rename Field dialog box, 9
summary rows, 58
Custom Fields dialog box
opening, 5
Sections
Calculation for assignment rows, 19
Calculation for task and group summary rows, 17, 62
Custom attributes, 6, 8, 57
Delete button, 57
Field, 6
Import Field button, 58
Rename button. *See* Rename Field dialog box
Type, 16, 17

Values to display, 64

Date and/or time literal, 22

Date/Time functions

Date, 34, 97, 158, 159, 160, 164, 166, 180, 185, 190

DateAdd, 34, 55, 142, 158, 180, 189

DateDiff, 34, 36, 42, 55, 142, 158, 164, 172, 173, 174, 175, 177, 178, 185, 189, 228, 230, 232, 233, 234, 235

DatePart, 34, 55, 158, 160, 161, 172, 173, 174, 175, 176, 177, 179, 185, 188, 189, 232, 233

IsDate, 34, 49, 50, 158, 161, 196, 205, 207, 209, 210, 212, 213, 214, 216, 238

Now, 34, 97, 158, 159

Time, 34, 35, 38, 39, 158, 159

Timer, 34, 158, 159, 160

Delete Custom Fields dialog box, 18

Empty string, 45

Expression

definition, 21

elements

functions. *See* Functions

literals, 22

operators, 24

nesting, 31

syntax, 21

type, 21

types explained

Boolean expressions. *See* logical expressions

comparison expressions, 21, *See* Comparisons

date expression, 21

logical expressions, 21

numeric expressions, 21

string expressions, 21, 44

Field

definition, 12

entry type

calculated, 18

calculated or entered, 18

entered, 18

null, 18

properties

category, 14

data type. *See* type

name, 13

type, 14

types explained

custom field, 16

non-custom field, 14

Field reference

category name, 7

definition, 7, 13, 21

initial value, 19

inserting, 7

Format specifiers, 103

Formula
#ERROR, 18, 25, 28, 29, 42, 43, 44, 45, 46, 47, 48, 57, 86
case sensitivity, 8
copy/pasting, 10
definition, 6, 21
documenting, 12
elements. *See* Expression: elements
enabling, disabling, 8
entering, 5
logical errors, 79
predefined constants, 55, 220
saving, 8
syntax check, 8, 84
syntax error, 10, 21, 25, 31, 44, 84
testing, 9
type mismatch, 18
Undo editing, 8
Formula dialog box
opening, 6
Sections
Edit formula, 7
Field button, 7, 19
Function button, 33
Import Formula button, 58
Operator buttons, 24
Functions
Math
Abs, 34, 118, 119, 120, 143
Atn, 34
Cos, 34
Exp, 34
Fix, 34, 118, 119, 120, 121, 122, 123
Int, 34, 53, 83, 118, 119, 120, 121, 122, 143, 159, 179, 180, 193, 199
Log, 34, 196
Rnd, 34
Sgn, 34, 118, 119, 120, 121
Sin, 34
Sqr, 34, 35, 44
Tan, 34
General functions
Choose, 34, 53, 54, 99
IsNull, 34, 49, 50, 54
IsNumeric, 34, 49, 50, 51, 52, 107, 205, 207, 214, 215
iif, 34, 45, 107
Switch, 34, 45, 46, 47, 48, 50, 53, 61, 65, 81, 99
Graphical Indicators
defining, 64
Graphical Indicators dialog box, 64
Sections
Import Indicator Criteria button, 58
Importing
Import Custom Field dialog box. *See* Import Field button

Import Formula dialog box. *See* Import Formula button
Import Indicator Criteria dialog box. *See* Import Indicator Criteria button
Logical literal. *See* Boolean literal
Logical parameters. *See* Logical expressions
Logical value. *See* Boolean literal
Manually scheduled task
evaluating based on task mode, 203
placeholder, 201
Null, 50
Numeric literal, 22
Operators
arithmetic operators, 24
Boolean operators. *See* logical operators
comparison operators, 27
concatenation operators, 29
logical operators, 28
operand, 24
precedence and associativity, 30
special operators Like, Between...And, 69
Project and Conversion functions
ProjDateConv, 34, 158, 219, 222, 223, 224, 225, 227
ProjDurConv, 34, 135, 137, 142, 219, 221, 222, 227, 230
Project and Date/Time functions
ProjDateAdd, 34, 39, 142, 158, 189, 219, 230
ProjDateDiff, 34, 35, 36, 38, 39, 136, 137, 142, 158, 209, 210, 211, 212, 213, 219, 230, 232, 233
ProjDateSub, 34, 142, 158, 189, 219, 230
ProjDateValue, 34, 50, 158, 162, 163, 168, 194, 200, 219, 227, 235, 238
Project functions
ProjDurValue, 34, 135, 142, 204, 219, 225
Roll-up methods. *See* Calculation for task and group summary rows
Rounding
arithmetic rounding. *See* round half up
explained, 115
Round function, 123
round half to even, 26, 117
round half up, 115
rounding off to the nearest even integer, 122
String. *See* String literal
String literal, 22
String parameters. *See* String expressions
String value. *See* String literal
Text functions
Format, 34, 55, 83, 91, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 116, 117, 120, 122, 147, 155, 156, 157, 158, 179, 180, 181, 182, 183, 185, 188, 198, 199, 200, 205, 207, 226
Instr, 34, 70, 100, 101, 229
LCase, 34, 98, 100, 104
Left, 34, 98, 101, 227, 229
Len, 22, 34, 45, 98, 101
LTrim, 34, 98
Mid, 34, 98
Right, 34, 98, 101
RTrim, 34, 98

Space, 34, 98
StrComp, 34, 55, 70, 99, 100
String, 34, 42, 45, 98, 107
Trim, 34, 98, 101
UCase, 34, 98, 100, 104

Text string. *See* String literal

Truncating, 118

Type conversions
explicit, 43
implicit, 43

Week parameters
firstdayofweek, 55, 102, 103, 172, 173, 175, 176, 177, 178, 179, 180, 181, 182, 183, 185, 188
firstweekofyear, 55, 102, 103, 172, 173, 175, 176, 177, 178, 179, 182, 185, 188

Zero-length string. *See* Empty string

About the Author

Visit the author’s website at www.ismetkocaman.com

eBook’s website: www.ismetkocaman.com/Formulas/eBook.html