The Boeing Company

OPP Adv Calc Fields

INTSCH303 – OPP Advanced Calculated Fields

Training Manual / Reference Guide

Integrated Planning & Scheduling - K. Jeff Turner (jeffry.turner@boeing.com)

10-9-2025

TABLE OF CONTENTS

Objectives

To become familiar with the elements of calculated fields and to be able to use functions to create calculated field expressions or interpret existing expressions.

Setup: In the Training Environment, have the students open the project "**INTSCH303**" and Save As to make a copy for themselves to work in. They should name their project in the format of <UserID>_INTSCH303. Open the view "TURNERKJ_STATUS_REQD".

Introduction

OPP has a very powerful feature that allows the user to create virtual fields that perform calculations called "Calculated Fields". The values in Calculated Fields are not stored in the database – just the name an expression. The values are calculated on-the-fly when displayed as a field in a view or referenced in another CLC object. Calculated field expressions can be very simple -- just get the value from another field -- or very complex involving many logical operations and calculations depending on your need. The more complex, the more difficult to construct. These calculated fields can be used in Filters and Global Edits.

Calculated fields cannot iterate through activities like a macro. Calculated fields can only look at fields on tables relative to the same record and make comparisons or perform calculations. A calculated field cannot, for example, compare the Baseline Finish on activity 1010 to the Baseline Finish on activity 1020. It can only compare data on the same record.

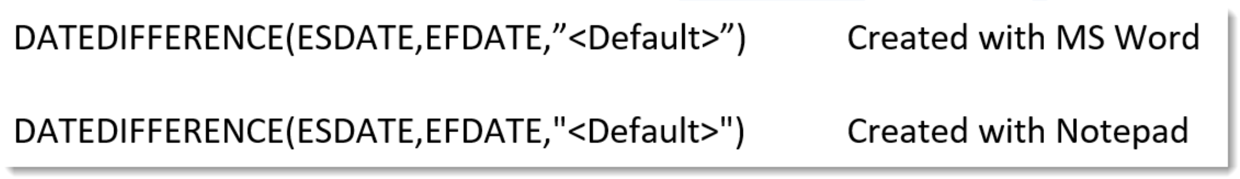
In developing calculated fields, it helps to be familiar with the most common field names such as:

Field Name	Field Description	Data Type
ACT_ID	Activity ID	Alphanumeric
DESCRIPTION	Activity Description	Text
ESDATE	Early Start	Date
EFDATE	Early Finish	Date
BSDATE	Baseline Start	Date
BFDATE	Baseline Finish	Date
ASDATE	Actual Start	Date
AFDATE	Actual Finish	Date
TGTSDATE	Target Start	Date
TGTFDATE	Target Finish	Date
ORIG_DUR	Original Duration	Duration
PPC	Physical % Complete	Numeric
XFDATE	Expected Finish	Date

USER_CHR01	User Character Field 1 (1-10)	Text
USER_NUM01	User Numeric Field 1 (1-10)	Numeric
USER_DTE01	User Date Field 1 (1-10)	Date
C_USER_FDTE01	User Finish Date 1 – CSPR (1-5)	Date
C_USER_DUR01	User Defined Duration Field 1 - CSPR (1-5)	Duration

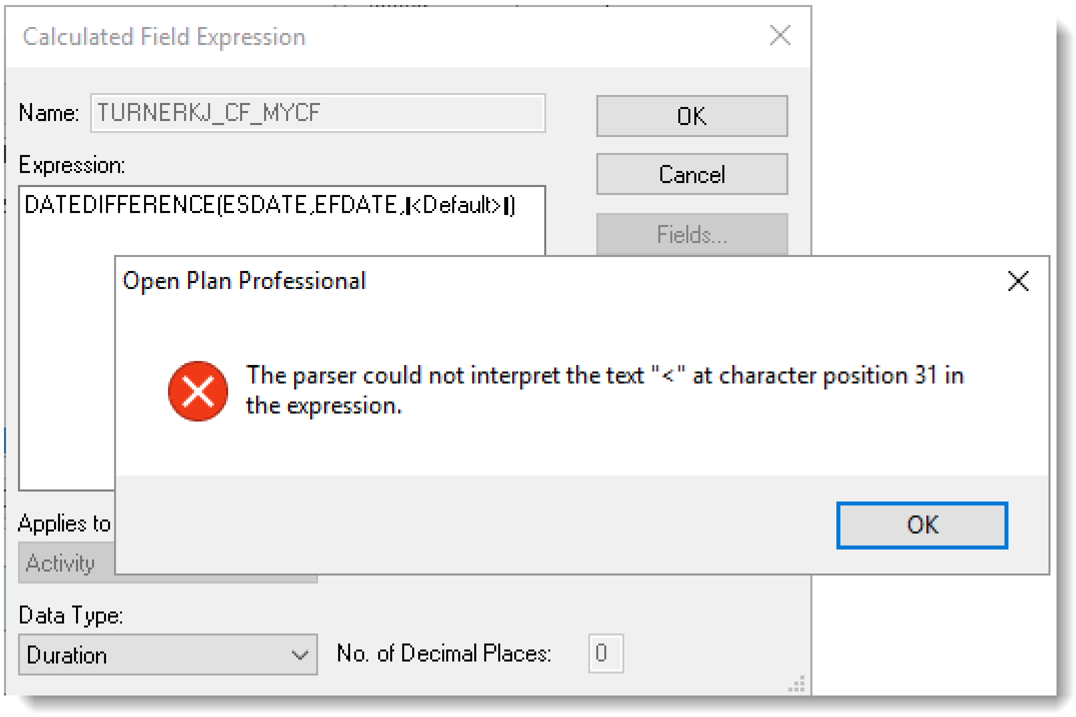
While developing calculated field expressions, use Notepad (or Notepad ++) to rough out the formula. If for no other reason, you can make the font much larger to read as you develop the expression making it much easier to see the commas and parenthesis that are vital to the expression syntax.

Be careful using other applications such as Microsoft Word or Excel as they tend to format text such as double quotation marks that will look *ok* but will generate an error in the calculated field Expression. In the example below, notice the subtle difference in the quotation marks when created with MS Word vs Notepad.



DATEDIFFERENCE(ESDATE,EFDATE,”<Default>”)	Created with MS Word
DATEDIFFERENCE(ESDATE,EFDATE,"<Default>")	Created with Notepad

If you copy the Word version into the OPP calculated field expression dialog box those quotation marks from Word will be translated to ASCII characters and will return an error. Thankfully, you’re given a clue, at least, where to look for the offender as shown below.



Calculated Field Expression

Name: TURNERKJ_CF_MYCF

Expression:

```
DATEDIFFERENCE(ESDATE,EFDATE,|<Default>|)
```

Open Plan Professional

The parser could not interpret the text "<" at character position 31 in the expression.

Applies to: Activity

Data Type: Duration

No. of Decimal Places: 0

Error: The Parser could not interpret the text

For more complex calculated fields, build from the bottom up, piece by piece testing each piece as you go. It's easier to troubleshoot if you've verified that the pieces work as you go rather than dump a large expression in at one time and try to determine what's causing a problem. OPP is very unforgiving when it comes to writing expressions – you must use proper syntax -- and is not the greatest at giving you helpful feedback as to why your expression does not work.

The two most common errors:

"The parser was not able to interpret the calculated field expression." (Normally caused by a problem with the syntax of the expression…i.e. missing comma, parenthesis…etc.)

"Incompatible data types in expression." (Normally caused by an expression returning one data type and the field data type being set to something else. Also caused when part of the expression is referring to an enumerated value but the square brackets are missing around the value.)

As a Calculated Field naming convention, we recommend: Your ID_CF_Description
Where:

Your ID_ is your BEMS ID in the case of your own personal CFs or a common program identifier such as “F18” or “F15” in the case of a real program standard use CF.

CF_ is used to identify the object as a Calculated Field (CF) Remember to use underscores in place of spaces.

Description is an abbreviated description of what the CF is doing. The name is limited to 60 characters.

Naming your CLC objects this way will accomplish 2 helpful things for you:

1) You can tell what kind of CLC object it is just by the name;
2) It will sort like objects together making them easier to find.

As you build the expression in Notepad, you can wrap the expression to make it easier to read; then copy/paste into the CF expression dialog box.

Expressions defining a calculated field can include the following elements:

- Constants
- Field names
- Functions
- Other calculated fields
- Mathematical operators
- Character operators
- Duration operators
- Relational operators
- Logical operators
- User-Defined Variables

This section discusses each of the elements of a calculated field expression with the exception of functions, which are discussed in the following section.

Calculated field expressions are limited to **4,096** characters.

Chapter 1 – Constants & Operators

Constants

Constants refer to specific values that are “hard coded”, you might say, into the calculated field expression. For example, suppose you wanted to look for the string “software” in the activity Description field. In the expression you will be looking for the specific value “software” which is considered a constant in the expression. You can include text, date, numeric, and logical constants in a calculated field expression according to the following guidelines:

Text — All text must be enclosed in either single or double quotes. For example:

- "Programmers"
- 'Phase I'

Dates — Dates must be enclosed in curly brackets { } and can be expressed in any valid date format. For example:

- {01JAN01}
- {12/01/01}

Durations — Durations must be enclosed between pipe characters (|) and can be expressed in any valid duration format. For example:

- |4h|
- |3.5d|

Numerics — Numeric constants can include any positive or negative value and can appear with or without decimal places. For example:

- 1
- 320000
- 12.1
- –123.78

Logical Values — Valid logical constants are as follows:

- TRUE or T
- FALSE or F

Enumerated Values — Values for enumerated fields (that is, fields with a limited number of valid values) must be enclosed between square brackets [ ]. For example, an expression can include references to any of the possible values for the Act_type field:

- [ASAP]
- [ALAP]
- [Start Milestone]
- [Finish Milestone]
- [Discontinuous]
- [Subproject]
- [Hammock]

- [Effort Driven]
- [External Subproject]

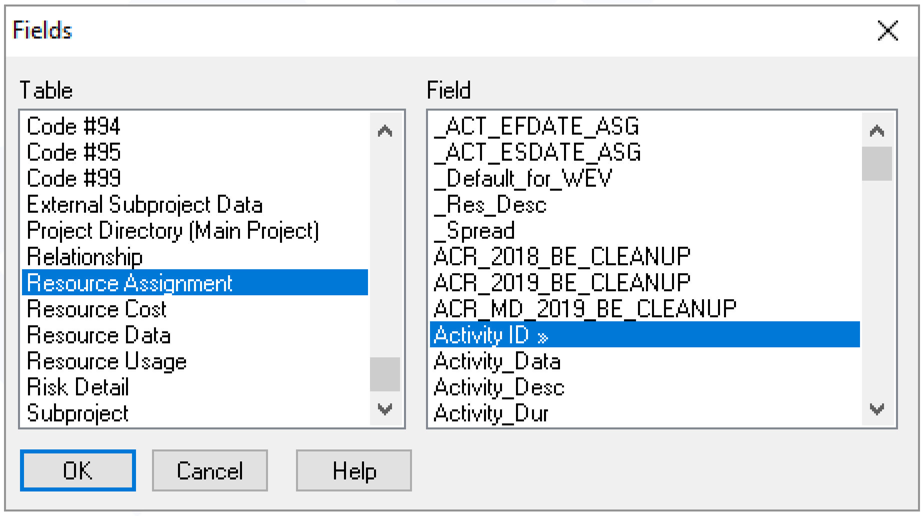
Field Names

When including a field in an expression for a calculated field, you must use the short database field name and not the descriptive name that is displayed in spreadsheet column headings. For example, if you want to define a calculated field expression that references the field containing early start dates, you must identify the field as ESDATE; not Early Start or ORIG_DUR; not Original Duration.

If you need to refer to fields stored in tables other than the primary data table, use the name of the linking field, followed by a period (.) and the field name in the linked table. For example:

- **C1.DESCRIPTION** — the description for a code stored in the C1 field.
- **RES_ID.DESCRIPTION** — the description for a resource ID stored in the Res_ID field.

If you display the list of available tables and fields by clicking **Fields** on the **Calculated Field Expression** dialog box, linking fields are distinguished by a double chevron (») next to the field name. In the example below, when the Resource Assignment table is selected, the linking field is "Activity ID »".



Linking field

Mathematical Operators

You can include the following mathematical operators in a calculated field expression:

Add (+) – Use the plus (+) symbol to add numeric values
Example: 72+72 will return "144"

Subtract (-) – Use the minus (-) symbol or dash to subtract numeric values
Example: 144-72 will return "72"

Multiply (*) – Use the asterisk (*) symbol to multiply numeric values
Example: 12*12 will return "144"

Divide (/) – Use the forward slash (/) to divide numeric values
Example: 144/12 will return "12"

Group (( )) – Use open and closed parentheses to indicate order of operations.
Example: ((8+4)*2)*6 will return "144"

Exponentiation (^ or **) – Use the caret or double-asterisk to raise to a power.
Example: 12^2 or 12**2 will return "144"

Expressions containing mathematical operators are evaluated according to the normal rules of precedence. Grouping operations are performed before multiplication and division operations. Multiplication and division operations are performed before addition and subtraction operations.

Character / String Operators

You can include the following character operators in a calculated field expression:

Concatenate (+) – Use the plus (+) sign to concatenate to values together as one. Using the plus sign concatenates *string* values but performs addition if *decimal* or *integer* values.
Example: "ACT_ID" + " " + "ACT_DESC" returns "10.10 Final Design" where "10.10" is the Activity ID and "Final Design" is the Activity Description.

Is Contained In ($) – Use the dollar currency ($) symbol to test if one string is contained in another string.
Example: "Design" $ ACT_DESC returns "True" if the string Design is contained in the Activity Description field.

The *Is Contained In* operator ($) returns a logical result of True or False.

Duration Operators

When working with durations, you can include the following duration operators in a calculated field expression:

- Add (+)
- Subtract (-)
- Multiply (*)
- Divide (/)

Relational Operators

Relational operators allow you to compare values relative to each other. Expressions for calculated fields can include the following relational operators:

Equal to (=)
Examples: 2=2 returns "True"; 2=3 returns "False"

Not equal to (<>)
Examples: 3<>2 returns "True"; 3<>3 returns "False"

Greater than (>)
Examples: 3>2 returns "True"; 2>3 returns "False"

Greater than or equal to (>=)
Examples: 3>=2 returns "True"; 3>=3 returns "True"; 2>=3 returns "False"

Less than (<)
Examples: 3<2 returns "False"; 2<3 returns "True"

Less than or equal to (<=)
Examples: 3<=2 returns "False"; 3<=3 returns "True"; 2<=3 returns "True"

Contained in ($)
Example: "A"**$**"ABC" returns "True"; "Z"**$**"ABC" returns "False"

Logical Operators

You can include the following logical operators in a calculated field expression:

AND
Example: AFDATE NOT_EMPTY **AND** PPC<>100

AND NOT
Example: Example: ORIG_DUR=|0| **AND NOT** ACT_TYPE=[Finish Milestone]

OR
Example: DESCRIPTION CONTAINS('Des') **OR** DESCRIPTION CONTAINS('Design')

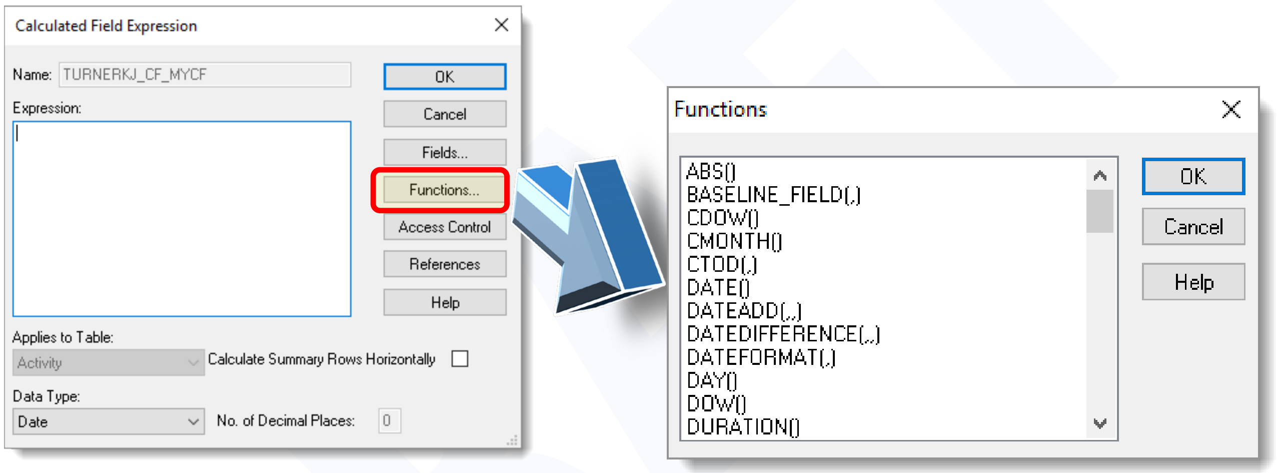
Group ()
Example: **(**ACT_TYPE=[Finish Milestone] OR ACT_TYPE=[Start Milestone]**)** and ORIG_DUR<>|0|

Chapter 2 – Frequently Used Functions

Functions act as "predefined" formulas that enable the user to more easily create custom fields by simplifying the expression. OPP provides 61 functions to use. In this chapter we will explore a few of the most commonly used functions.

Getting Help with Functions

Creating calculated fields is largely a matter of becoming familiar with common functions used to manipulate data. Get familiar with the Functions that are available in OPP. To see the functions available, create a new or edit an existing CF. Once inside the Expression window, click the Functions button to see list.



Calculated Field Expression

Name: TURNERKJ_CF_MYCF

Expression:

Applies to Table: Activity

Calculate Summary Rows Horizontally ☐

Data Type: Date

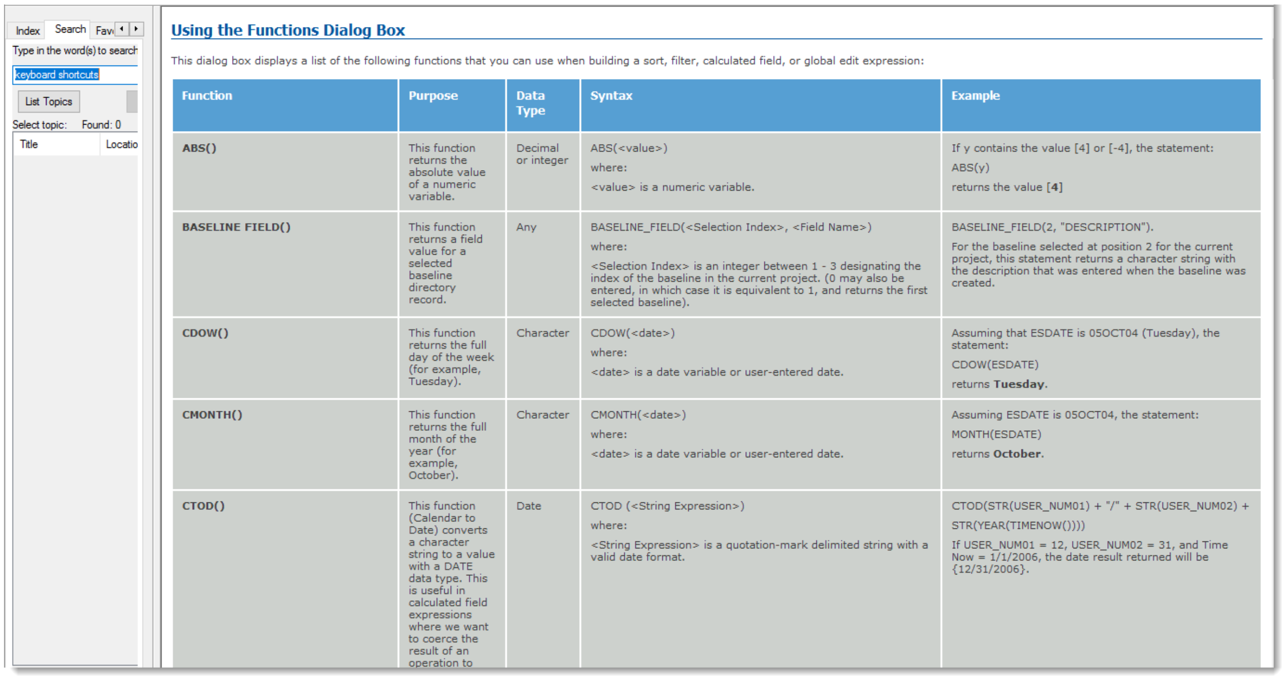
No. of Decimal Places: 0

Functions

ABS()
BASELINE_FIELD(,)
CDOW()
CMONTH()
CTOD(,)
DATE()
DATEADD(,,)
DATEDIFFERENCE(,,)
DATEFORMAT(,)
DAY()
DOW()
DURATION()

To see more detail about each function, press the Help button on the Functions dialog box. You can click the link for each function to get more detail about it.

You can also access the Functions directly from the "Help" button in the Calculated Field Expression dialog box.



Using the Functions Dialog Box

This dialog box displays a list of the following functions that you can use when building a sort, filter, calculated field, or global edit expression:

Function	Purpose	Data Type	Syntax	Example
ABS()	This function returns the absolute value of a numeric variable.	Decimal or integer	ABS(<value>) where: <value> is a numeric variable.	If y contains the value [4] or [-4], the statement: ABS(y) returns the value [**4**]
BASELINE FIELD()	This function returns a field value for a selected baseline directory record.	Any	BASELINE_FIELD(<Selection Index>, <Field Name>) where: <Selection Index> is an integer between 1 - 3 designating the index of the baseline in the current project. (0 may also be entered, in which case it is equivalent to 1, and returns the first selected baseline).	BASELINE_FIELD(2, "DESCRIPTION"). For the baseline selected at position 2 for the current project, this statement returns a character string with the description that was entered when the baseline was created.
CDOW()	This function returns the full day of the week (for example, Tuesday).	Character	CDOW(<date>) where: <date> is a date variable or user-entered date.	Assuming that ESDATE is 05OCT04 (Tuesday), the statement: CDOW(ESDATE) returns **Tuesday**.
CMONTH()	This function returns the full month of the year (for example, October).	Character	CMONTH(<date>) where: <date> is a date variable or user-entered date.	Assuming ESDATE is 05OCT04, the statement: MONTH(ESDATE) returns **October**.
CTOD()	This function (Calendar to Date) converts a character string to a value with a DATE data type. This is useful in calculated field expressions where we want to coerce the result of an operation to	Date	CTOD (<String Expression>) where: <String Expression> is a quotation-mark delimited string with a valid date format.	CTOD(STR(USER_NUM01) + "/" + STR(USER_NUM02) + STR(YEAR(TIMENOW()))) If USER_NUM01 = 12, USER_NUM02 = 31, and Time Now = 1/1/2006, the date result returned will be {12/31/2006}.

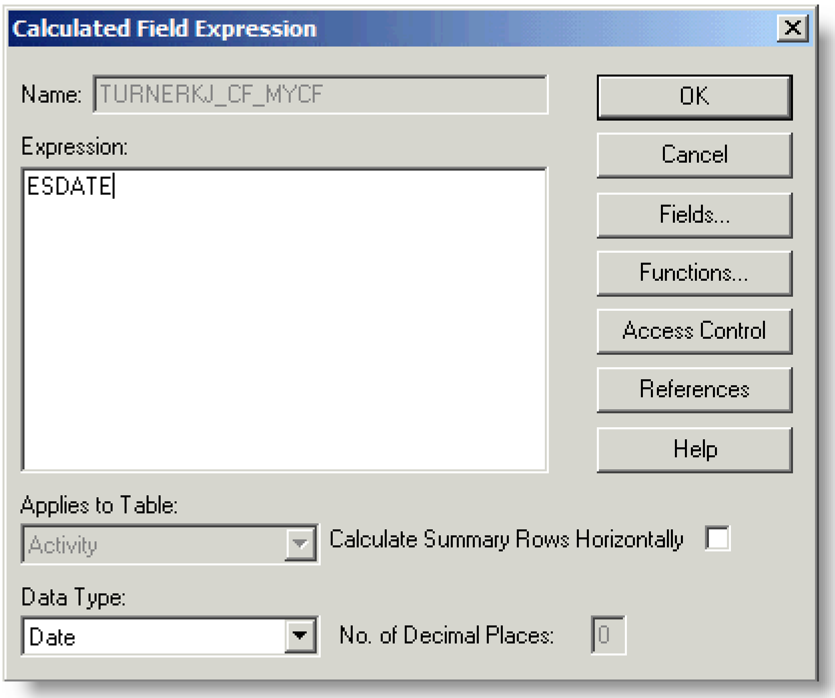
Functions Help dialog

Users frequently need to perform calculations based on dates, so let's begin by working with OPP date functions. To start, let's create a new calculated field to get the Early Start date. No function is necessary at this point.

Create a new Calculated Field (CF)

1. Go to the **Tools tab**, Custom Fields group and click **Calculated Fields**.
2. Click the **New** button
3. Give the new Calculated Field a name. Let's use <your ID>_CF_MYCF
4. Applies to Table: should already be **Activity**;
5. Set the Data Type: to **Date**
6. Click **OK**.
7. In the Expression: window, enter "**ESDATE**" (for Early Start date). Use the Fields…button to look up field names until you are familiar with them.

Tip: When you create new CLC objects use "CF" in the name for calculated fields, "F" for filters and "GE" for global edits. This simple naming convention makes it easy to tell one from the other and sorts all your like CLC objects together – all your calculated fields will be together, all your filters together and the same for global edits.



Name: TURNERKJ_CF_MYCF

Expression:

```
ESDATE
```

Applies to Table: Activity

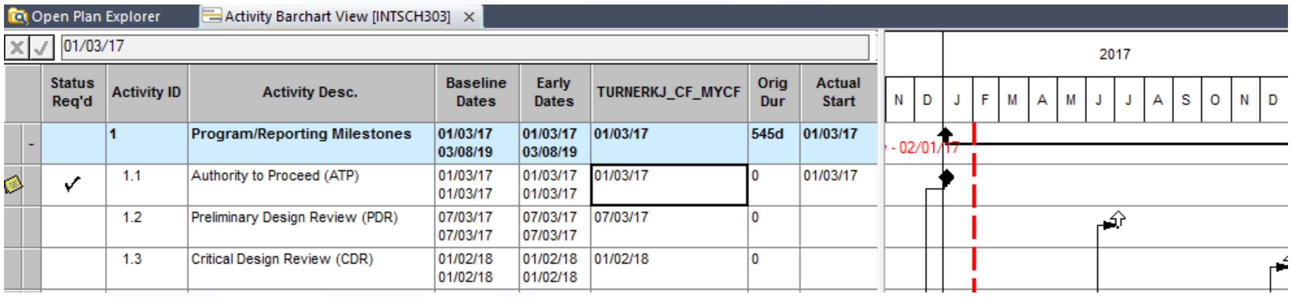
Calculate Summary Rows Horizontally ☐

Data Type: Date

No. of Decimal Places: 0

Calculated field expression to get Early Start date

Press OK to accept the calculated field expression; then insert the new calculated field into the view, as such...



	Status Req'd	Activity ID	Activity Desc.	Baseline Dates	Early Dates	TURNERKJ_CF_MYCF	Orig Dur	Actual Start
-		1	**Program/Reporting Milestones**	**01/03/17 03/08/19**	**01/03/17 03/08/19**	**01/03/17**	**545d**	**01/03/17**
	✓	1.1	Authority to Proceed (ATP)	01/03/17 01/03/17	01/03/17 01/03/17	01/03/17	0	01/03/17
		1.2	Preliminary Design Review (PDR)	07/03/17 07/03/17	07/03/17 07/03/17	07/03/17	0	
		1.3	Critical Design Review (CDR)	01/02/18 01/02/18	01/02/18 01/02/18	01/02/18	0	

The new calculated field will return the Early Start date as expected.

Next, let's format the date to a specific date format. The current default is MM/dd/yy which is set in Project Properties, Preferences. If you want this calculated field to display a different format, such as MM/dd/yy HH:mm, we can use the DATEFORMAT function to do that.

Frequently Used Functions

In Chapter 2 we'll illustrate several calculated field expressions focusing on the functions that are used most often. See Chapter 3 for all functions. At the end of the previous section, we created a new calculated field that returned the Early Start date (ESDATE). No function was used, we just grabbed the date out of the ESDATE field and displayed it. Now, let's use some common functions to manipulate that date, beginning with DATEFORMAT.

It's helpful to enter function key words in uppercase to make them easier to identify and read in a complex expression (i.e. instead of Dateformat, use DATEFORMAT).

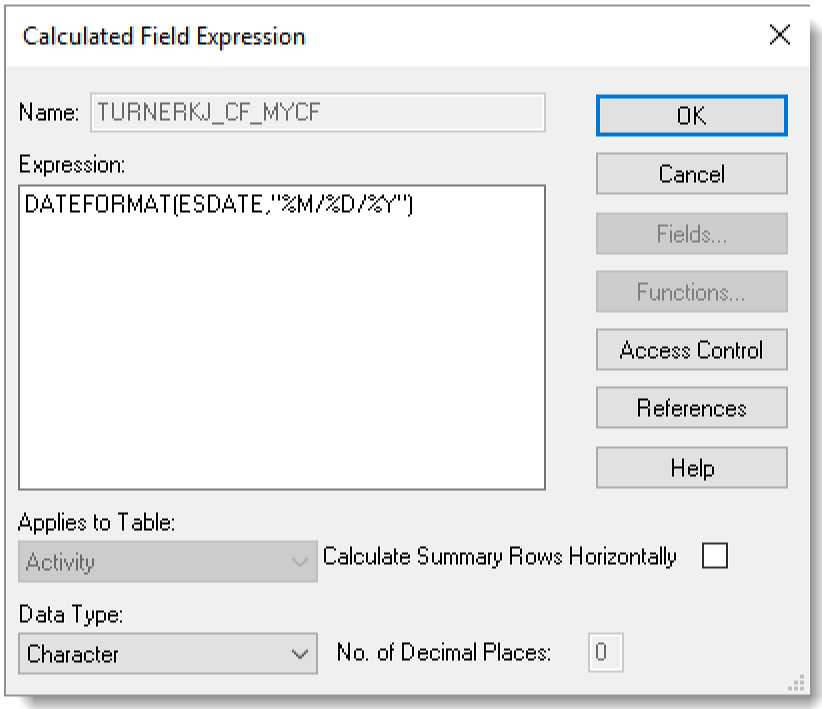
DATEFORMAT()

Forces a date value to a specified format based on the date format codes shown below. If you want to see the date in a specific format, use this function to do that.

Syntax: **DATEFORMAT(**<date field>**, <**date format code**>)**
Data Type: Character
Example: DATEFORMAT(ESDATE, "%M/%D/%Y") returns "10/15/21"
Example: DATEFORMAT(ESDATE, "%M/%D/%Y %H:%T") returns "10/15/21 08:00"
Example: DATEFORMAT(ESDATE, "%M-%D-%Y") returns "10-15-21"
Example: DATEFORMAT(ESDATE, "%D/%A/%Y") returns "15Oct21 08:00"

Date Format Codes

%T	Minutes	15
%H	Hour	8
%Z	AM / PM	A
%D	Day of Month	31
%V	Day of Month	Tuesday
%W	Day Abbreviation	Tues
%F	Single Character Day	T
%K	Week Number	52
%M	Numeric Month	11
%L	Month	November
%A	Month Abbreviation	Nov
%S	Single Character Month	N
%Q	Numeric Quarter	4
%C	Year	2002
%Y	2-digit Year	02



Calculated Field Expression

Name: TURNERKJ_CF_MYCF

Expression:

```
DATEFORMAT(ESDATE,"%M/%D/%Y")
```

Applies to Table: Activity

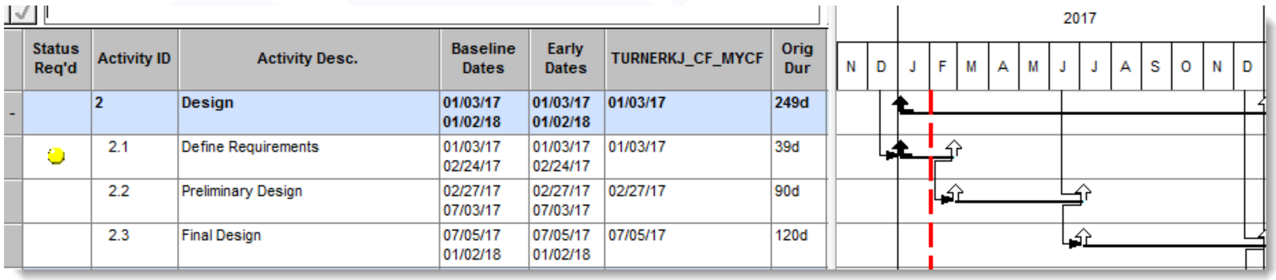
Calculate Summary Rows Horizontally ☐

Data Type: Character

No. of Decimal Places: 0

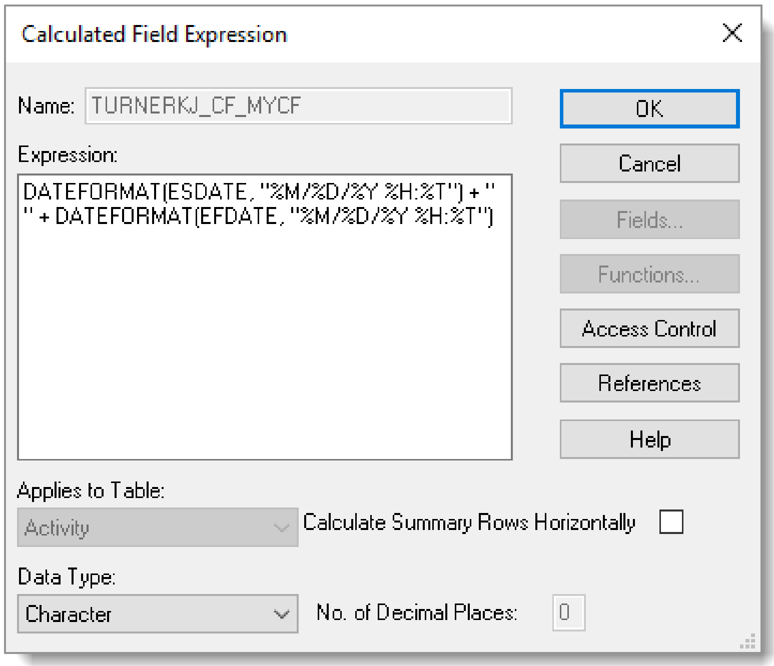
DATEFORMAT Function example

Note, you must change the Data Type to *Character* because it’s not read as a valid date any longer – it has been formatted. If you anticipate using this field result for a date calculation, do not use DATEFORMAT yet, do the calculation first; then format the final result for display.



Status Req'd	Activity ID	Activity Desc.	Baseline Dates	Early Dates	TURNERKJ_CF_MYCF	Orig Dur
	2	**Design**	**01/03/17 01/02/18**	**01/03/17 01/02/18**	**01/03/17**	**249d**
●	2.1	Define Requirements	01/03/17 02/24/17	01/03/17 02/24/17	01/03/17	39d
	2.2	Preliminary Design	02/27/17 07/03/17	02/27/17 07/03/17	02/27/17	90d
	2.3	Final Design	07/05/17 01/02/18	07/05/17 01/02/18	07/05/17	120d

Now let's display the Early Finish date in the same field with a space between them.
DATEFORMAT(ESDATE, "%M/%D/%Y %H:%T") + " " + DATEFORMAT(EFDATE, "%M/%D/%Y %H:%T")



Name: TURNERKJ_CF_MYCF

Expression:

```
DATEFORMAT(ESDATE, "%M/%D/%Y %H:%T") + "
" + DATEFORMAT(EFDATE, "%M/%D/%Y %H:%T")
```

Applies to Table: Activity

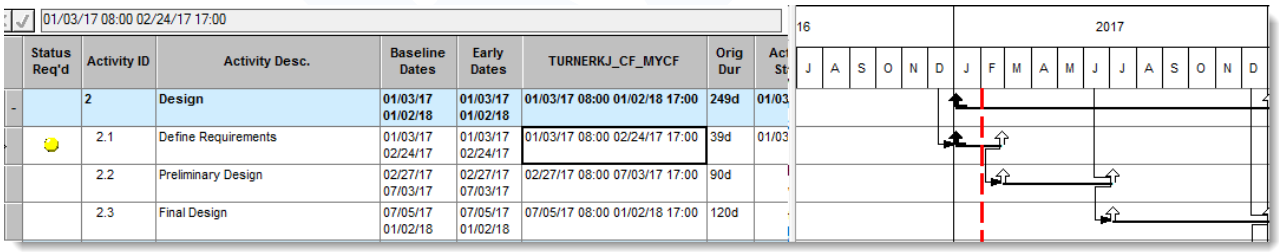
Calculate Summary Rows Horizontally ☐

Data Type: Character

No. of Decimal Places: 0

Concatenating ESDATE with EFDATE

Press OK to accept the calculated field expression.



01/03/17 08:00 02/24/17 17:00

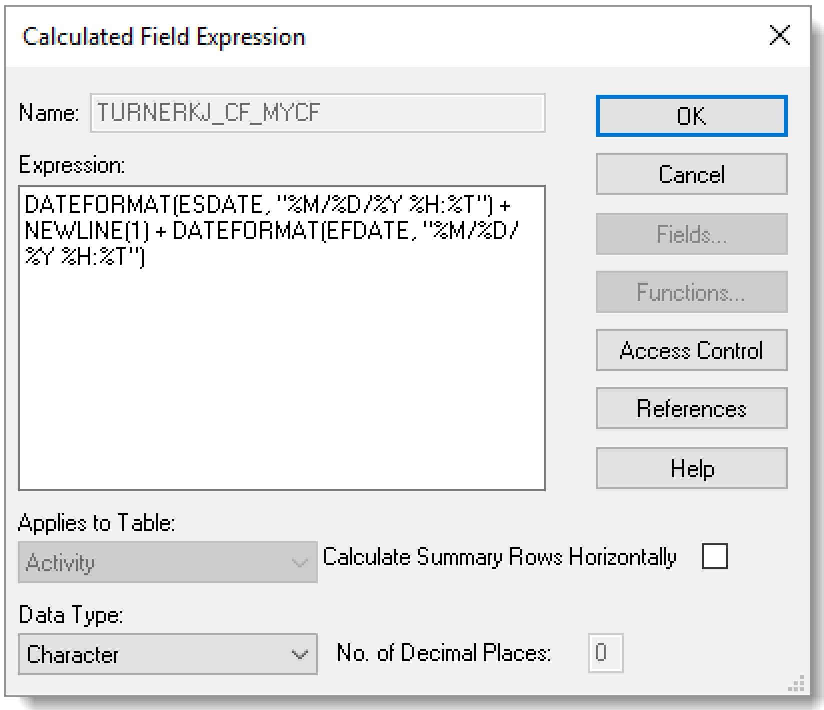
Status Req'd	Activity ID	Activity Desc.	Baseline Dates	Early Dates	TURNERKJ_CF_MYCF	Orig Dur	Act St
	2	**Design**	**01/03/17 01/02/18**	**01/03/17 01/02/18**	**01/03/17 08:00 01/02/18 17:00**	**249d**	**01/03**
●	2.1	Define Requirements	01/03/17 02/24/17	01/03/17 02/24/17	01/03/17 08:00 02/24/17 17:00	39d	01/03
	2.2	Preliminary Design	02/27/17 07/03/17	02/27/17 07/03/17	02/27/17 08:00 07/03/17 17:00	90d	
	2.3	Final Design	07/05/17 01/02/18	07/05/17 01/02/18	07/05/17 08:00 01/02/18 17:00	120d	

The result is a little hard to read as one line, so let's use the NEWLINE function to wrap the Early Finish date to the second row.

NEWLINE()

Wraps to the next line. (Be sure the field is set to wrap data in column and heading.)

Syntax: **NEWLINE(1)**
Example: DATEFORMAT(ESDATE, "%M/%D/%Y %H:%T") **+ NEWLINE(1) +** DATEFORMAT(EFDATE, "%M/%D/%Y %H:%T")


Calculated Field Expression

Name: TURNERKJ_CF_MYCF

Expression:
```
DATEFORMAT(ESDATE, "%M/%D/%Y %H:%T") +
NEWLINE(1) + DATEFORMAT(EFDATE, "%M/%D/
%Y %H:%T")
```

Applies to Table: Activity

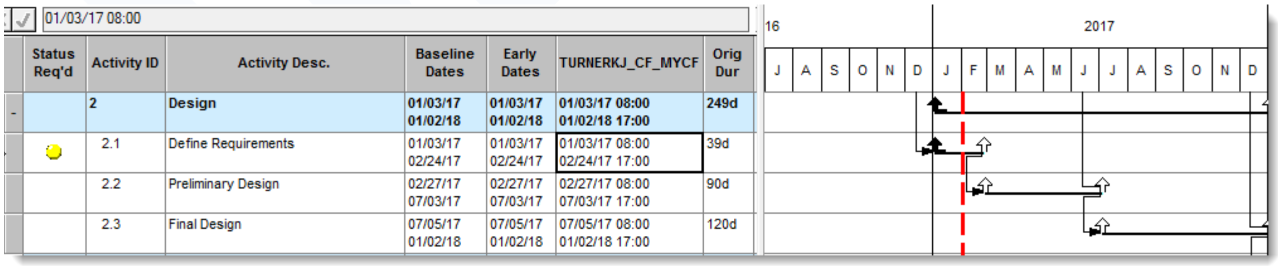
Calculate Summary Rows Horizontally ☐

Data Type: Character

No. of Decimal Places: 0

DATEFORMAT with NEWLINE(1) to wrap text

Press OK to accept the calculated field expression.


01/03/17 08:00

Status Req'd	Activity ID	Activity Desc.	Baseline Dates	Early Dates	TURNERKJ_CF_MYCF	Orig Dur
	2	**Design**	**01/03/17 01/02/18**	**01/03/17 01/02/18**	**01/03/17 08:00 01/02/18 17:00**	**249d**
●	2.1	Define Requirements	01/03/17 02/24/17	01/03/17 02/24/17	01/03/17 08:00 02/24/17 17:00	39d
	2.2	Preliminary Design	02/27/17 07/03/17	02/27/17 07/03/17	02/27/17 08:00 07/03/17 17:00	90d
	2.3	Final Design	07/05/17 01/02/18	07/05/17 01/02/18	07/05/17 08:00 01/02/18 17:00	120d

NEWLINE(1) wraps 1 line; NEWLINE(2) wraps to 2 lines (double-spaces); NEWLINE(3) triple spaces...etc.

Switching gears a bit but still working with dates, let's calculate the difference between Early Start and Early Finish. For this, we use the DATEDIFFERENCE function.

Tip/Trick with the NEWLINE function

This calculated field expression checks conditions and displays a value; then wraps to the next line and if it meets another condition, it displays another label and wraps to the next line.

Original expression:

```
IIF(AFDATE="" and INTSCH_CPncpDPndp_inRMAR = [True], "CP/DP","") + NEWLINE (1) +
IIF(AFDATE="" and INTSCH_HO <> "", "HO: " + INTSCH_HO,"") + NEWLINE (1) +
IIF(AFDATE="" and TOTALFLOAT <= |0d|,"TF<=0d","") + NEWLINE (1) +
IIF(AFDATE="" and SV < 0.00 or SPI < 0.00,"SV or SPI <0.00","") + NEWLINE (1) +
IIF(AFDATE="" and C31 <> "","Risk Assoc","")
```

However, it leaves a blank line for each row even if the activity doesn't meet the criteria. How do you get it not to wrap if the condition does *not* exist?

Rewritten as:

```
IIF(AFDATE="" and INTSCH_CPncpDPndp_inRMAR = [True], "CP/DP" +NEWLINE(1),"") +
IIF(AFDATE="" and INTSCH_HO <> "","HO: " + INTSCH_HO +NEWLINE(1),"") +
IIF(AFDATE="" and TOTALFLOAT <= |0d|,"TF<=0d" +NEWLINE(1),"") +
IIF(AFDATE="" and (SV < 0.00 or SPI < 0.00),"SV or SPI <0.00" +NEWLINE(1),"") +
IIF(AFDATE="" and C31 <> "","Risk Assoc" +NEWLINE(1),"")
```

The key is to move the NEWLINE(1) function *inside the statement* so that it only wraps to the new line if the previous row exists (so it only wraps if it needs to).

Another tip for wrapping uses the STRTRN function (discussed later). Using the GET_ASSGNS function we can get the RES and Labor type. For example, the expression:

```
GET_ASSGNS("Res_ID|Resource_Class")
Returns:   GT.01.011,Labor;GT.01.013,Labor;GT.01.012,Labor

STRTRAN(GET_ASSGNS("Res_ID|Resource_Class"),";",NEWLINE(1))
Returns:   GT.01.011,Labor
           GT.01.013,Labor
           GT.01.012,Labor
```

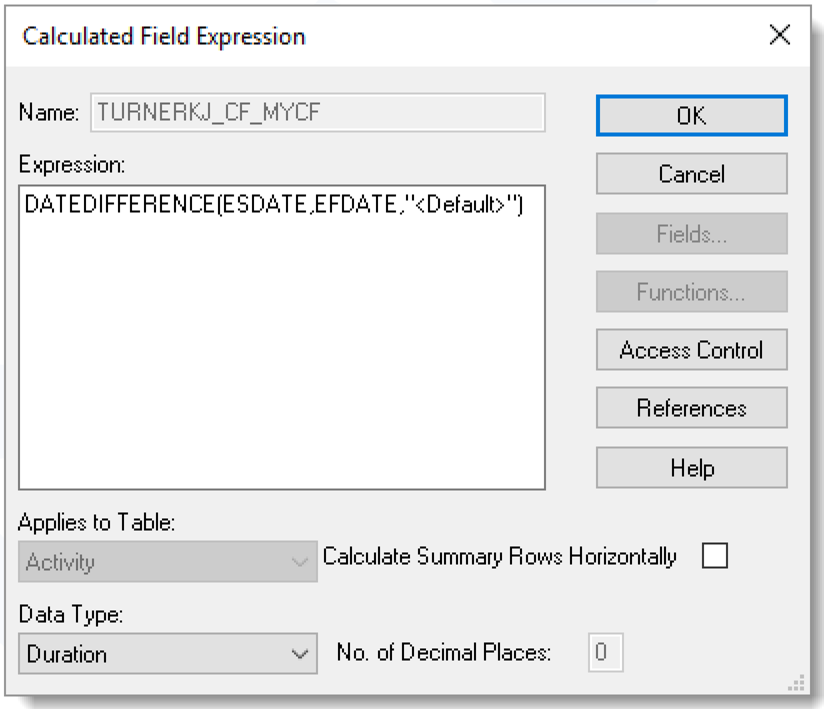
The STRTRAN function replaces ";" with NEWLINE(1) causing the values to wrap.

DATEDIFFERENCE()
Returns the Duration difference between two dates based on a calendar. It returns a *Duration* value so the data type should be set to Duration.

Syntax: **DATEDIFFERENCE(**Date1**,**Date2**,**Calendar**)**
Data Type: Duration
Example: **DATEDIFFERENCE**(ESDATE**,**EFDATE**,**"<Default>"**)**

Recommendation: Use "CLH_ID" to refer to the activity calendar assigned to the task in the Calendar ID field (with no quotation marks around the value).

Example: **DATEDIFFERENCE**(ESDATE**,**EFDATE**,**CLH_ID**)**


Calculated Field Expression

Name: TURNERKJ_CF_MYCF

Expression:

```
DATEDIFFERENCE(ESDATE,EFDATE,"<Default>")
```

Applies to Table: Activity

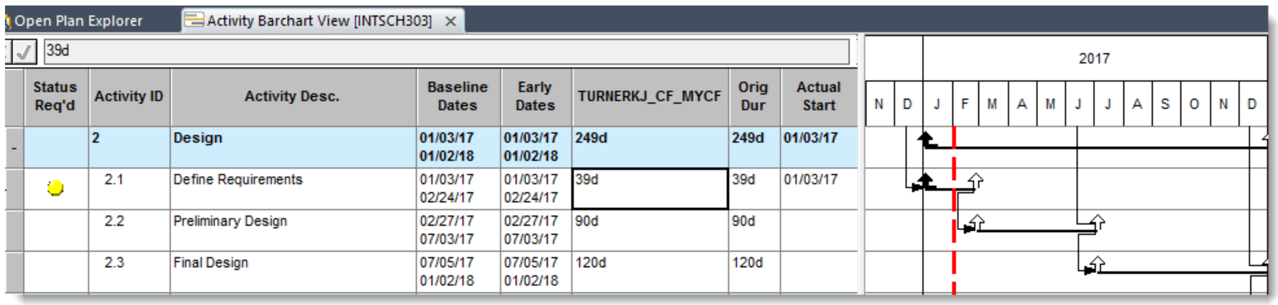
Calculate Summary Rows Horizontally ☐

Data Type: Duration

No. of Decimal Places: 0

DATEDIFFERENCE Function

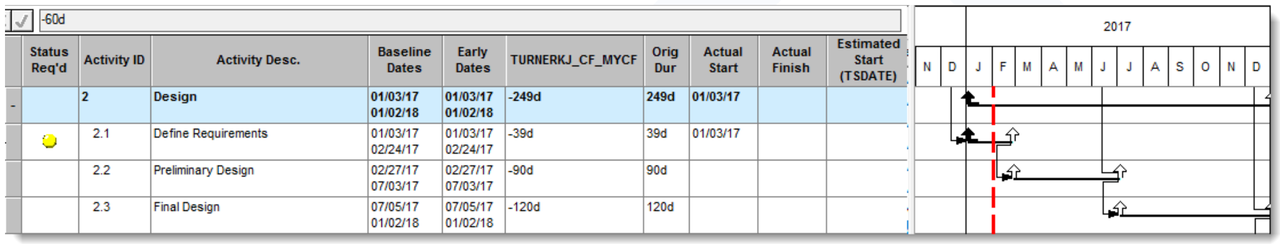
Press OK to accept the calculated field expression. Notice that our calculated field now returns the duration difference between Early Start and Early Finish (39d) which is exactly equal to the Original Duration (as it should be).


Open Plan Explorer
Activity Barchart View [INTSCH303]
2017
N D J F M A M J J A S O N D

39d

Status Req'd	Activity ID	Activity Desc.	Baseline Dates	Early Dates	TURNERKJ_CF_MYCF	Orig Dur	Actual Start
	2	**Design**	**01/03/17 01/02/18**	**01/03/17 01/02/18**	**249d**	**249d**	**01/03/17**
●	2.1	Define Requirements	01/03/17 02/24/17	01/03/17 02/24/17	39d	39d	01/03/17
	2.2	Preliminary Design	02/27/17 07/03/17	02/27/17 07/03/17	90d	90d	
	2.3	Final Design	07/05/17 01/02/18	07/05/17 01/02/18	120d	120d	

If you reverse the order of the dates in the DATEDIFFERENCE function, the result will be a negative value. For example, in this expression we switch positions of ESDATE and EFDATE: **DATEDIFFERENCE(**EFDATE**,**ESDATE**,**"<Default>"**)**. The result will be -39d for activity 2.1 instead of +39d.

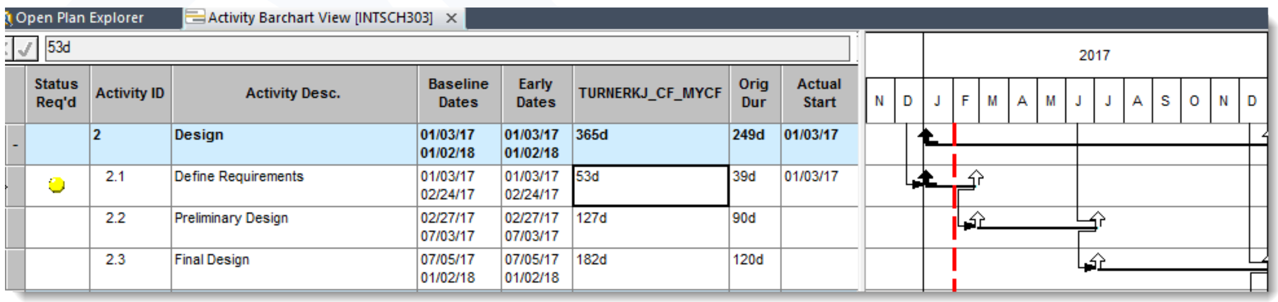


-60d

Status Req'd	Activity ID	Activity Desc.	Baseline Dates	Early Dates	TURNERKJ_CF_MYCF	Orig Dur	Actual Start	Actual Finish	Estimated Start (TSDATE)
	2	**Design**	**01/03/17 01/02/18**	**01/03/17 01/02/18**	**-249d**	**249d**	**01/03/17**		
●	2.1	Define Requirements	01/03/17 02/24/17	01/03/17 02/24/17	-39d	39d	01/03/17		
	2.2	Preliminary Design	02/27/17 07/03/17	02/27/17 07/03/17	-90d	90d			
	2.3	Final Design	07/05/17 01/02/18	07/05/17 01/02/18	-120d	120d			

Doing this might be useful when comparing Early Finish to Baseline Finish, for example. A negative value would indicate *late*; whereas a positive value *early*.
DATEDIFFERENCE(EFDATE**,**BFDATE**,**"<Default>"**)**

Now, let's edit the expression to change the calendar. Use the expression:
DATEDIFFERENCE(ESDATE**,**EFDATE**,**"7 DAY - NO HOLIDAYS"**)**


Open Plan Explorer
Activity Barchart View [INTSCH303]
2017
N D J F M A M J J A S O N D

53d

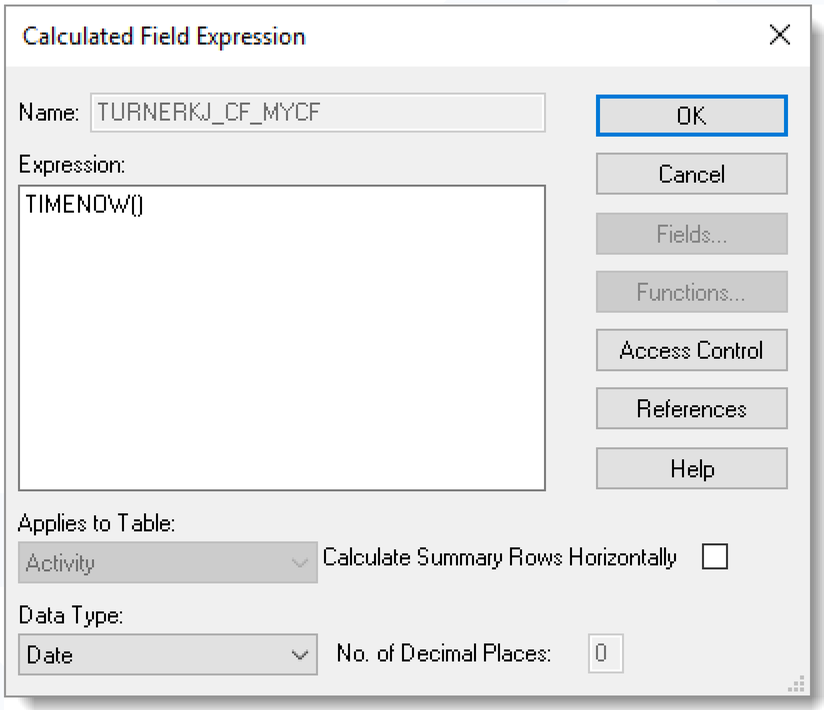
Status Req'd	Activity ID	Activity Desc.	Baseline Dates	Early Dates	TURNERKJ_CF_MYCF	Orig Dur	Actual Start
	2	**Design**	**01/03/17 01/02/18**	**01/03/17 01/02/18**	**365d**	**249d**	**01/03/17**
●	2.1	Define Requirements	01/03/17 02/24/17	01/03/17 02/24/17	53d	39d	01/03/17
	2.2	Preliminary Design	02/27/17 07/03/17	02/27/17 07/03/17	127d	90d	
	2.3	Final Design	07/05/17 01/02/18	07/05/17 01/02/18	182d	120d	

Based on a 7-day calendar, the result shows 53d (calendar days) for Activity 2.1 instead of 39d (business days).

There is no tolerance for typos. You must enter the name of the calendar exactly correct or the expression will ignore your calendar entry and use the <Default> (or whatever the default calendar is set to in Project Properties, Preferences.)

TIMENOW()
Returns the project Timenow() date.
Syntax: **TIMENOW()**
Data Type: Date

Often times, a user will need to reference the Timenow date in a function. This can easily be accomplished using the function TIMENOW(). Creating a calculated field with the expression: TIMENOW() will return the current Timenow date set for the project.


Calculated Field Expression

Name: TURNERKJ_CF_MYCF

Expression:

TIMENOW()

Applies to Table: Activity

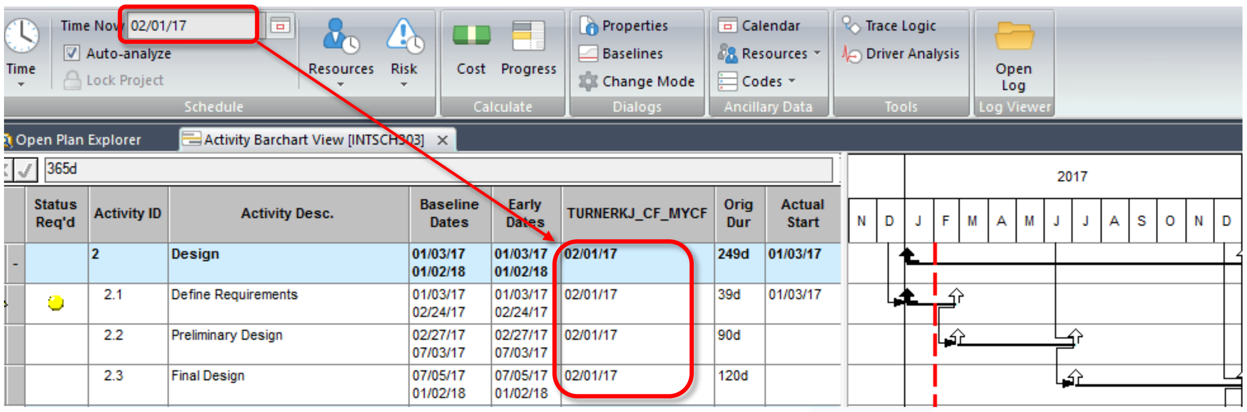
Calculate Summary Rows Horizontally ☐

Data Type: Date

No. of Decimal Places: 0

TIMENOW() Function

Press OK to accept the calculated field expression.



Status Req'd	Activity ID	Activity Desc.	Baseline Dates	Early Dates	TURNERKJ_CF_MYCF	Orig Dur	Actual Start
	2	**Design**	**01/03/17 01/02/18**	**01/03/17 01/02/18**	**02/01/17**	**249d**	**01/03/17**
●	2.1	Define Requirements	01/03/17 02/24/17	01/03/17 02/24/17	02/01/17	39d	01/03/17
	2.2	Preliminary Design	02/27/17 07/03/17	02/27/17 07/03/17	02/01/17	90d	
	2.3	Final Design	07/05/17 01/02/18	07/05/17 01/02/18	02/01/17	120d	

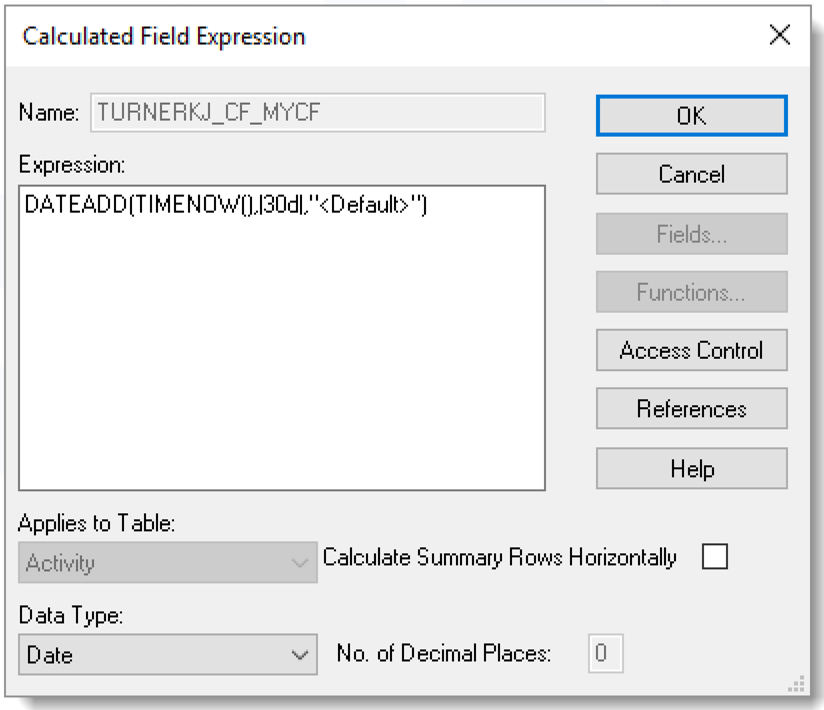
DATEADD()
Use the DATEADD function anytime you have a date and want to add time (duration) to it to determine what date that would be.

Syntax: DATEADD(Date,Duration,Calendar) Returns a *Date* value.
Data Type: Date
Example: **DATEADD(**TIMENOW()**,**|30d|**,**"<Default>"**)**
Example: **DATEADD(**TIMENOW()**,**|30d|**,**CLH_ID**)**
Example: **DATEADD(**TIMENOW()**,**|30d|**,**"5 DAY – BOEING HOLIDAYS"**)**

CLH_ID instructs it to use the specific calendar assigned to the activity in the CALENDAR_ID field. Otherwise, include the specific calendar name in double-quotes.

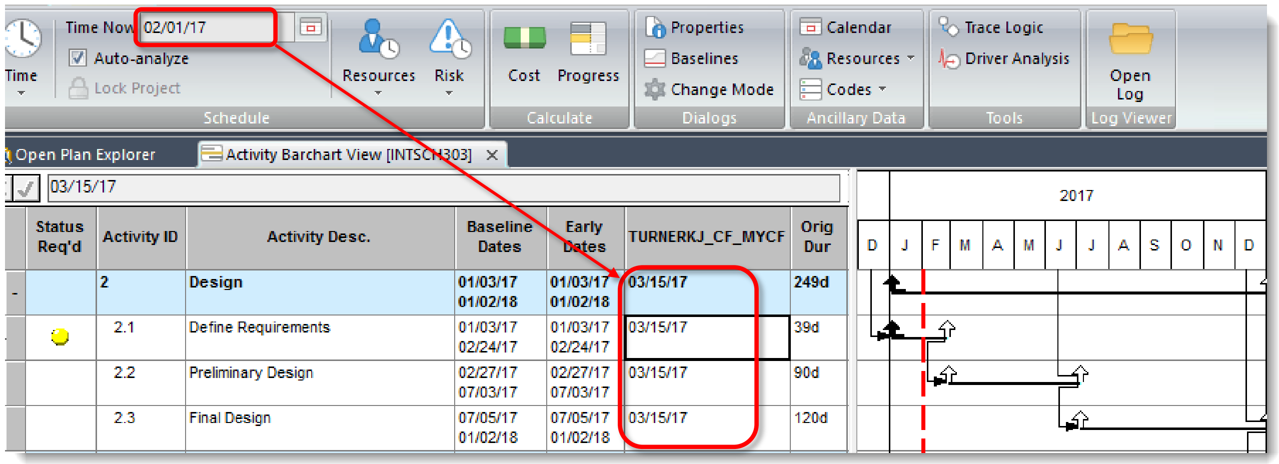
Practical application - Assume you want to determine a 30d window (lookahead) from Timenow. We can combine the DATEADD and TIMENOW functions together.

So in our example, the expression would be: **DATEADD(**TIMENOW()**,|**30d**|,**"<Default>"**)**
Notice that any duration value in the expression must be enclosed in piping symbols "|" or you'll get an error.


Calculated Field Expression
Name: TURNERKJ_CF_MYCF
Expression:
```
DATEADD(TIMENOW(),|30d|,"<Default>")
```
Applies to Table: Activity
Calculate Summary Rows Horizontally ☐
Data Type: Date
No. of Decimal Places: 0

DATEADD Function

Press OK to accept the calculated field expression.



Time Now 02/01/17

03/15/17

Status Req'd	Activity ID	Activity Desc.	Baseline Dates	Early Dates	TURNERKJ_CF_MYCF	Orig Dur
	2	Design	01/03/17 01/02/18	01/03/17 01/02/18	03/15/17	249d
●	2.1	Define Requirements	01/03/17 02/24/17	01/03/17 02/24/17	03/15/17	39d
	2.2	Preliminary Design	02/27/17 07/03/17	02/27/17 07/03/17	03/15/17	90d
	2.3	Final Design	07/05/17 01/02/18	07/05/17 01/02/18	03/15/17	120d

Timenow plus 30 business days

Note that because we used a business day based calendar (<Default>), the expression is adding 30 *business days* to Timenow which will be longer than a physical month. To add 30 *calendar days*, edit the expression to use a 7-day calendar such as “7 DAY - NO HOLIDAYS” or change the duration used in your expression from 30d to 20d.

Since the data type used for this field is a *Date*, we can reference the value in this field for other calculation comparisons. For example, you can compare your Baseline Start (or Baseline Finish) date to see if it falls between Timenow and this 30d window.

While we haven't discussed the IIF function yet, an expression like the following will return "Yes" if the activity's Baseline Finish falls within this 30d window.

IIF(BFDATE>TIMENOW() and BFDATE<DATEADD(TIMENOW(),|30d|,"7 DAY - NO HOLIDAYS"),"Yes","No").

You can then filter on or highlight the activities that return “Yes” as being in the 30d window.

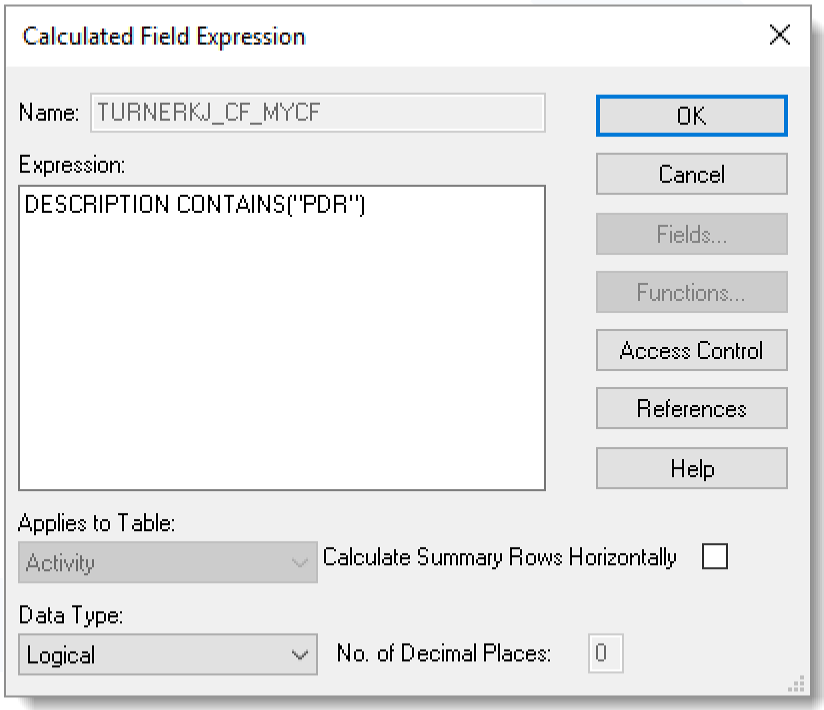
Since there is no DATESUBTRACT function, if you want to subtract a duration from a date; just multiply the duration value by -1. So in our example, to subtract 30d from time now, the expression would be **DATEADD(**TIMENOW(),|30d|*-1, "<Default>"**)**.

CONTAINS()

The CONTAINS function is particularly useful when you want to determine if a specific string of text is present in another text value.

Syntax: <field name> **CONTAINS(**“text string”**)**
Data Type: Logical
Example: DESCRIPTION **CONTAINS(**"PDR"**)**

For example, given the activity description “Preliminary Design Review (PDR)”, the expression: DESCRIPTION **CONTAINS(**"PDR"**)** returns “True”.



Calculated Field Expression

Name: TURNERKJ_CF_MYCF

Expression:

```
DESCRIPTION CONTAINS("PDR")
```

Applies to Table: Activity

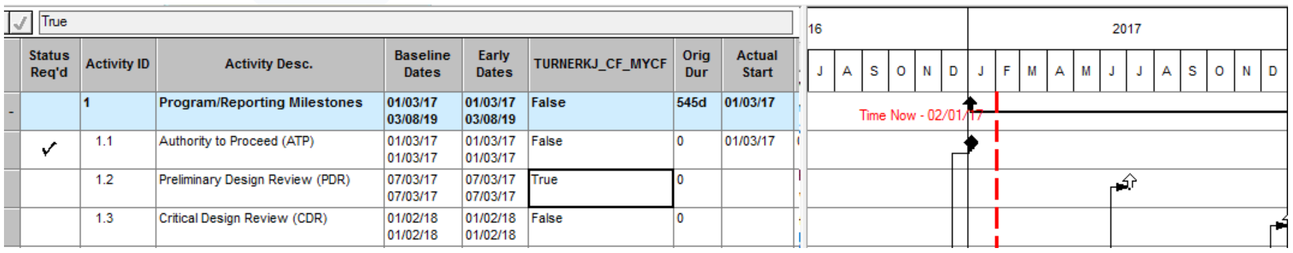
Calculate Summary Rows Horizontally ☐

Data Type: Logical

No. of Decimal Places: 0

CONTAINS Function

Press OK to accept the calculated field expression.



True

	Status Req'd	Activity ID	Activity Desc.	Baseline Dates	Early Dates	TURNERKJ_CF_MYCF	Orig Dur	Actual Start
-		1	Program/Reporting Milestones	01/03/17 03/08/19	01/03/17 03/08/19	False	545d	01/03/17
	✓	1.1	Authority to Proceed (ATP)	01/03/17 01/03/17	01/03/17 01/03/17	False	0	01/03/17
		1.2	Preliminary Design Review (PDR)	07/03/17 07/03/17	07/03/17 07/03/17	True	0	
		1.3	Critical Design Review (CDR)	01/02/18 01/02/18	01/02/18 01/02/18	False	0	

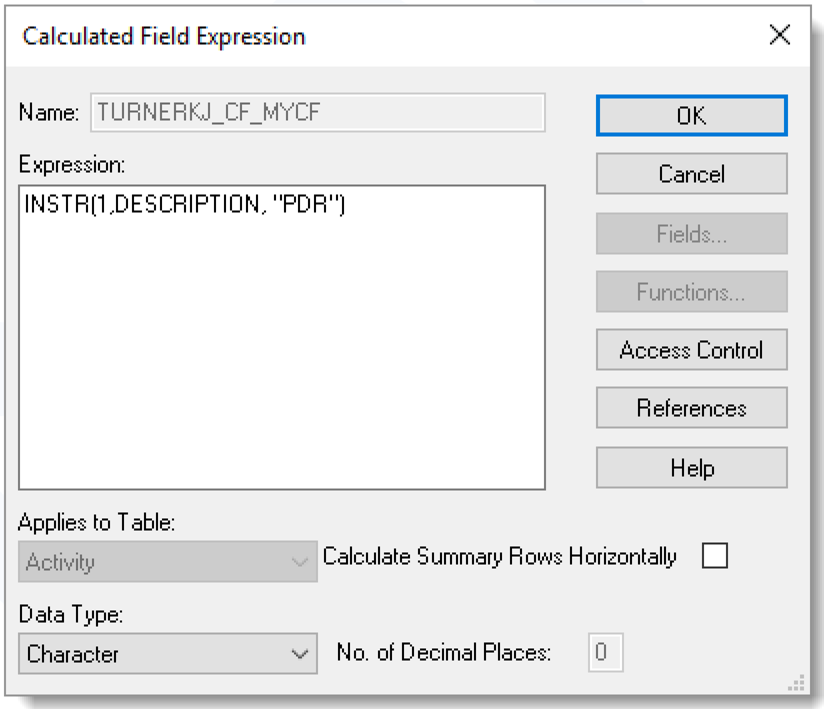
As an alternative, the expression “PDR” **$** DESCRIPTION also returns “True” using the **$** operator.

You can also use "NOT CONTAINS" to return True if the string *does not contain* the specified search string.

INSTR()
The INSTR function is a little more precise than the CONTAINS function. The INSTR function allows us to test for the presence of a character or string of characters within a field, and return the starting position of that string if found. For example, to find the occurrence of "PDR" in the activity Description field use the following:

Syntax: **INSTR(**<starting position>**,**<field>**,**"text string"**)**
Data Type: Integer
Example: **INSTR(**1,DESCRIPTION, "PDR"**)**

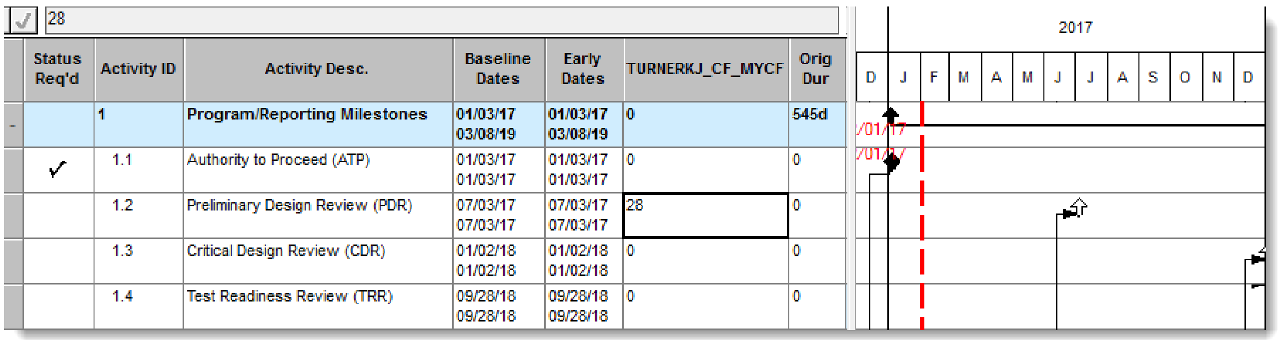
Given the activity description "Preliminary Design Review (PDR) " the following expression would return, "28". **INSTR(**1,DESCRIPTION, "PDR"**)**. If not found, the function returns "**0**".



Name: TURNERKJ_CF_MYCF
Expression:
INSTR(1,DESCRIPTION, "PDR")
Applies to Table: Activity
Calculate Summary Rows Horizontally ☐
Data Type: Character
No. of Decimal Places: 0

INSTR Function

Press OK to accept the calculated field expression.



28

	Status Req'd	Activity ID	Activity Desc.	Baseline Dates	Early Dates	TURNERKJ_CF_MYCF	Orig Dur
-		**1**	**Program/Reporting Milestones**	**01/03/17 03/08/19**	**01/03/17 03/08/19**	**0**	**545d**
	✓	1.1	Authority to Proceed (ATP)	01/03/17 01/03/17	01/03/17 01/03/17	0	0
		1.2	Preliminary Design Review (PDR)	07/03/17 07/03/17	07/03/17 07/03/17	28	0
		1.3	Critical Design Review (CDR)	01/02/18 01/02/18	01/02/18 01/02/18	0	0
		1.4	Test Readiness Review (TRR)	09/28/18 09/28/18	09/28/18 09/28/18	0	0

Beginning at the first character in the field, look for the string "PDR". If it finds it, the function returns the character position of the first letter of the string – in this case, activity 1.2 has the string PDR beginning in position 28. And unfortunately, yes, the search string is case-sensitive. **INSTR(**1,DESCRIPTION, "pdr"**)** would return “0”.

One way around this case sensitive issue would be to use another function to “upper case” the Description field first; then test for "PDR". If the Description field contained "Preliminary Design Review (pdr)", the following calculated field expression would get any occurrence of the string PDR whether upper, lower or mixed case.

INSTR(1,**UPPER(**DESCRIPTION**)**, "PDR"**)**

The function **UPPER(**<field>**)** converts all text in the field to upper case, thus eliminating the case-sensitive issue.

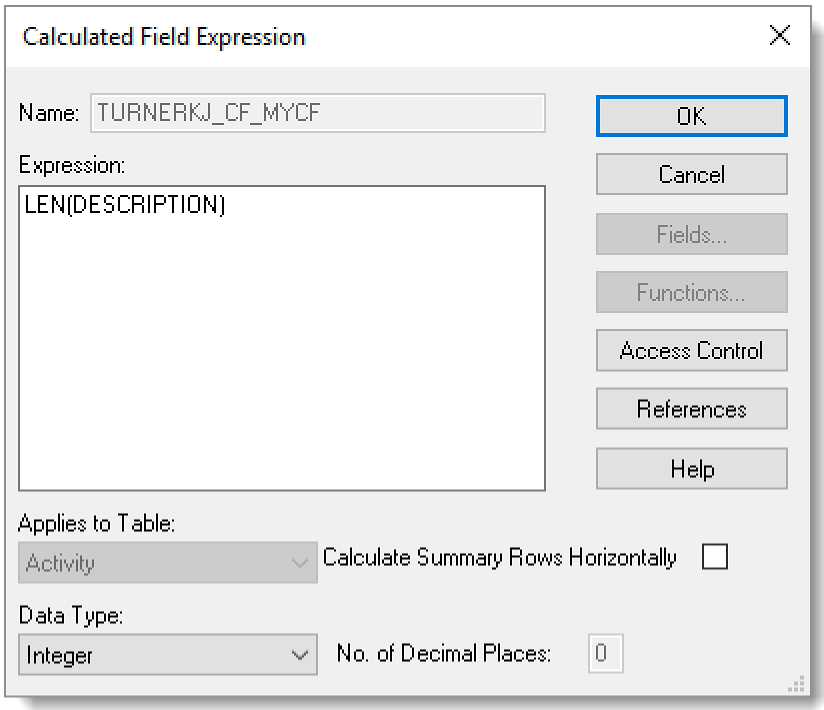
LEN()

The LEN function will count the number of characters in a string. Given the activity description "Preliminary Design Review (PDR) " the following expression would return, "31".

Syntax: LEN(<field>)

Date Type: Integer

Example: **LEN(**DESCRIPTION**)**



Calculated Field Expression

Name: TURNERKJ_CF_MYCF

Expression:

```
LEN(DESCRIPTION)
```

Applies to Table: Activity

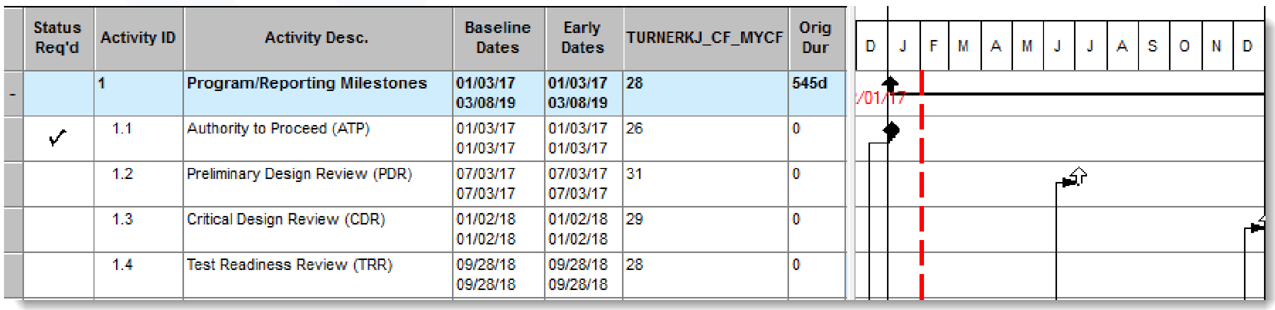
Calculate Summary Rows Horizontally ☐

Data Type: Integer

No. of Decimal Places: 0

LEN Function

Press OK to accept the calculated field expression. The character length of the Description field are returned (including spaces).



Status Req'd	Activity ID	Activity Desc.	Baseline Dates	Early Dates	TURNERKJ_CF_MYCF	Orig Dur
	1	**Program/Reporting Milestones**	**01/03/17 03/08/19**	**01/03/17 03/08/19**	**28**	**545d**
✓	1.1	Authority to Proceed (ATP)	01/03/17 01/03/17	01/03/17 01/03/17	26	0
	1.2	Preliminary Design Review (PDR)	07/03/17 07/03/17	07/03/17 07/03/17	31	0
	1.3	Critical Design Review (CDR)	01/02/18 01/02/18	01/02/18 01/02/18	29	0
	1.4	Test Readiness Review (TRR)	09/28/18 09/28/18	09/28/18 09/28/18	28	0

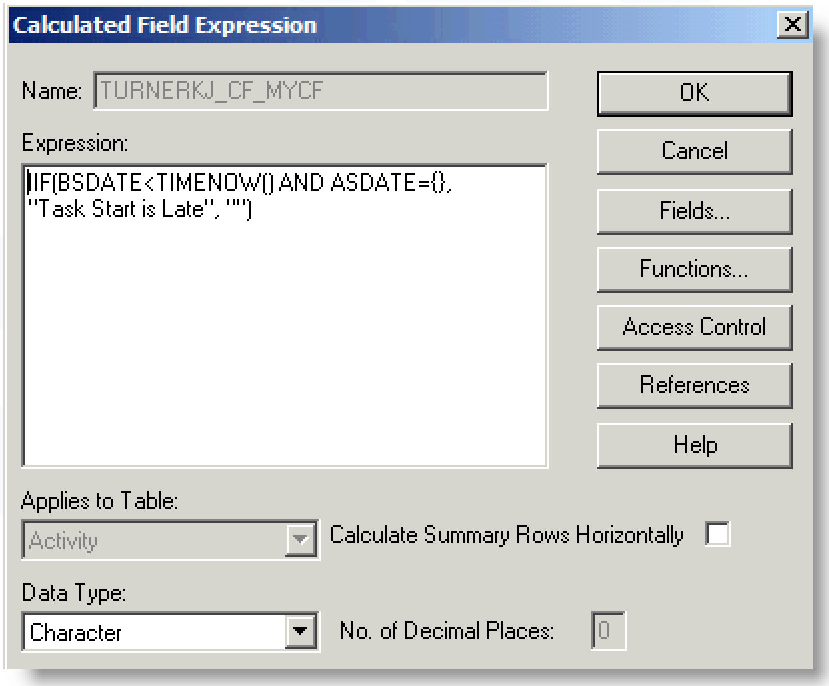
At first, you may not consider this function particularly useful, however when combined with other string functions, it enables you to strip out selected text from other fields. More on this later.

IIF()
By far, one of the most powerful functions is the Immediate IF function because it enables you to test for conditions and return results based on the condition that exists. These can be a simple, "If this condition is true, then do this, else do that" or they can be very complex with multiple layers of embedded IIF functions.

Syntax: **IIF(**<Condition>**,**<Do This if True>**,**<Do This if False>**)**
Data Type: Any

Practical example, you want to check to see if an activity is late to the Baseline Start. That means the condition that has to be met is the Baseline Start date is earlier than Timenow AND the the Actual Start date is empty. In this expression, we will combine the TIMENOW() function with the AND operator. The expression would look like this...

IIF(BSDATE<**TIMENOW() AND** ASDATE={}**,** "Task Start is Late","")
Read as: If Baseline Start is earlier than Timenow and Actual Start is empty, then display "Task Start is Late", else show nothing. Use the "AND" as an operator to test both conditions that must be met.



Name: TURNERKJ_CF_MYCF

Expression:

```
IIF(BSDATE<TIMENOW() AND ASDATE={},
"Task Start is Late", "")
```

Applies to Table: Activity

Calculate Summary Rows Horizontally ☐

Data Type: Character

No. of Decimal Places: 0

IIF Function

Now let's get a little fancier... in our first example, we used the IIF to make a decision with two (2) possible outcomes. What if there were more than two possible outcomes? What if we wanted to check if not only the Start was late, but also the Finish? In our first example we only checked the Baseline Start, but it could also be late if the Baseline Finish is late.

Let's write the same expression now, but check for the Baseline Finish being late. The expression would look like this:

```
IIF(BFDATE<TIMENOW() and AFDATE={},"Task Finish is Late","")
```

Now, we can combine the two IIF statements into a single expression called a "Nested IIF" statement. Let's look at the two statements independently first.

Statement 1

```
IIF(BSDATE<TIMENOW() and ASDATE={},"Task is Start Late","")
```

Statement 2

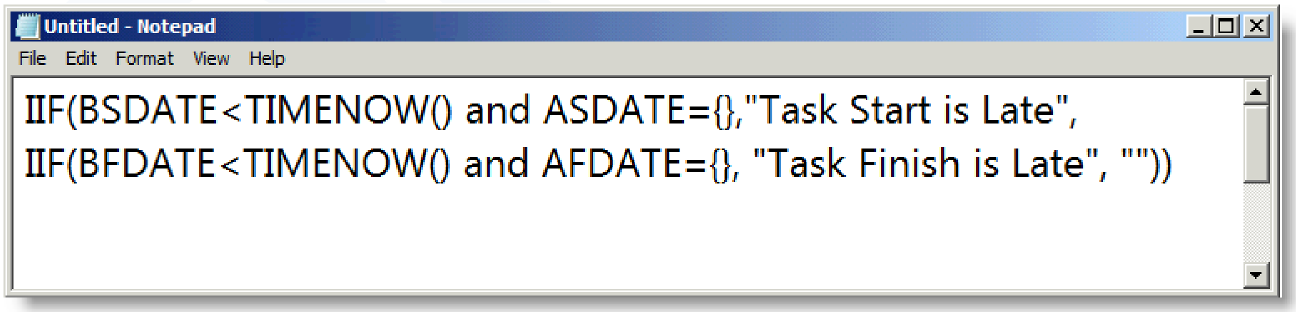
```
IIF(BFDATE<TIMENOW() and AFDATE={},"Task Finish is Late","")
```

Copy all of Statement 2 to your clipboard; then paste it into the False part of Statement 1 – highlighted in yellow in Statement 1 to make it easier to see. The combined (nested) statement would look like this:

```
IIF(BSDATE<TIMENOW() and ASDATE={},"Task is Start Late",
IIF(BFDATE<TIMENOW() and AFDATE={},"Task Finish is Late",""))
```

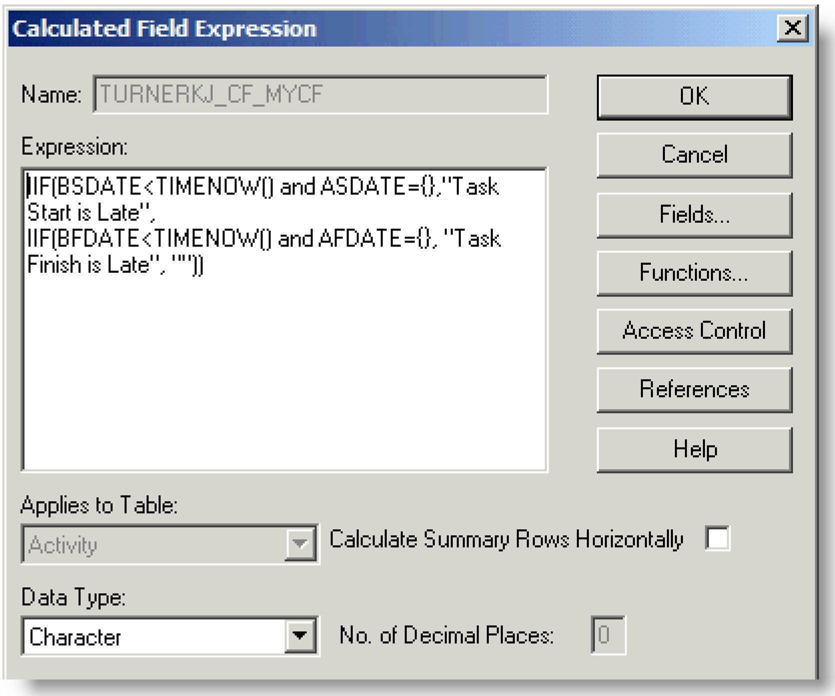
The statement is easier to read if formatted like this as two lines:

```
IIF(BSDATE<TIMENOW() and ASDATE={},"Task is Start Late",
 <with an imaginary "else" implied here>
IIF(BFDATE<TIMENOW() and AFDATE={},"Task Finish is Late",""))
```



```
IIF(BSDATE<TIMENOW() and ASDATE={},"Task Start is Late",
IIF(BFDATE<TIMENOW() and AFDATE={}, "Task Finish is Late", ""))
```

Embedded IIF Expression written in Notepad



Name: TURNERKJ_CF_MYCF

Expression:

```
IIF(BSDATE<TIMENOW() and ASDATE={},"Task
Start is Late",
IIF(BFDATE<TIMENOW() and AFDATE={}, "Task
Finish is Late", ""))
```

Applies to Table: Activity

Calculate Summary Rows Horizontally ☐

Data Type: Character

No. of Decimal Places: 0

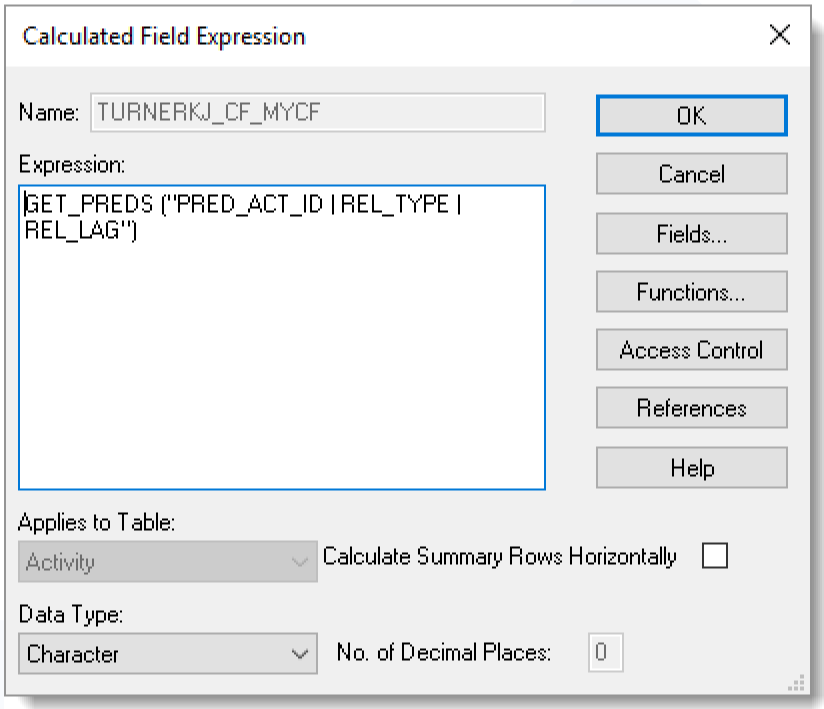
Embedded IIF Function

These can be very tedious to "think through" to get the right logic stream to correctly isolate and return the desired results. To facilitate the construction of these, try to the "Nested IIF Builder" tool in Excel.

GET_PREDS()
The GET_PREDS function allows you to list the predecessors for each activity

Syntax: GET_PREDS(<fieldname1> |<fieldname2> |<fieldname3>...)
Data Type: Character
Example: **GET_PREDS (**"PRED_ACT_ID **|** REL_TYPE **|** REL_LAG"**)**

Note the placement of double quotation marks (**"**) around the field names and the piping symbols (**|**) separating each relationship field.



Name: TURNERKJ_CF_MYCF

Expression:

```
GET_PREDS ("PRED_ACT_ID | REL_TYPE |
REL_LAG")
```

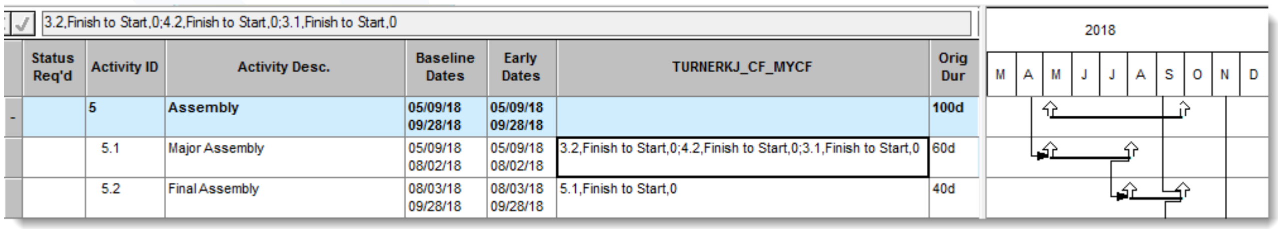
Applies to Table: Activity

Calculate Summary Rows Horizontally ☐

Data Type: Character

No. of Decimal Places: 0

GET_PREDS Function



3.2,Finish to Start,0;4.2,Finish to Start,0;3.1,Finish to Start,0

Status Req'd	Activity ID	Activity Desc.	Baseline Dates	Early Dates	TURNERKJ_CF_MYCF	Orig Dur
	5	**Assembly**	**05/09/18 09/28/18**	**05/09/18 09/28/18**		**100d**
	5.1	Major Assembly	05/09/18 08/02/18	05/09/18 08/02/18	3.2,Finish to Start,0;4.2,Finish to Start,0;3.1,Finish to Start,0	60d
	5.2	Final Assembly	08/03/18 09/28/18	08/03/18 09/28/18	5.1,Finish to Start,0	40d

The result shows the predecessor activity ID followed by the relationship type; then any lag value. This is pretty easy to read unless there are more than one predecessor. As you can see with activity 5.1, there are 3 predecessors strung together separated by a semicolon. We can use the STRTRAN function to make this easier to read.

STRTRAN()

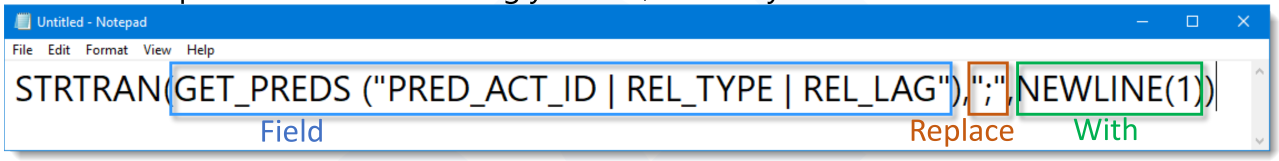
The STRTRAN function replaces one substring with another substring and returns the revised string. Think of it as a "substitute" -- one string for another.

Syntax: STRTRAN(<string1>, <string2>, <string3>)
Data Type: Character
Example: **STRTRAN(**DESCRIPTION**,**","**,**";"**)**

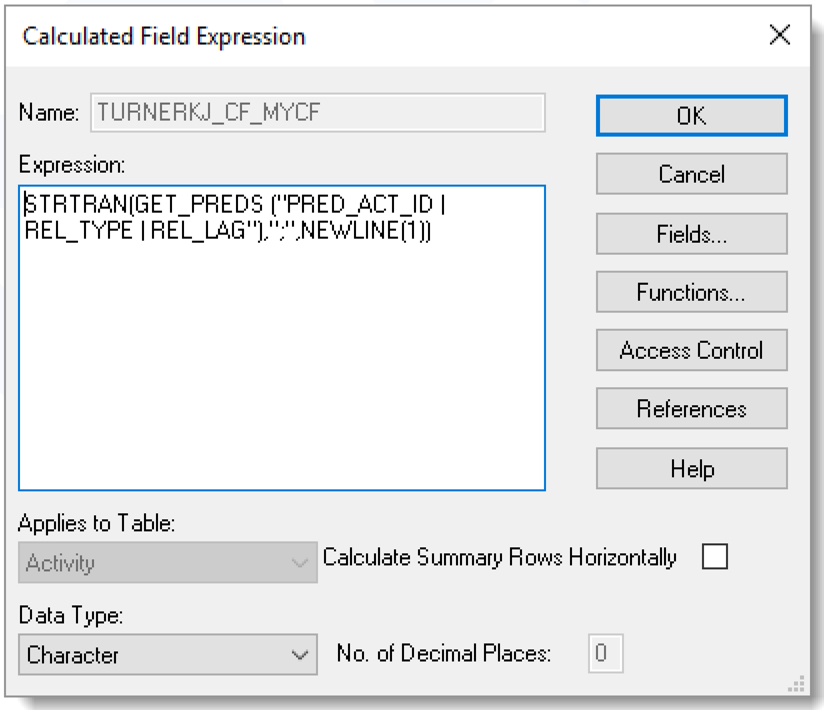
Where	<string1> =	the field name or specific string value you're looking for
	<string2> =	the substring you want to replace
	<string3> =	the substring you want to replace with or substitute

We can use the STRTRAN function to replace the semicolon (;) in the result from the GET_PREDS function with the NEWLINE function. This will cause the field to wrap to a new line for each relationship making it much easier to read.

While the expression looks a little ugly at first, it's really not too bad.



```
STRTRAN(GET_PREDS ("PRED_ACT_ID | REL_TYPE | REL_LAG"),";",NEWLINE(1))
```



Name: TURNERKJ_CF_MYCF

Expression:

```
STRTRAN(GET_PREDS ("PRED_ACT_ID |
REL_TYPE | REL_LAG"),";",NEWLINE(1))
```

Applies to Table: Activity

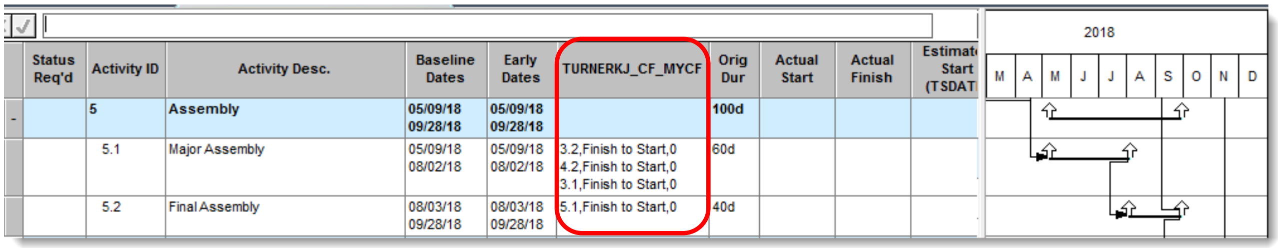
Calculate Summary Rows Horizontally ☐

Data Type: Character

No. of Decimal Places: 0

STRTRAN Function

Press OK to accept the calculated field expression.



	Status Req'd	Activity ID	Activity Desc.	Baseline Dates	Early Dates	TURNERKJ_CF_MYCF	Orig Dur	Actual Start	Actual Finish	Estimate Start (TSDAT
-		5	**Assembly**	**05/09/18 09/28/18**	**05/09/18 09/28/18**		**100d**			
		5.1	Major Assembly	05/09/18 08/02/18	05/09/18 08/02/18	3.2,Finish to Start,0 4.2,Finish to Start,0 3.1,Finish to Start,0	60d			
		5.2	Final Assembly	08/03/18 09/28/18	08/03/18 09/28/18	5.1,Finish to Start,0	40d			

Parsing text strings

Extracting data from text strings can be very useful for reporting. The next several examples show how to pull pieces of text from another string. This can be done with any character field, but let's look at a typical code value in the C9 Integrated Scheduler code field. Given a C9 code file assignment of "**TURNER_JEFFRY>307039**", let's create a calculated field that redisplays the value as just "JEFFRY TURNER" (firstname lastname).

First, we'll create a new calculated field on the Activity table as a Character data type field. We'll begin by showing different ways to pull pieces of the text string; then combine a couple at the end to show the final result.

1. The expression C9.CDR_ID
   Returns: TURNER_JEFFRY>307039

2. The expression **INSTR(**1,C9.CDR_ID,"_"**)**
   Returns: "7"; the position of "_" in the string

3. The expression **LEFT(**C9.CDR_ID**,INSTR(**1,C9.CDR_ID,"_"**)**-1**)**
   Returns: "TURNER"

4. The expression **LEFT(**C9.CDR_ID**,INSTR(**1,C9.CDR_ID,"_"**)**-1**)** + ", "
   Returns: "TURNER, " last name followed by a comma space

5. The expression **MID(**C9.CDR_ID**,INSTR (**1,C9.CDR_ID,"_"**)**+1**)**
   Returns: "JEFFRY>307039" the string after the "_"

6. The expression **INSTR (**1,C9.CDR_ID,">"**)**
   Returns: "14"; the position of ">" in the string

7. The expression **MID(**C9**,INSTR (**1,C9.CDR_ID,">"**)**+1**)**
   Returns: 307039

8. The expression **(INSTR(**1,C9.CDR_ID,">"**)**-1**)-INSTR(**1,C9.CDR_ID,"_"**)**
   Returns: "6"; number of characters in first name

9. The expression **MID(**C9.CDR_ID,**INSTR(**1,C9.CDR_ID,"_"**)**+1**,(INSTR(**1,C9.CDR_ID,">"**)**-1**)-INSTR(**1,C9.CDR_ID,"_"**))**
   Returns: "JEFFRY"; the first name

10. The expression **LEFT(**C9.CDR_ID**,INSTR(**1,C9.CDR_ID,"_"**)**-1**)** + ", " + **MID(**C9.CDR_ID**,INSTR(**1,C9.CDR_ID,"_"**)**+1**,(INSTR(**1,C9.CDR_ID,">"**)**-1**)**-**INSTR(**1,C9.CDR_ID,"_"**))**
    Returns: "TURNER, JEFFRY"; last name, first name

    The final expression is the result of combining **#4** with **#9** using a "+" to concatenate the two pieces together.

11. The expression **MID(**C9.CDR_ID**,INSTR(**1,C9.CDR_ID,"_"**)**+1,**(INSTR(**1,C9.CDR_ID,">"**)**-1**)-INSTR(**1,C9.CDR_ID,"_"**))** **+ " " +** **LEFT(**C9.CDR_ID**,INSTR(**1,C9.CDR_ID,"_"**)**-1**)**
    Returns: “JEFFRY TURNER”

    This expression combines **# 9** and **#3** using "+" to reverse the positions and concatenate the two pieces together.

    Expression #9 from above:
    MID(C9.CDR_ID**,INSTR(**1,C9.CDR_ID,"_"**)**+1**,(INSTR(**1,C9.CDR_ID,">"**)**-1**)**-**INSTR(**1,C9.CDR_ID,"_"**))**
    Returns: "JEFFRY"

    Expression #3 from above:
    LEFT(C9.CDR_ID**,INSTR(**1,C9.CDR_ID,"_"**)**-1**)**
    Returns: "TURNER"

User-Defined Variables

When Open Plan parses a calculated field (CF) expression, it optimizes the expression internally by looking for *multiple occurrences of a subexpression* and replacing these occurrences with an internal variable.

For example, let's say you have defined another CF called TimeNowPlusFive: Timenow() + |5d|
You then use this CF in another CF: IIF(EFDATE > TimeNowPlusFive, TimeNowPlusFive, EFDATE)

Internally, Open Plan will evaluate the TimeNowPlusFive CF only once for a particular cell, and use the result of this evaluation each time the same subexpression is encountered within the CF. In order for Open Plan to perform this optimization, each occurrence of the subexpression must be identical in terms of capitalization and spacing (for example, "EFDATE > ASDATE" is not the same as "efdate > asdate"). This optimization is performed automatically and assures efficient evaluation provided that the user has been consistent as explained above in the use of subexpressions that occur multiple times.

Open Plan also provides a second, more pro-active way for the user to write a CF expression by using variables. Three advantages of using this second option are that
1) the expression becomes much more readable and easier to maintain, since redefining the variable needs to be done in only one place, and
2) the parsing of a very complicated expression that contains variables takes less processing time than the same expression using repeated subexpressions.
3) all expressions necessary to define the result are contained in one place rather than having to go look up the definitions for other fields that are referred to.

Here are the rules for using variables within a CF:

- Variables are defined above the main CF expression and are contained within a BEGIN VARIABLES/END VARIABLES block.
- No blank rows are allowed between the headings BEGIN VARIABLES and END VARIABLES.
- Each variable definition must be on its own line. No carriage returns to wrap the lines within a single variable.
- The result type of a variable definition (for example, character, duration, integer, decimal, date, finish date, and logical) is determined automatically by Open Plan.
- Variable names should not contain spaces or be identical to field names, function names, or other items used within Open Plan where ambiguity may occur. For example, a variable named "123" could be confused with a numeric value. "act_id" would be confused with the Open Plan field of the same name.

- Variables are not case sensitive.
- The format for a variable definition is: <variable name> = <variable expression>.
- The definition of a variable expression may reference other variables, but referenced variables must have been previously defined. So the order that the variables are listed in the variables section determine the order they can be referenced in the expression. For example, the first variable cannot reference another variable that hasn't been defined yet. In the first example below, A cannot reference B because B is defined after A. (You cannot reference a variable before it's been defined.)

  Incorrect
  ```
  BEGIN VARIABLES
  A=B+1
  B=1
  END VARIABLES
  ```

  Correct
  ```
  BEGIN VARIABLES
  A=1
  B=A+1
  END VARIABLES
  ```

Example calculated field expression using variables. The structure starts with a "BEGIN VARIABLES" command followed by the variables themselves; then an "END VARIABLES" command followed by the calculated field expression. The key components have been color coded below to make them easier to see.

```
BEGIN VARIABLES
TNplus30d = DATEFORMAT(DATEADD(TIME_NOW,|30d|,CLH_ID), '%M/%D/%Y')
TNplus60d = DATEFORMAT(DATEADD(TIME_NOW,|60d|,CLH_ID), '%M/%D/%Y')
TNplus90d = DATEFORMAT(DATEADD(TIME_NOW,|90d|,CLH_ID), '%M/%D/%Y')
NotSumOrExtSub=IIF(ACT_TYPE<>[Subproject] and ACT_TYPE<>[External
Subproject],"True","False")
END VARIABLES

IIF(COMPSTAT=[COMPLETE], "Complete",
IIF(BSDATE={},"No Baseline",
IIF(NotSumOrExtSub="True" and (BSDATE<>{} and TIMENOW(x)>= BSDATE and
ASDATE={}) or (BFDATE<>{}and TIMENOW(x)>=BFDATE and AFDATE={}),"Late",
IIF(NotSumOrExtSub="True" and (TNplus30d>=BSDATE or TNplus30d>= BFDATE),"30
Days",
```

```
IIF(NotSumOrExtSub="True" and (TNplus60d>=BSDATE or TNplus60d>=BFDATE),"60
Days",
IIF(NotSumOrExtSub="True" and (TNplus90d>=BSDATE or TNplus90d>=BFDATE),"90
Days",
IIF(NotSumOrExtSub="True" and (TNplus90d>=BSDATE or TNplus90d>=BFDATE)," ","
")))))))
```

When one or more components in an expression are used multiple times, it's quicker/easier to update the components as variables in one spot than to have to update each occurrence in the expression itself.

Due to a known bug/defect, the use of the *less than* (<) symbol by itself for comparison does not parse properly when using Variables. So instead of BFDATE<TIMENOW(x), turn it around and phrase it as TIMENOW(x)>=BFDATE.

The same expression written without variables would look like this:

```
IIF(COMPSTAT=[COMPLETE], "Complete",
IIF(BSDATE={},"No Baseline", IIF(ACT_TYPE<>[Subproject] and ACT_TYPE<>[External
Subproject] and (BSDATE<>{} and BSDATE<TIMENOW(x) and ASDATE={}) or
(BFDATE<>{}and BFDATE<TIMENOW(x) and AFDATE={}),"Late",
IIF(ACT_TYPE<>[Subproject] and ACT_TYPE<>[External Subproject] and
(BSDATE<DATEFORMAT(DATEADD(TIME_NOW,|30d|,CLH_ID), '%M/%D/%Y') or
BFDATE<DATEFORMAT(DATEADD(TIME_NOW,|30d|,CLH_ID), '%M/%D/%Y')),"30 Days",
IIF(ACT_TYPE<>[Subproject] and ACT_TYPE<>[External Subproject] and
(BSDATE<DATEFORMAT(DATEADD(TIME_NOW,|60d|,CLH_ID), '%M/%D/%Y') or
BFDATE<DATEFORMAT(DATEADD(TIME_NOW,|60d|,CLH_ID), '%M/%D/%Y')),"60 Days",
IIF(ACT_TYPE<>[Subproject] and ACT_TYPE<>[External Subproject] and
(BSDATE<DATEFORMAT(DATEADD(TIME_NOW,|90d|,CLH_ID), '%M/%D/%Y') or
BFDATE<DATEFORMAT(DATEADD(TIME_NOW,|90d|,CLH_ID), '%M/%D/%Y')),"90 Days",
IIF(ACT_TYPE<>[Subproject] and ACT_TYPE<>[External Subproject] and
(BSDATE>DATEFORMAT(DATEADD(TIME_NOW,|90d|,CLH_ID), '%M/%D/%Y') or
BFDATE>DATEFORMAT(DATEADD(TIME_NOW,|90d|,CLH_ID), '%M/%D/%Y'))," "," ")))))))
```

In complex expressions, the use of variables can make understanding what the calculated field is doing much easier to follow logically.

Tip: Variables can be used to add descriptive commentary to explain the purpose or add other explanatory information about what the expression is intended to do. For example, you can create a variable named “Purpose” and set it equal to a text string that explains something about the expression.

```
BEGIN VARIABLES
Purpose=”This calculated fields will…etc.”
END VARIABLES
```

CHAPTER 3 – OPP FUNCTIONS

ABS(<value>**)**
Data Type: Decimal or integer
The function ABS() finds the absolute value. The expression: ABS(|-2|)
Returns: 2

BASELINE_FIELD(<SelectedBaselineIndex>, <BaselineFieldName>**)**
Data Type:
The function BASELINE_FIELD() allows you to get the value in a field for a specific selected baseline

CDOW(<date>**)**
Data Type: Character
The function CDOW returns the Day Of the Week for a date field. For example, assuming that the ESDATE is 10/07/16 (Friday), the statement: CDOW(ESDATE)
Returns: Friday.

CMONTH(<date>**)**
Data Type: Character
The function CMONTH returns the Month for a specific date field.
Assuming ESDATE is 10/07/16, the statement: CMONTH(ESDATE)
Returns: October.

CTOD(<String Expression>**)**
Data Type: Date
The function CTOD (Convert TO Date) takes a string value and converts it to a date
For example, if USER_NUM01 = 12, USER_NUM02 = 31 and Time Now = 1/1/2006
the following expression: CTOD(STR(USER_NUM01) + "/" + STR(USER_NUM02) + "/" + STR(YEAR(TIMENOW())))
Returns: 12/31/2006.

DATE( )
Data Type: Date
The function DATE( ) returns the current date. Assuming the current date is October 7, 2016, the statement: DATE( )
Returns: 10/07/16. (using the default date format in Project, Preferences)

DATEADD(<start date>**,** <Dur>**,** <calendar >**)**
Data Type: Date
The function DATEADD allows you to add a given duration to a date based on a stated calendar to get a new date. For example, assuming Time Now = 10/05/16
6 DAY - BOEING HOLIDAYS = the name of the calendar you want to use;
the expression: DATEADD (TIMENOW(),|2d|, "6 DAY - BOEING HOLIDAYS")
returns {10/07/16}

Note the quotation marks around the exact calendar name in the expression and the piping symbols around the duration value.

To subtract a date, multiply the duration value times -1. For example
DATEADD (TIMENOW(),|2d|*-1, "6 DAY - BOEING HOLIDAYS")

DATEDIFFERENCE(<date1>**,** <date2>**,** <calendar>**)**
Data Type: Duration
The function DATEDIFFERENCE allows you to calculate the duration difference between two dates.

Assuming that ESDATE is 10/05/16 and EFDATE is 10/07/16, the statement:
DATEDIFFERENCE (ESDATE, EFDATE, "6 DAY - BOEING HOLIDAYS")
returns the duration value: 2d

When comparing different date fields (i.e. ASDATE to BSDATE), if the date in the second position of the function is earlier than the date in the first position, the returned value will be a negative duration, like -2d.

Note the quotation marks around the exact calendar name in the expression.   If you just want to use the default calendar being used for the project, substitute the calendar name with CLH_ID, so the expression would look like: DATEDIFFERENCE (ESDATE, EFDATE, CLH_ID)
Note, there are not quotation marks around the CLH_ID because it's not a specific calendar name.
Also, spaces are not required to separate the values in the expression but it makes it easier to read.

DATEFORMAT(<date field>**,** <format string>**)**
Data Type: Character
The function DATEFORMAT allows you to format a date to a specific date format.

Assuming that ESDATE is October 7, 2016, the statement: DATEFORMAT (ESDATE, "%M%D%Y")
returns: 10/07/16

Note the quotation marks around the formatting part of the statement.
Also, spaces are not required to separate the values in the expression but it makes it easier to read.

%T – Minutes	15
%H – Hour	8
%Z – AM or PM	A
%D – Day of the month	31
%V – Day	Tuesday
%W – Day Abbreviation	Tues
%F – Single Character Day	T
%K – Week Number	52
%M – Numeric Month	11
%L – Month	November
%A – Month Abbreviation	Nov
%S – Single Character Month	N
%Q – Numeric Quarter	4
%C – Year	2002
%Y – Two-digit Year	02

DAY(<date>**)**
Data Type: Integer
The function DAY allows you to get the numerical day of the month from a date field.
For example, assuming: ESDATE = 10/04/16, the expression: DAY(ESDATE)
returns: 4

DOW(<date>**)**
Data Type: Integer
The function DOW allows you to get the numerical day of the week from a date field.
Assuming that ESDATE is 10/07/16 (Friday), the statement: DOW(ESDATE)
returns 6.

DURATION(<Dur Value>**)**
Data Type: Integer

The function DURATION calculates the equivalent minutes for any duration expressed in any unit of time such as |2w| (t=minutes, h=hours, d=days, w=weeks, m=months) Assuming an 8-hour workday, the expression: DURATION(|2d|)
returns: 960 (8 hours x 2 days x 60 minutes/hour).

DURATION(|2w|)
returns: 4800 (8 hrs x 10 days x 60 minutes/hour)
Note the piping symbols around the duration value in the expression.

EVAL(<exp>**)**
Data Type:
The function EVAL forces a string value that could be interpreted as specific type of data as that data type.

For example, assuming that: ESDATE=10/07/16 and USER_CHR01 contains "ESDATE"; then the expression: EVAL(USER_CHR01)
returns: 10/07/16
Your data type for this Calculated Field should be set to "Date" type.

Another example, assuming that the value of:
USER_CHR01 is "ESDATE" and USER_CHR02 is "EFDATE" and that ESDATE is 10/05/16 and EFDATE is 10/07/16, the expression: EVAL(USER_CHR02+"-"+USER_CHR01) returns a duration of: 2d.
Your data type for this Calculated Field should be set to "Duration" type.

FAIL_EVALUATE(<String Expression>**)**
Data Type: Boolean
The function FAIL_EVALUATE tests for a condition that normally would cause the calculated field to return either a blank value or an invalid result (the latter would keep the remainder of the expression from being evaluated).

FAIL_EVALUATE("C4")
If there is no code file assigned at index 4, (in this example, Control Accounts) this calculated field will return True which means it "failed to evaluate". Note that if we substituted the expression "C4 IS_EMPTY" as an alternative to using FAIL_EVALUATE, the returned value would be True whether C4 does not exist or C4 exists but is blank.

Your data type for this Calculated Field should be set to "Logical" or "Character" type.

FISCALPERIOD(<Date>**,**<Reporting Calendar>**)**

Data Type: Character
The function returns the LABEL field from the reporting calendar that includes the supplied date. For example, given the following DATE and LABEL fields:

DATE	LABEL
01/26/2001	Jan 2001
02/23/2001	Feb 2001
03/23/2001	Mar 2001

FORMAT_HEADING_ITEM(<Expression>**,**<WidthInCharUnits>**,** <Wordwrap>**,**<Summarize>**)**
Data Type: Character
Given the following calculated field expression:
FORMAT_HEADING_ITEM(C2 + " - ," 20,0,0) + FORMAT_HEADING_ITEM(C2.DESCRIPTION, 40, 1,0) + FORMAT_HEADING_ITEM(C2.< Default >,20,1,0)

GET_ASSGNS(<fieldname1>[|<fieldname2>...]**)**
Data Type: Character
Assuming that the resource assignments for the activity are ENG (a level 1 resource pool) and TECH.MARY (a level 2 resource), the statement:
GET_ASSGNS("RES_ID|RES_LEVEL")
returns ENG,1.00;TECH.MARY,2.00.

GET_CHILDREN(<FieldList>**)**
Data Type: Character
GET_CHILDREN("ACT_ID|DESCRIPTION")
Given an activity "1.01,"with children, "1.01.01" and "1.01.02,"the returned string would be the following:
"1.01.01,First child of 1.01;1.01.02,Second child of 1.01"

GET_COSTS(<fieldname1>[| <fieldname2>...]**)**
Data Type: Character
Assuming that there are two cost records for the activity, that the actual costs are 1,000 and 4,000, and that the actual quantities are 1 and 2, the statement:
GET_COSTS("ACWP_CST|ACWP_QTY")
It then returns 1000.00,1.00;4000.00,2.00.

GET_FIELD(<TableType>**,** <UniqueID>**,**<FieldName>**)**
Data Type: Character

GET_FIELD("C2",PARENT(C2),"DESCRIPTION")
On the Activity table, if the current Activity has a code 2 value of "1.2.1.3," the returned string would be "1.2.1. System Engineering."

GET_FIRST_RECORD_IN_SUMMARY(<TableType>**,** <UniqueID>**,**<FieldName>**)**
Data Type:
The purpose of the GET_FIRST_RECORD_IN_SUMMARY function is to allow the user to construct a calculated field result in a summary row that uses a break column value rather than a summary value. For example, given the following filter in a barchart (with a visibility setting of "All levels of Rollup and Detail"), with a spreadsheet pane subsectioned on TOTALFLOAT:
TOTALFLOAT > |2d|
The filter would evaluate to TRUE for all child rows of subsections where the TOTALFLOAT break-on value is greater than |2d|. However, the filter would evaluate to FALSE on the subsection summary rows since the value of the TOTALFLOAT field on the summary row is undefined (durations cannot be summarized). Even for a numeric or date field (which can be summarized), the filter would have to be written differently to account for a summarized value on a summary row. To work around this problem, the following syntax will give us the result we want:
GET_FIRST_RECORD_IN_SUMMARY("TOTALFLOAT") > |2d|

GET_NOTE(<category>**)**
Data Type: Character
Assuming that a category <Document> has been set up for an activity table, the statement:
GET_NOTE("Document")
returns the Document category note about the activity.

GET_PREDS(<fieldname1>[|<fieldname2>...]**)**
Data Type: Character
Assuming that the predecessor to an activity is A100 and that the relationship type is Finish to Start, the statement: GET_PREDS("PRED_ACT_ID|REL_TYPE")
It then returns A100,Finish to Start.

GET_RELATED(<CollectionType>, <FieldList>**)**
Data Type: Character
Assuming that we are displaying an activity spreadsheet, and that activity "A" is the current activity, the expression below will return the activity ID and early start fields for the record that matches the activity on the first selected baseline.
GET_RELATED("A01," "ACT_ID|ESDATE") returns "A,12Oct04"

GET_RELATED_Count(<CollectionType>, <FieldList>**)**
Data Type: Character
This function returns a record count. The two parameters for the function are the related collection identifier and an optional filter. You use this primarily with activity and baseline activity tables.

GET_RISKS(<fieldname1>[| <fieldname2>...]**)**
Data Type: Character
The statement: GET_RISKS("ESDATE1")
returns each early start date calculated during risk analysis.

GET_SUCCS(<fieldname1>[|<fieldname2>...]**)**
Data Type: Character
Assuming that the successor to an activity is A200 and that the relationship type is Finish to Start, the statement: GET_SUCCS("SUCC_ACT_ID|REL_TYPE")
Returns: 2.1,Start to Start

GET_USAGES(<fieldname1>[| < fieldname2>...]**)**
Data Type: Character
Assuming that an activity having a single use record shows that 24 units of the resource ENG have been used, the statement: GET_USAGES("RES_ID|RES_USED")
Returns: ENG,24.

GO_MONTH(<date>**,**<Intg>**)**
Data Type: Date
Assuming that ESDATE is 04OCT04, the statement: GO_MONTH(ESDATE,-2)
Returns: 04AUG04

HAS_NOTE(<string>**)**
Data Type: Logical
Assuming that an activity has a note in the category "Scope", the statement:
HAS_NOTE("Scope")
Returns: True

IIF(<logicexp>**,**<iftrue>**,**<iffalse>**)**
Data Type: Any
Assuming that ESDATE is 19JUN01, the statement:
IIF(ESDATE>{01JUL01}, "Underway", "Planned")
Returns: Planned.

INLIST(<search>**,**<value1>**,** <value2>**,** ...**)**
Data Type: Logical
The function INLIST checks for the existence of a value in a list of values.
Assuming that the early start date month is June, which is the 6th month, the statement:
INLIST(MONTH(ESDATE),1,4,7,10)
Returns: False.

The MONTH function returns the numeric month value of the ESDATE.
Basically, it's looking for "6" in the list of "1, 4, 7, 10" and does not find it so it returns "False"

INSTR(<start>**,**<string1>**,**<string2>**)**
Data Type: Integer
The function INSTR checks for the presence of a string in a field and returns the character position where the string starts.

For example, assuming DESCRIPTION="SITE COORDINATION AND DESIGN"
And you want to know if the string "COORDINATION" appears in the string. The following expression: INSTR(1,DESCRIPTION, "COORDINATION")
Returns: 6.

Beginning in the 1st character position of the DESCRIPTION field, the INSTR function searches for the string "COORDINATION" and finds the string beginning with character position 6.

Note, the string you are searching for must be enclosed in quotation marks (double or single).

LEFT(<string>**,**<int>**)**
Data Type: Character
The function LEFT enables you to extract a string beginning from the left side of text string. For example, beginning from the Left, extract the first 4 characters from the string, the expression: LEFT("SITE COORDINATION AND DESIGN", 4)
Returns: "SITE"

LEN(<data>**)**
Data Type: Integer
The function LEN counts the number of characters in a string.
For example, if you want to know how many characters are in the activity description, Assuming the description is "Preliminary Design Review (PDR)", the expression:

LEN(DESCRIPTION)
Returns: 31

LEVEL(<ID>**)**
Data Type: Integer
The function LEVEL() returns the numeric value of the structural level of the value being tested. Assuming that a code file is assigned to the project in position C1, and the value assigned to an activity is 1.1.1, then the statement: LEVEL(C1)
Returns: "3" -- the level of 1.1.1 in the code structure.

LOCAL(<ID>**,** <level>**)**
Data Type: Character
Assuming that a code file is assigned to the project in position C1, the statement:
LOCAL(C1)
Returns the local (rightmost) portion of C1.

LOWER(<string>**)**
Data Type: Character
The function LOWER forces a text string to all lower case.
For example, assuming USER_CHR01 contains "Dig Hole", the expression:
LOWER(USER_CHR01)
Returns: "dig hole"

LTRIM(<string>**)**
Data Type: Character
The function LTRIM checks for the presence of spaces beginning on the left and removes them. For example, assuming DESCRIPTION="   Administration" (has leading spaces), the expression: LTRIM(DESCRIPTION)
Returns: "Administration"

MAX(<value1>**,** [<value 2>**,** ...]**)**
Data Type: Decimal, Integer
The function MAX() returns the maximum value found in a list. If the values are numeric, the maximum is the largest number; if the values are dates, the maximum is the latest date. Assuming that value1 is 4, value2 is 25, and value3 is 66, the statement:
MAX(4, 25, 66)
Returns: 66.

MID(<string>**,**<start>**,**<int>**)**
Data Type: Character

Extract a value/text string from within another string
The statement: MID("SITE COORDINATION AND DESIGN",6, 12) returns COORDINATION

This function does essentially the same thing as the SUBSTR function. The difference is in how the function will index the characters. The MID function indices the characters in the text string "TEXT" as 1,2,3,4 whereas the SUBSTR function will index them as 0,1,2,3.

MIN(<value1>**,**<value 2>**, ...)**
Data Type: Integer, decimal
Assuming that value1 is 4, value2 is 25, and value3 is 66, the statement: MIN(4, 25, 66)
Returns: 4.

MONTH(<date>**)**
Data Type: Integer
The function MONTH() returns the numeric value of the month in a date field.
Assuming ESDATE is 04JUL04, the statement: MONTH(ESDATE)
Returns: 7.

NEWLINE(<value>**)**
Data Type: Integer
The function NEWLINE(1) wraps string values to a new line

The expression:
```
    "Early: "+DATEFORMAT(ESDATE, "%M%D%Y")+NEWLINE(1)+
    "Late: "+DATEFORMAT(LSDATE, "%M%D%Y")+NEWLINE(1)+
    "Sched: "+ DATEFORMAT(SSDATE, "%M%D%Y")
```
returns a string similar to the following:
```
    Early: 04/12/16
    Late: 04/16/16
    Sched: 04/14/16
```

Notice that the NEWLINE() function has forced the Late and Scheduled dates to be moved to the next line. Including a number between the parentheses indicates how many blank lines to wrap. NEWLINE(2) will double-space.
The plus (+) sign is used to concatenate the function into the string.

Also, when this field is added to a view, the setting to Wrap Text in Cells must be toggled ON for the column.

NUMBER_FORMAT(<Intg>**,** <string>**,** <Intg>**)**
Data Type: Integer
The expression: NUMBER_FORMAT(1000, "$.", 2)

Returns $1,000.00

OCCURS(<string1>**,**<string2>**)**
Data Type: Integer
The statement: OCCURS("Bright light circuits", "ight")
Returns: 2.

PARENT(<ID>**,** <level>**)**
Data Type: Character
Assuming that a code file is assigned to the project in position C1, then the statement:
PARENT(C1)
returns the ID of the immediate parent of C1.

RECORD_NUMBER()
Data Type: Integer
This function is useful in general export scripts. For example:
EXPORT csv Sample Export Script
TABLE ACT
FIELD RECORD_NUMBER()
FIELD ACT_ID

RIGHT(<string>**,**<int>**)**
Data Type: Character
The function RIGHT enables you to extract a string beginning from the right side of text string. For example, beginning from the Right, extract the last 6 characters from the string, the expression: RIGHT("SITE COORDINATION AND DESIGN", 6)
Returns: "DESIGN"

ROUND(<value>**,** <precision>**,** <type>**)**
Data Type: Decimal, Integer
The function ROUND is used to round a value up or down to a specified precision. For example, the expression: ROUND(458.9738, 2, "up")
Returns: 458.98.

The expression: ROUND(orig_dur / 3.0, "d")
returns the original duration divided by 3.0 and rounded up to the nearest day.

The expression: ROUND(4589, -2)
Returns: 4600.

SPACE(<int>**)**
Data Type: Character
The statement: SPACE(6)
Returns six space characters.

SQRT(<value>**)**
Data Type: Decimal, Integer
The statement: SQRT(4)
Returns: 2.

STR(<value>,<length>,<Dec>)
Data Type: Character
The statement: STR(1.2345,3,1)
Returns: 1.2

Assuming that ESDATE is October 5, 2004, the statement: STR(ESDATE)
Returns 05OCT04

STRTRAN(<string1>**,**<string2>**,**<string3>**)**
Data Type: Character
If the Activity Description is "PDR Finished", the expression:
STRTRAN(DESCRIPTION,"Finished","Complete")
Returns: PDR Complete

STUFF(<string1>**,**<start>**,**<int>**,**<string2>**)**
Data Type: Character

SUBSTR(<string>**,** <start>**,** <length>**)**
Data Type: Character
SUBSTR("ABCDEF", 3, 2)
Returns: CD

TIMENOW( )

Data Type: Date
The function (TIMENOW) returns the current Timenow (Status date) in Project, Properties, Status tab. Assuming the currently set Time Now date is October 7, 2004, the statement:
TIMENOW( )
Returns: 10/07/14.
The date format used will be the default format set under Project Properties, Preferences. You can use the DATEFORMAT function to force the format to the desired date format.

TRIM(<string>)
Data Type: Character
The function TRIM checks for the presence of spaces at the end of a string and removes them. For example, assuming that DESCRIPTION = "Administration   "   (field has trailing spaces), the expression:
TRIM(DESCRIPTION)
Returns: "Administration"

UPPER(<string>**)**
Data Type: Character
The function UPPER forces a text string to all upper case.
For example, assuming USER_CHR01 contains "Dig Hole", the expression:
UPPER(USER_CHR01)
Returns: "DIG HOLE"

USER_ID(<string>**)**
Data Type: Character
The expression USER_ID()
Returns: "SYSADMIN" (or whomever the User ID is)

VAL(<string>**)**
Data Type: Decimal, Integer
The function VAL is used to convert a string to a value. For example, the expression:
VAL("1678")
Returns: 1678

YEAR(<date>**)**
Data Type: Integer

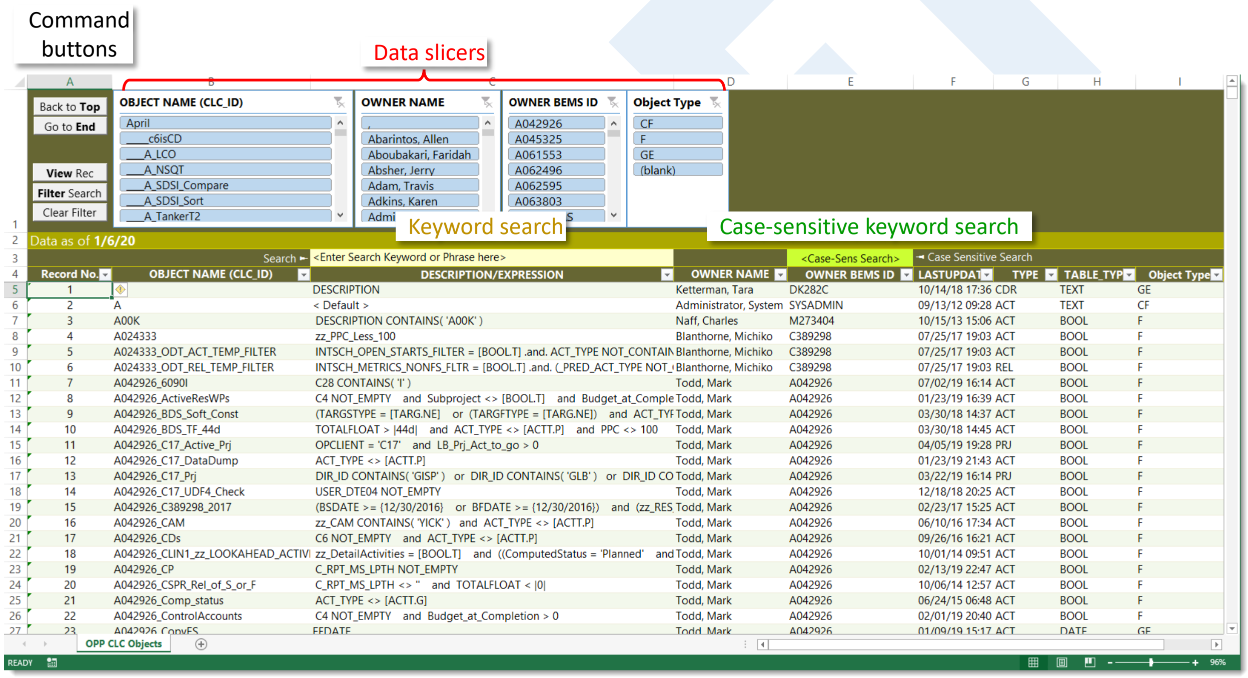
The function YEAR returns the numerical year of the date field identified. For example, assuming ESDATE is 10/07/20, the expression:
YEAR(ESDATE)
Returns: 2020.

CHAPTER 4 – CLC TOOLS

OPP CLC OBJECT BROWSER

The CLC Object Browser is an Excel tool allowing you to search CLC objects and get expressions even if you don't have access to the object. The tool enables you to do key word search that are case or non-case sensitive. Using the data slicers at the top, you can easily filter to the 3 major types of CLC objects: Calculated Fields, Filters and Global Edits.

Available for download from http://ips.web.boeing.com/index.htm; click the Tools button and look in the OPP section.

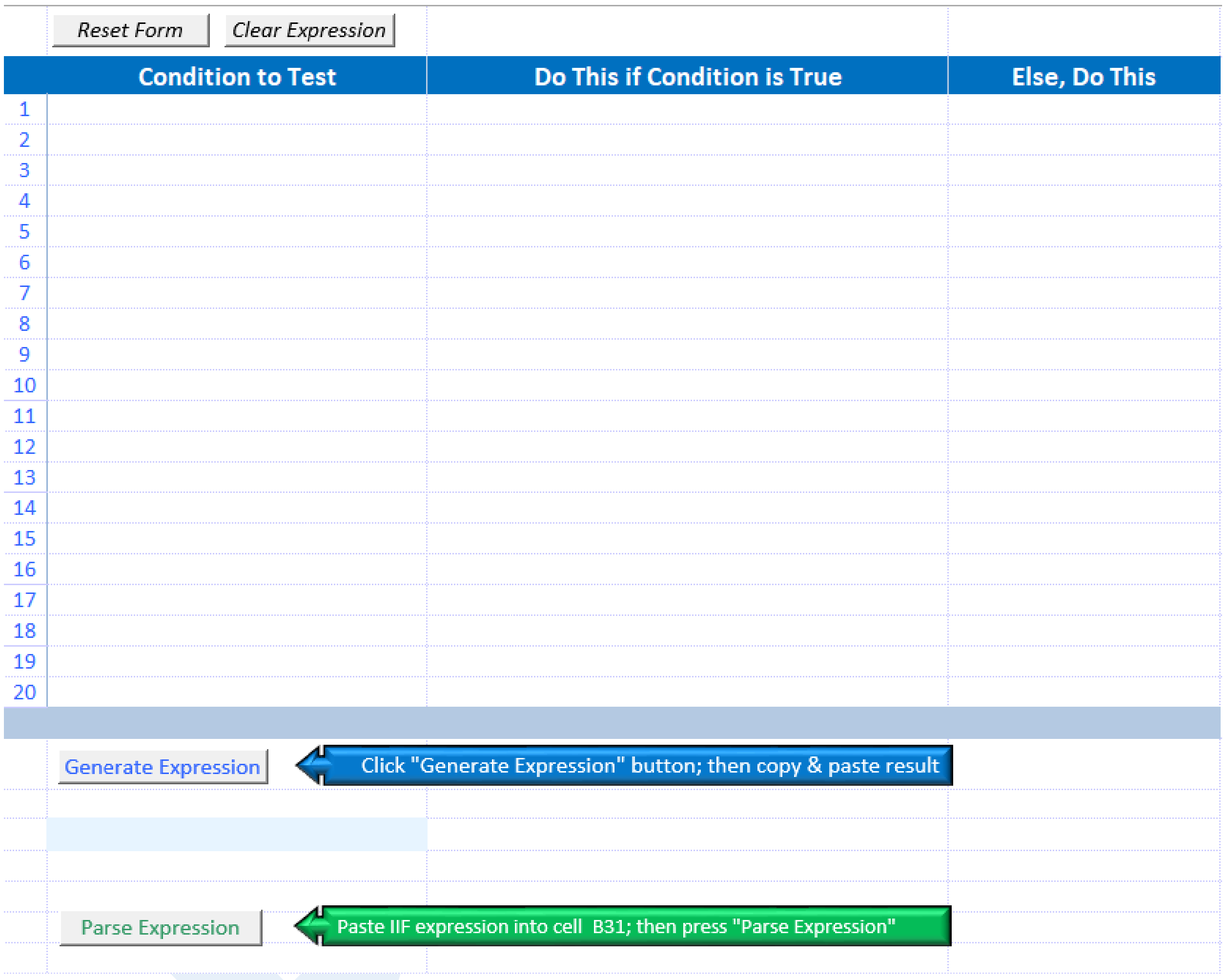


Record No.	OBJECT NAME (CLC_ID)	DESCRIPTION/EXPRESSION	OWNER NAME	OWNER BEMS ID	LASTUPDAT	TYPE	TABLE_TYP	Object Type
1		DESCRIPTION	Ketterman, Tara	DK282C	10/14/18 17:36	CDR	TEXT	GE
2	A	< Default >	Administrator, System	SYSADMIN	09/13/12 09:28	ACT	TEXT	CF
3	A00K	DESCRIPTION CONTAINS('A00K')	Naff, Charles	M273404	10/15/13 15:06	ACT	BOOL	F
4	A024333	zz_PPC_Less_100	Blanthorne, Michiko	C389298	07/25/17 19:03	ACT	BOOL	F
5	A024333_ODT_ACT_TEMP_FILTER	INTSCH_OPEN_STARTS_FILTER = [BOOL.T] .and. ACT_TYPE NOT_CONTAIN	Blanthorne, Michiko	C389298	07/25/17 19:03	ACT	BOOL	F
6	A024333_ODT_REL_TEMP_FILTER	INTSCH_METRICS_NONFS_FLTR = [BOOL.T] .and. (_PRED_ACT_TYPE NOT_	Blanthorne, Michiko	C389298	07/25/17 19:03	REL	BOOL	F
7	A042926_6090I	C28 CONTAINS('I')	Todd, Mark	A042926	07/02/19 16:14	ACT	BOOL	F
8	A042926_ActiveResWPs	C4 NOT_EMPTY and Subproject <> [BOOL.T] and Budget_at_Comple	Todd, Mark	A042926	01/23/19 16:39	ACT	BOOL	F
9	A042926_BDS_Soft_Const	(TARGSTYPE = [TARG.NE] or (TARGFTYPE = [TARG.NE]) and ACT_TYF	Todd, Mark	A042926	03/30/18 14:37	ACT	BOOL	F
10	A042926_BDS_TF_44d	TOTALFLOAT > |44d| and ACT_TYPE <> [ACTT.P] and PPC <> 100	Todd, Mark	A042926	03/30/18 14:45	ACT	BOOL	F
11	A042926_C17_Active_Prj	OPCLIENT = 'C17' and LB_Prj_Act_to_go > 0	Todd, Mark	A042926	04/05/19 19:28	PRJ	BOOL	F
12	A042926_C17_DataDump	ACT_TYPE <> [ACTT.P]	Todd, Mark	A042926	01/23/19 21:43	ACT	BOOL	F
13	A042926_C17_Prj	DIR_ID CONTAINS('GISP') or DIR_ID CONTAINS('GLB') or DIR_ID CO	Todd, Mark	A042926	03/22/19 16:14	PRJ	BOOL	F
14	A042926_C17_UDF4_Check	USER_DTE04 NOT_EMPTY	Todd, Mark	A042926	12/18/18 20:25	ACT	BOOL	F
15	A042926_C389298_2017	(BSDATE >= {12/30/2016} or BFDATE >= {12/30/2016}) and (zz_RES	Todd, Mark	A042926	02/23/17 15:25	ACT	BOOL	F
16	A042926_CAM	zz_CAM CONTAINS('YICK') and ACT_TYPE <> [ACTT.P]	Todd, Mark	A042926	06/10/16 17:34	ACT	BOOL	F
17	A042926_CDs	C6 NOT_EMPTY and ACT_TYPE <> [ACTT.P]	Todd, Mark	A042926	09/26/16 16:21	ACT	BOOL	F
18	A042926_CLIN1_zz_LOOKAHEAD_ACTIVI	zz_DetailActivities = [BOOL.T] and ((ComputedStatus = 'Planned' and	Todd, Mark	A042926	10/01/14 09:51	ACT	BOOL	F
19	A042926_CP	C_RPT_MS_LPTH NOT_EMPTY	Todd, Mark	A042926	02/13/19 22:47	ACT	BOOL	F
20	A042926_CSPR_Rel_of_S_or_F	C_RPT_MS_LPTH <> '' and TOTALFLOAT < |0|	Todd, Mark	A042926	10/06/14 12:57	ACT	BOOL	F
21	A042926_Comp_status	ACT_TYPE <> [ACTT.G]	Todd, Mark	A042926	06/24/15 06:48	ACT	BOOL	F
22	A042926_ControlAccounts	C4 NOT_EMPTY and Budget_at_Completion > 0	Todd, Mark	A042926	02/01/19 20:40	ACT	BOOL	F
23	A042926_ConvES	EFDATE	Todd, Mark	A042926	01/09/19 15:17	ACT	DATE	GE

CLC Object Browser

Nested IIF Builder

The Nested IIF Builder is an Excel tool that simplifies the creation of multi-level IIF expressions by allowing the user to structure the arguments in a table format and then press a button to generate the corresponding expression with the required syntax. Available for download from: http://ips.web.boeing.com/Source/Scheduling/Tools/OPP/NestedIIFBuilder.zip



	Condition to Test	Do This if Condition is True	Else, Do This
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			

Click "Generate Expression" button; then copy & paste result

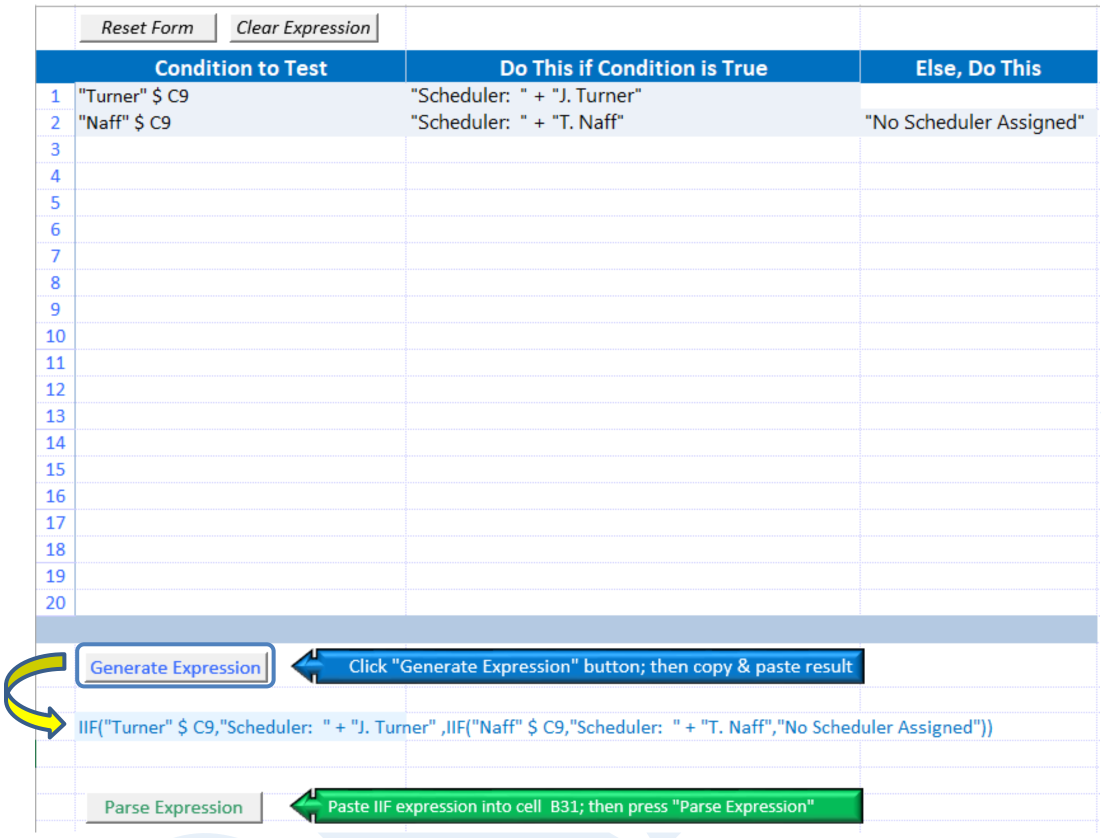
Paste IIF expression into cell B31; then press "Parse Expression"

Nested IIF Builder

In the example below, we want to create a nested IIF statement that will look in the C9 Scheduler code field for either "Turner" or "Naff". If it finds Turner, it will return "J. Turner"; if it finds Naff it will return "T. Naff" and if neither, it will return "No Scheduler Assigned".

You still have to follow the proper syntax for calculated field expressions; but the benefit of the tool is keeping track of the nesting and the placement of commas and parentheses.

Once the table is populated, press the "Generate Expression" button and the tool will create the IIF expression for you.


Reset Form
Clear Expression
Generate Expression
Click "Generate Expression" button; then copy & paste result
Parse Expression
Paste IIF expression into cell B31; then press "Parse Expression"

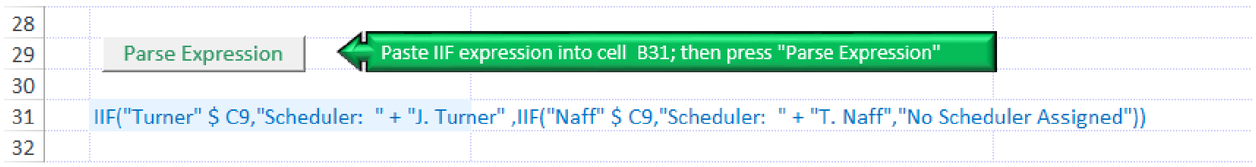
	Condition to Test	Do This if Condition is True	Else, Do This
1	"Turner" $ C9	"Scheduler: " + "J. Turner"	
2	"Naff" $ C9	"Scheduler: " + "T. Naff"	"No Scheduler Assigned"
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			

```
IIF("Turner" $ C9,"Scheduler:  " + "J. Turner" ,IIF("Naff" $ C9,"Scheduler:  " + "T. Naff","No Scheduler Assigned"))
```

```
IIF("Turner" $ C9,"Scheduler:  " + "J. Turner" ,IIF("Naff" $ C9,"Scheduler:  " + "T. Naff","No Scheduler Assigned"))
```

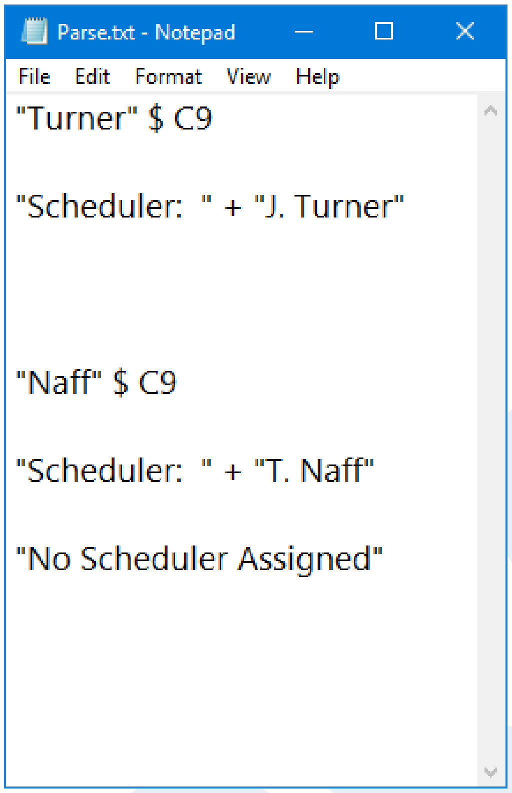
Press the Clear Expression button to clear the cell (B26) of the Expression.
Press the Reset Form button to clear the table of conditions/tests – making ready for a new expression.

If you have a nested IIF statement, you can paste it into cell B31 and press the Parse Expression button. This will parse the IIF statements into a Notepad document showing the breakdown of IIF statements making it easier to read what it's doing. For example, if we copy the same expression generated above and paste it into cell B31 and press the Parse Expression button, the result will look like the following:


Parse Expression
Paste IIF expression into cell B31; then press "Parse Expression"

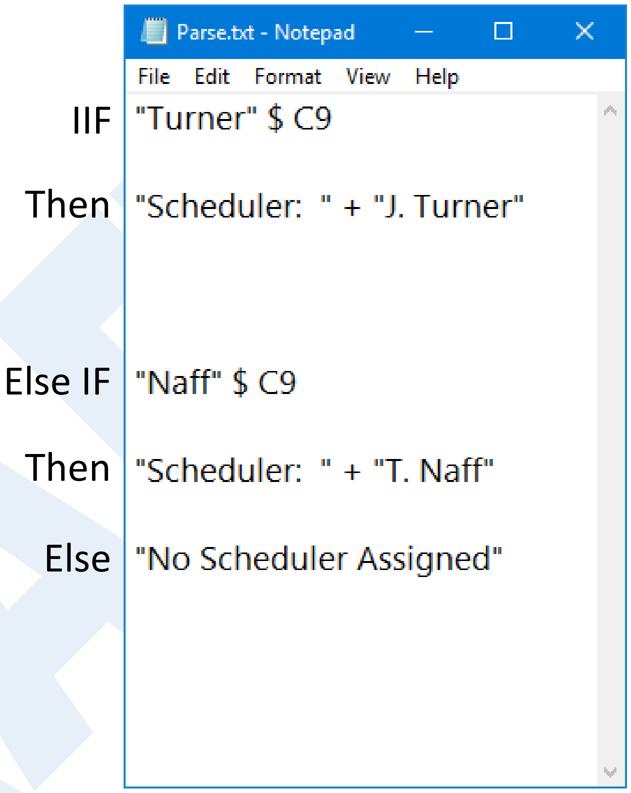
IIF("Turner" $ C9,"Scheduler: " + "J. Turner" ,IIF("Naff" $ C9,"Scheduler: " + "T. Naff","No Scheduler Assigned"))

Parsed Output



```
"Turner" $ C9

"Scheduler:  " + "J. Turner"



"Naff" $ C9

"Scheduler:  " + "T. Naff"

"No Scheduler Assigned"
```

Read this way...



```
IIF      "Turner" $ C9

Then     "Scheduler:  " + "J. Turner"



Else IF  "Naff" $ C9

Then     "Scheduler:  " + "T. Naff"

Else     "No Scheduler Assigned"
```

It breaks down the IIF statement into an easier to read format than one long string such as the following:

IIF("Turner" $ C9,"Scheduler: " + "J. Turner" ,IIF("Naff" $ C9,"Scheduler: " + "T. Naff","No Scheduler Assigned"))

This simple example is not too difficult to read, but more complex statements can be very confusing to read if left as one continuous run-on statement.