Chameleon Information Management Services Limited

INFOFLEX v5

FUNCTIONS AND EXPRESSIONS USER GUIDE

Document control

Document name	User Guide for InfoFlex Functions and Expressions
Confidentiality	Customer
Owner	Helen Vickers
Version	1.6
InfoFlex Version	5.70.0100
Last revised by	JW
Last revised date	Nov 2017
Status	Final

Document history

Date	Doc version	Ifx version	Editor	Change
March 2010	1.0.7	5.30.3700	JE	New document
June 2010	1.0.8	5.40.0100	JW	Update for 5.40.0100 optional 3rd IIF argument 2.1.6 DAYS was merged with 2.1.5 Adjustment to 2.8 Options for Missing, Blank and Unknown values
July 2010	1.0.9	5.40.0100	JW	Further revision of 2.8 Options for Missing, Blank and Unknown values
June 2011	1.1.0	5.40.0400	JE	Revision to DateDiff, Months, Years and Workday Function
June 2011	1.1.1	5.40.0400	JW	Revisions to 2.5.3 (missing word “omit”)
July 2012	1.2	5.50.0100	JW	New Calendar Days and ISINCALENDAR expressions
March 3013	1.3	5.50.0400	JW	Update for 5.40.0400 New functions: NewID and Hashvalue
May 2014	1.4	5.60.0200	JW	Update for 5.60.0200 Calendar Days and Is In Calendear can be updated using Direct SQL.
Oct 2015	1.5	5.60.0400	JW	Update for 5.60.0400 New CONFIGVALUE function
Nov 2017	1.6	5.70.0100	JW	Update for 5.70.0100 New UserFullName and UserDepartment functions. External UserData, NEWGUID, UTC, LOCALTIME functions

CONTENTS

1 USING CALCULATIONS

InfoFlex provides a set of functions to build a formula to manipulate data.

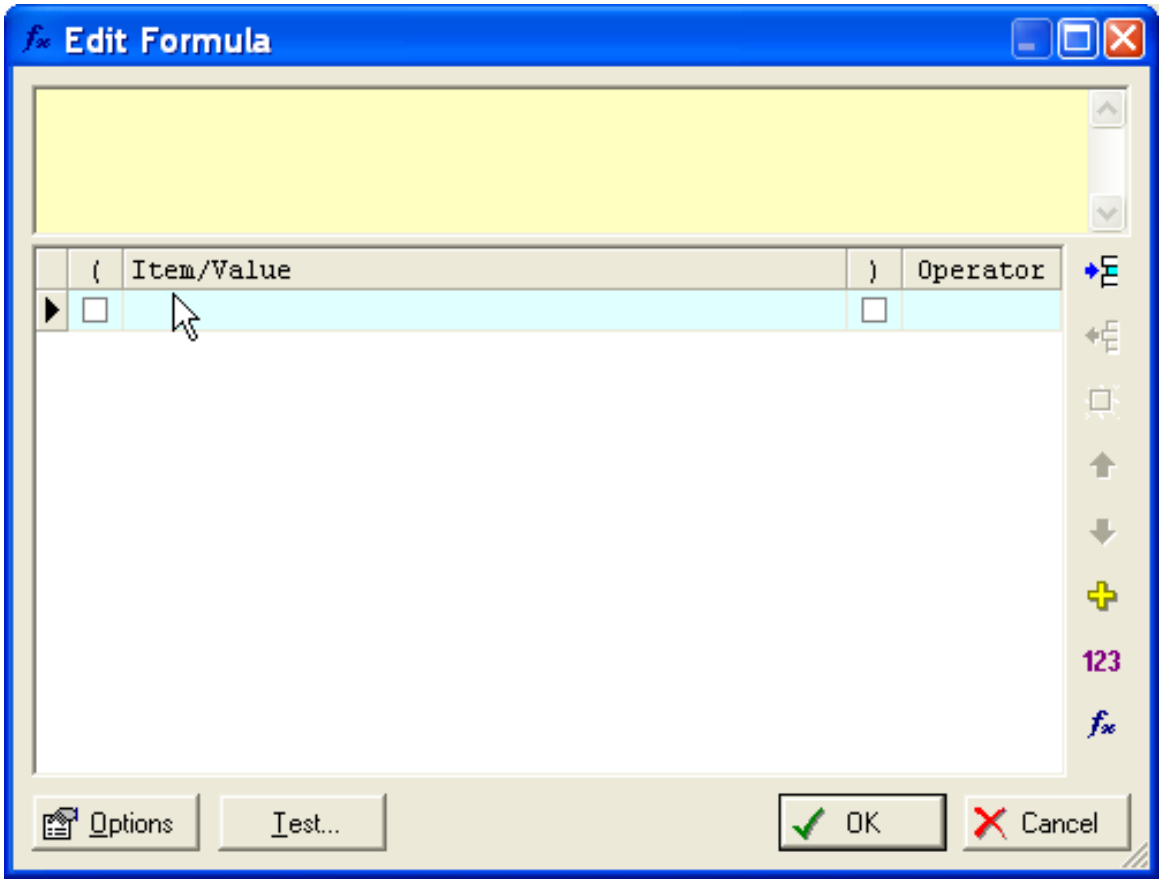
The **Edit Formula** window is used in several places in Design Management, for example:

- Calculated items
- Default Values
- Item Validations
- Event auto-creation
- Data view switching
- Branching items
- Action Conditions

The **Edit Formula** dialog is used whenever criteria containing formulae need to be defined. The formula might simply perform a calculation and enter the result in an item, or else it will test to see if data entered meets the formula and then carry out an action (eg displaying a message switching data views, generating an event, branching to another panel).

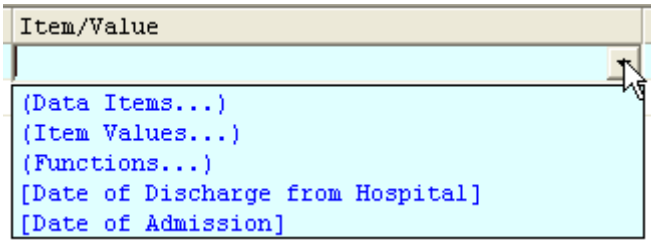
The **Edit Formula** box is always accessed by pressing the ***fx*** (Edit Formula) button.

The **Edit Formula** window is displayed.

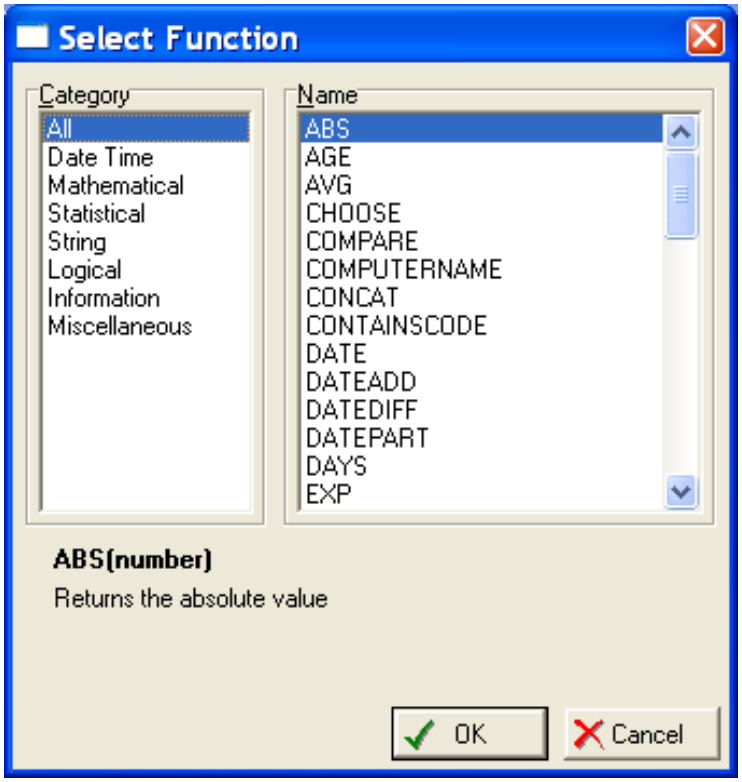


The window allows you to build up a formula row by row. The formula can be as simple or complex as you wish.

The dropdown list in the **Item/Value** column can display any or all of the following (depending on the context of the formula)



If you select **(Functions…)** a **Select Function** window allows you to select a function for use in the formula (e.g. ABS, AGE, AVG, CHOOSE). The Functions are also grouped by category to make them easier to find.



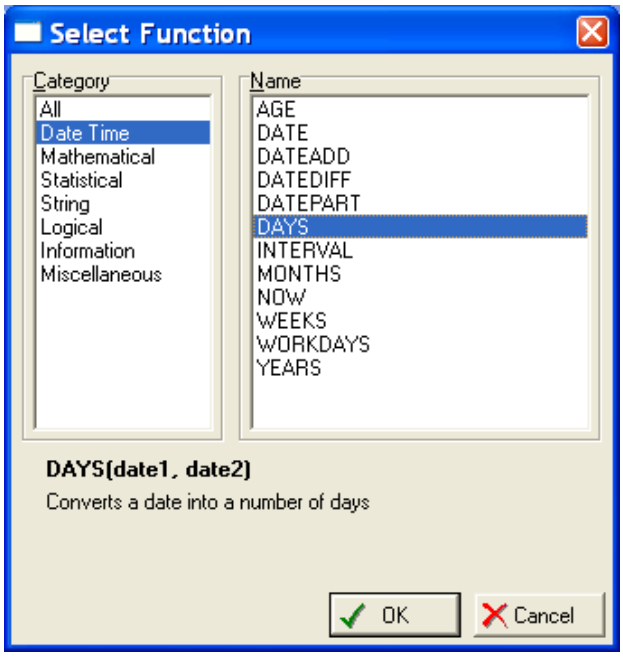
Common Terms

Formula = An expression that calculates something, based on the values in the fields, it can involve functions and operators and has to be in the correct syntax.

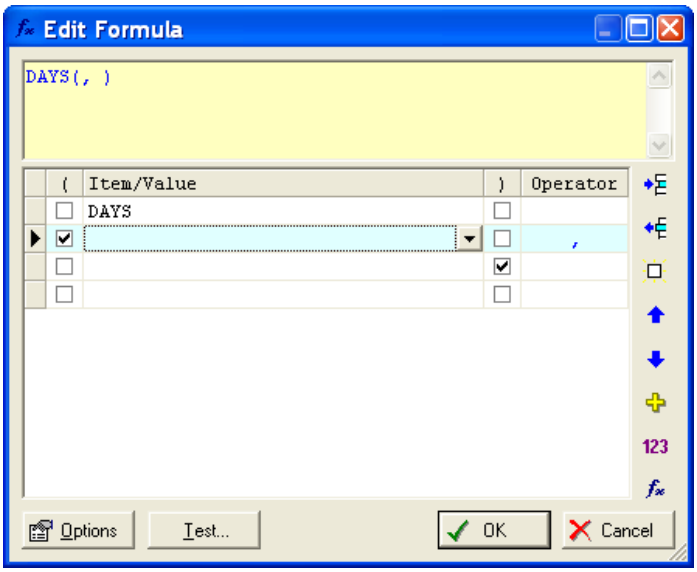
Syntax = The rules or pattern for constructing an expression or function.

Function = An expression frequently used in formulas. A function may have arguments. When we evaluate the function it returns a value.

Argument =A value that a function uses to perform operations or calculations. A placeholder for an actual expression supplied when the function is used in another formula. This means that arguments of the functions can themselves be data items, actual values or the results of other function calls. So for example, when we say that an argument is a number, it can be an actual number, a value data item, or an InfoFlex expression that evaluates to a number.



Notice when you select a function, the window displays the syntax and a brief definition e.g. **DAYS(date1, date2)** Converts a date into a number of days. 'Date1' and 'date2' are the arguments to the function. If the arguments have square brackets around them then the argument is optional.



When you select a Function, the Edit Formula automatically displays the brackets and commas that are used for the respective calculation. The arguments need to be inserted in the appropriate spaces.

Note that functions can be used within functions. When planning a complex formula such as this, it is advisable to plan it on paper first.

Take care with the position of the commas and brackets and note that they may not be positioned correctly automatically.
To assist building formula, if there is a simple syntax error such as a missing bracket, the text in the yellow box turns red to indicate that the formula is invalid.

The next section of this document will provide further clarification on all the functions available to use and their syntax.

2 INFOFLEX FUNCTIONS CATEGORY

2.1 Date and Time

2.1.1 AGE

Calculates the age in years at a selected date using a selected date of birth. If one type of format unit is used it returns a number. If more than one type of format unit is used it returns text.

Syntax: AGE(DOB, [fmt], [ref])

DOB = Date of Birth
Fmt = Optional: Output Format:

Character	Description
d	= number of days.
m	=number of months.
y	= number of years.

These are possible values that can be used in combination e.g. "ym", "yd", "ymd", "md", (note to use combinations, e.g. "ym", "ymd" you need the Calculated Age data item to be a text item). For single letter formats the Calculated Age data item can be a text or a value item. Defaults to "y".
Ref = Optional: The reference is the date against which the age is calculated. Defaults to Now.

If the argument has square brackets around then they are optional. If you leave out [fmt] and [ref] the format will default to years and the reference default to now.

Examples:

Assuming NOW	= "10/02/2010"	
AGE(23/07/1980)	= "29"	(value item)
AGE(23/07/1980, "ym")	= "29y 6m"	(text item)
AGE(23/07/1980, "d")	= "10794"	(value item)
AGE(23/07/1980, "ymd")	= "29y 6m 18d"	(text item)

If the formula looked as follows: AGE([Date of Birth], "ym", [Operation date]) the calculation would return the age in years and months of a person on the date of their operation.

2.1.2 CALENDARDAYS

Returns the number of dates in a defined calendar that occur between a start and end dates. For example the calendar might contain all the bank holidays in a year, and the function will return the number of bank holidays defined in the dictionary that fall between the start and end date.

Can be used in conjunction with the Working Days function to calculate the number of working days between two dates. The Working Days function calculates the number of days between two dates not including weekends. The Calendar Days function could be used to calculate the number of bank holidays between the two dates. The number of bank holidays could then be subtracted from the Working Days calculation to give the number of working days not including weekends or bank holidays.

Calendars Dictionary

The calendars for use with the calendar function are defined in a Calendars dictionary. The dictionary can hold one or more calendars, each calendar being a separate subject. It is therefore possible to define several different calendars, each with a different combination of dates for use with different calculated items. Multiple dictionaries can be defined if necessary.

The Calendars dictionary should have the following structure:

Root Event, primary identifier is a Text value and is the name of the calendar eg Public Holidays
Repeat Child Event, identifier is a Date Value formatted to show date only.
Each instance of the repeat event is a date in the calendar.

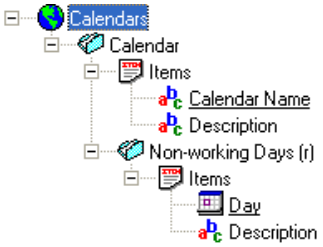
(Additional data items can be defined if necessary.)

Each subject in the dictionary is a calendar. The subject's primary identifier is used as the CalendarName parameter in the expression.

Each repeat event is a calendar date. A date-only format should be used for the repeat event identifier since time values are ignored by the function.

Example

A dictionary design looks like this:

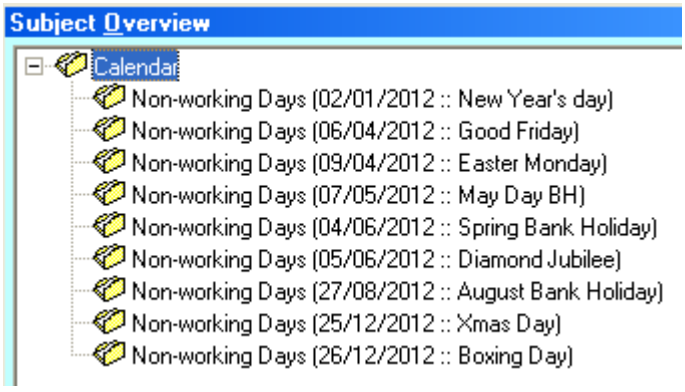


The dictionary might have several subjects, each subject being a different calendar containing a set of dates.

The subject might have several Non-working Days events.

One Non-working Days event is defined for each date.

Note that if this function is being used in conjunction with the **WorkingDays** function, weekend dates should not be added to the calendar. Only dates falling on Mon – Fri should be included to avoid the risk of counting weekend dates twice.



Syntax: CalendarDays([StartDate], [EndDate], "CalendarName")

StartDate =A date or date/time. Normally the earliest date.
EndDate = A date or date/time. Normally the latest date.
CalendarName The subject in the Calendars dictionary which contains the calendar dates to be checked against. The **CalendarName** parameter must match the subject's primary identifier in the Calendars dictionary.
1) If the dictionary definition name is **Calendars** then the **CalendarName** parameter is simply the primary identifier value of the relevant subject. For example, where the calendar dictionary is called **Calendars** and the subject's primary identifier is **Public Holidays**, the syntax would be:

CALENDARDAYS(([Start Date], [End Date], 'Public Holidays')

This expression returns the number of dates defined in the Public Holidays calendar that fall within the start and end dates. Public Holidays is a subject in the Calendars Dictionary.

2) However, if the dictionary definition name is not Calendars, the **CalendarName** parameter must include the dictionary name as well as the subject's primary identifier in the format **Dictionary.subjectID**.

For example where the calendar dictionary is called **My Dictionary** and the subject's primary identifier is **My Holidays**, the syntax would be:

CALENDARDAYS(([Start Date], [End Date], 'My Dictionary.My Holidays')

This expression returns the number dates defined in the My Holidays calendar that fall within the start and end dates. My Holidays is a subject in the My Dictionary dictionary.

Example

InfoFlex Data Item	Value for example
Date1	01/01/2012
Date2	31/12/2012

CALENDARDAYS([Date1], [Date2], "Public Holidays (England and Wales)") = 9 using the example dictionary above. (2012 had an extra Bank Holiday for the Diamond Jubilee.)

Example using CalendarDays in conjunction with WorkDays function
The CalendarDays function can be used in conjunction with WORKDAYS to calculate the number of working days between two dates. For example:

WORKDAYS([Start Date], [End Date]) – CALENDARDAYS(([Start Date],[End Date], 'Public Holidays')

The Calendar Days function calculates the number of dates in the Public Holidays dictionary that fall between the start and end dates. The **WorkDays** function calculates the number of working days that fall between the start and end dates (ie number of days not including weekends). The number of Calendar Days is then subtracted from the number of WorkDays to give the number of workdays not including weekends or public holidays defined in the calendar.

Important Notes
If additions are made to the calendar (for example more non-working days are added), the changes are not immediately applied for the data entry user since the dates are cached to enhance performance. Currently the cache is refreshed every 15 minutes, but it is advisable that users log out of InfoFlex and re-login if changes are made to the Calendars dictionary.

If the dictionary name or the subject's primary identifier do not match the parameter value in the expression, the function will return 0.

The ability to update calculated items using the Calendar functions via Direct SQL was introduced in InfoFlex version 5.60.0200. Previously these items should only have been updated using Row by Row.

2.1.3 DATE

Converts a valid date expression entered into a text field into a date field. DATE returns a date.

Syntax: DATE(expr)

Expr = Text that can be read as a valid date.

Examples:
Date("7 January 2010") - would return 07/01/2010

Notes: 1) The function will not recognize the date expression if 'th' 'st' or 'rd' come after the number.
2) The day should come before the month.

2.1.4 DATEADD

Adds a period of time onto a date, so DATEADD returns a date.

Syntax: DATEADD(interval, number, date)

Interval = The unit of the Number and the format of the result. The user should enter one of the abbreviations listed below

Abbreviations	Datepart
yyyy	**Year**
q	**quarter**
m	**Month**
y *(NB: don't use this for Year)*	**dayofyear**
d	**Day**
ww	**Week**
h *(Ifx expressions and Queries to Access)* **hh** *(Queries to SQL*)	**Hour**
n	**minute**
s	**second**

Number = the period of time to be added in relation to the Interval. Must be a whole number, if it is not an integer it will only add the whole part of it.

Date = Is the date or date-and-time to which the period of time will be added.

***Example*:**
DateAdd("d", 10, [Referral date]) would return a date 10 days after the referral date.

2.1.5 DATEDIFF

Returns the specified interval between two dates. DATEDIFF returns a number.

Syntax: DATEDIFF(interval, date1, date2)

Interval = The type of time period you are counting, e.g. are you counting days, years, hours, minutes, etc (See table above).
Date1 =A date or date/time. Normally the earliest date.
Date2 = A date or date/time. Normally the latest date.

***Examples*:**
DateDiff("d", [Referral Date], [Appointment Date]) – tells you how many days there was between the referral date and the appointment date.
DateDiff("n", [Time into theatre], [Time out of theatre]) – tells you how many minutes the subject was in theatre.

***Note*:** To get a positive number returned, put the earliest date first.

2.1.6 DATEPART

For example, what hour something happened on, or what year something happened in. DATEPART returns a number.

Syntax: DATEPART(interval, date)

Interval = The type of time period you are counting, e.g. are you counting days, years, hours, minutes, etc (See table above).
Date = A date or date/time. The date or date/time from which part will be extracted.

Examples:
Datepart("hh", [Clinic Visit Date]) – tells you the hour on which the Clinic visit date is made
Datepart("yyyy", Now()) – it gives you the current year e.g. 2010.

Notes: In Queries only, you can additionally use weekday "dw" for SQL dbs and "w" for Access dbs to return a number that corresponds to the day of the week, for example: Sunday = 1, Saturday = 7.

2.1.7 DAYS

Returns number of days between Date1 and Date2. DAYS returns a number.

Syntax: DAYS(date1, date2)

Date1 =A date or date/time. Normally the earliest date.
Date2 = A date or date/time. Normally the latest date.

Example:
Days([Onset of Symptons],[Admission Date]) –how many days there were between the onset of symptoms and the patients admission date.

***Note*:** To get a positive number returned, put the earliest date first.

2.1.8 INTERVAL

Calculates the date interval in years or other units given the start and end dates. INTERVAL returns: text.

Syntax: INTERVAL(start, [end], [fmt], [units])

Start = a date or a date-and-time. E.g. a date item.

End = Optional: a date or a date-and-time (if omitted, defaults to current date and time).
Fmt = Optional: Output format e.g. "YMDHNS", "HNS", "YMD" (if omitted, defaults to "Y")

Possible Formats

Character	Description
S	= number of seconds.
N	= number of minutes.
H	= number of hours.
D	= number of days.
M	= number of months.
Y	= number of years.

Units = Optional: The unit options (if omitted, defaults to "Short units")
The unit options are:

NO UNITS	=	No units will be displayed.
SHORT UNITS	=	y,m,d,h,m,s
MEDIUM UNITS	=	yrs,mths,dys,hrs,mins,secs
LONG UNITS	=	years, months, days, hours, minutes, seconds

Examples:
Interval("01/01/2009 00:00", "11/02/2010 10:44", "YMD", "SHORT UNITS") = 1y 1m 10d
Interval("01/01/2009 00:00", "11/02/2010 10:44", "HN", "MEDIUM UNITS") = 9754 hrs 44 mins
Interval ("01/01/2009 00:00", "11/02/2010 10:44", "YMDHN", "LONG UNITS") = 1 years 1 months 10 days 10 hours 44 minutes

2.1.9 MONTHS

Returns the interval between the month part in Date1 and Date2. MONTHS returns a number.

Syntax: MONTHS(date1, date2)

Date1 =A date or date/time. Normally the earliest date.
Date2 = A date or date/time. Normally the latest date.

Example:
Months([Onset of Symptoms],[Admission Date]) –how many months there were between the onset of symptoms and the patients admission date.

***Note*:** To get a positive number returned, put the earliest date first.

2.1.10 NOW

Returns the current date and time.

Syntax: NOW()

Example:
NOW() returns the current date and time e.g. 09/02/2010 13:55:05

2.1.11 WEEKS

Returns the number of weeks between Date1 and Date2. WEEKS returns a number.

Syntax: WEEKS(date1, date2)

Date1 =A date or date/time. Normally the earliest date.
Date2 = A date or date/time. Normally the latest date.

Example:
Weeks([Onset of Symptoms], NOW())–how many weeks there were between the onset of symptoms and today's date.
Weeks(01/10/2009, NOW()) = 18 weeks

***Note*:** To get a positive number returned, put the earliest date first.

2.1.12 WORKDAYS

Returns the number of working days between 2 dates Omits weekends but does not omit bank holidays. WORKDAYS returns a number.

Syntax: WORKDAYS(date1, date2)

Date1 =a date or date and time
Date2 = a date or date and time

Example:
Workdays(NOW(), DateAdd("d", 10, [Referral date])) – would return how many workdays there are from today to 10 days after the referral date.

2.1.13 YEARS

Returns the interval between the years part in Date1 and Date2. YEARS returns a number.

Syntax: YEARS(date1, date2)

Date1 = a date or date/time. Normally the earliest date or date and time.
Date2 = a date or date/time. Normally the latest date or date and time

***Example*:**
Years([Onset of Symptoms], [Admission Date]) –how many years there were between the onset of symptoms and the patients admission date.

***Note*:** To get a positive number returned, put the earliest date first.

2.1.14 UTC

Takes a Local time and converts it to a UTC Time (Coordinated Universal Time).

Syntax: UTC(localtime)
Localtime = a date and time (e.g. a date item in InfoFlex), in the local time

Example:

UTC([DateItem])	Date/time in UTC is returned. Result can be used as a string or as a date.
UTC(Now())	Date/time in UTC is returned. Result can be used as a string or as a date.

2.1.15 LOCALTIME

Takes a UTC time and converts it to local time.

Syntax: LOCALTIME(utc)
utc = a date and time (e.g. a date item in InfoFlex), in UTC time.

Example:

LocalTime([DateItem])	Date/time in localtime is returned. Result can be used as a string or as a date.
LocalTime(Now())	Date/time in localtime is returned. Result can be used as a string or as a date.

2.2 Mathematical

To understand the following examples in the Mathematical, Statistical and Logical Function categories, assume the following data values. The (**number**) argument refers to any expression that evaluates to a number.

InfoFlex Data Item	Value for example
Date of Birth	16-05-1967
Appointment Date	31/07/2000
Date of Referral	27/07/2000
Duration of Symptoms	06 months
Boolean Item	True
Date of this Radiotherapy	31/07/2000
Planned Number of Courses	12
Dose at this Appointment	1000
Next Session in:	02 Weeks
Other Presenting Symptoms	

Function

2.2.1 ABS

Returns the absolute value.

Syntax: ABS(number)

Example:
ABS(Duration of Symptoms) Value of Calculation: 6

2.2.2 EXP

Returns the value of e (the base of natural logarithms) raised to the power of the given number.

Syntax: EXP(number)

Examples:
Exp(0) Value of Calculation: 1
Exp([Dose at this Appointment] – [Dose at this Appointment]) Value of Calculation: 1

2.2.3 LOG

Returns the natural logarithm of the number.

Syntax: LOG(number)

Example:
Log(1) Value of Calculation: 0
Log(Planned Number of Courses) Value of Calculation: 2.484906649788

2.2.4 PERCENT

Returns the ratio of two values as a percentage.

Syntax: PERCENT (number1, number2)

Number1 = the smallest number.
Number2 = the highest number.

Example:
PERCENT(13,100) Value of Calculation = 13
PERCENT(240, 300) Value of Calculation = 80
PERCENT([Next Session in], [Duration of Symptoms]) Value of Calculation = 33.3333333333333

2.2.5 SIGN

Returns positive if positive expression, negative if negative expression and zero if neutral expression.

Syntax: SIGN(number)

Examples:

Sign("-11")	Value of Calculation: -1
Sign([Duration of Symptoms])	Value of Calculation: +1
Sign([Duration of Symptoms]-[Duration of Symptoms])	Value of Calculation: 0

2.2.6 SQROOT

Returns the square root of a number.

Syntax: SQROOT(number)

Example:

Sqroot(9) Value of Calculation: 3

2.3 Statistical

Function

2.3.1 AVG

Returns the average value in a series.

Syntax: AVG(number1, number2…)

Examples:
Avg(number1, number2, ….) Returns the average (arithmetic mean) of its arguments
Avg([Duration of Symptoms], [Duration of Symptoms]) If Duration of Symptoms equals 6 months then value of calculation is 6
Avg(100, 150, 200, 250) Value of Calculation: 175

2.3.2 MAX

Returns the maximum value in a series. The MAX function also works on dates and texts.

Syntax: MAX(argument1…)

Examples:
Max("02/03/2010", "02/05/2009", "20/08/2007") = returns 02/03/2010
Max("apple", "pear", "fruit") **=** returns 'pear'. The function determines the higher value in relation to where the letters in the word fall in the alphabet. Capital letters has a lower value than a lower cased letter.
Max(-11,5,3) Value of Calculation: 5
Max(12, 24) Value of Calculation: 24

2.3.3 MIN

Syntax: MIN(argument1, argument2 ...)

Returns the minimum value in a series. Argument can return a number, date, or text.

Examples:
Min("02/03/2010", "02/05/2009", "20/08/2007") = returns 20/08/2007
Max("apple", "pear", "fruit") **=** returns apple. The function determines the higher value in relation to where the letters in the word fall in the alphabet. Capital letters has a lower value than a lower cased letter.
Min(-11,5,3) =Value of Calculation: -11
Min(12, 24) =Value of Calculation: 12

2.4 String

Function

The arguments in the following functions have to be text strings but they can come from other data items, or an actual bit of text, or from the results of another function.

2.4.1 COMPARE

Compares two text strings and returns a value indicating result (returns 0 if the texts are equal, returns -1 if text1 is less than text2 and returns 1 if text1 is greater than text2). A text is less or greater than another with regards to the position of the letters in the alphabet. The order being compared against is 1,2,3…, A,B,C,… a,b,c,… e.g. 'C' is greater (later) than 'A' but 'a' is greater (later) than C.

Syntax: COMPARE (text1, text2, [case_sensitive])

Text1 = text
Text2 = text
Case sensitive = Optional: If set to 0, the case is ignored when comparing two text value. If set to 1, the comparison is case-sensitive. Defaults to 1.

Example:
Compare("Abbot", "Abbot", 0) = 0
Compare("Abbot", "Bromley", 0) = -1
Compare("Bromley", "Abbot", 0) = 1
Compare("Bromley", "abbot", 1) =-1

2.4.2 CONCAT

Returns concatenation of two text strings i.e. it joins two bits of text together.
The arguments have to be text strings but they can come from other data items, or an actual bit of text, or from the results of another function. CONCAT returns text.

Syntax: CONCAT(text1, text2)

Examples:
Concat("Hello ", [Forename]) = "Hello Jonathan" (note the space required after "Hello ")
Concat([Title], Concat(" ", [Surname])) = "Mr Ellis" (note the space between quotations)

Note: Spaces are not automatically inserted between the two concatenated text strings. If spaces are required these should be included in the expression. If two data items are being concatenated, the space will need to be inserted via an additional concatenation function (see second example).

2.4.3 FIND

Returns how many characters there are from the start position in a text string to the first occurrence of the text you are searching for (includes spaces). FIND returns a number.

Syntax: FIND(text, match_text, [start_pos])

Text = The whole piece of text you are searching in
Match_text = The part of the text you are searching for. Note that the match text search is case sensitive.
Start_pos = Optional: This specifies the position in the text string to start searching from. A value of '1' will start searching from the first character of the text string, '2' the second character etc. Defaults to 1.

Example:
Find("Headache and Nausea", "ache", 1) = 5

2.4.4 FINDREV

Returns the position of the last occurrence of a match string within a text string (includes spaces).

Syntax: FINDREV(text, match_text, [start_pos])

Text = The whole text you are searching in

Match_text = The part of the text you are searching for. Note that the match text search is case sensitive.

Start_pos = Optional: This specifies where the search in the text string will start from (defaults to last character of the text).

Example

FindRev("Headache and Nausea", "ache",) =5

FindRev("Headache and Stomach ache", "ache") =22

Note: the returned number relates to the position of the first character of the match_text in the text string.

2.4.5 FORMAT

Converts a value to a string and applies a format or mask. The return varies depending on the value being formatted.

Syntax: FORMAT(value, mask)

Value =Any valid expression.

Mask =Format of resulting string.
The mask can include the following strings:

Numeric values:

Character	Description
0	Digit placeholder. Display a digit or a zero.
#	Digit placeholder. Display a digit or nothing.
.	Decimal placeholder.
%	Percentage placeholder.
,	Thousand separator.
E- E+ e- e+	Scientific format.

Date values:

Character	Description
:	Time separator.
/	Date separator.
C	Format the date as ddddd and Format the time as ttttt, in that order.
D	Format the day as a number without a leading zero (1 - 31).
DD	Format the day as a number with a leading zero (01 - 31).
DDD	Format the day as an abbreviation (Sun - Sat).
DDDD	Format the day as a full name (Sunday - Saturday).
DDDDD	Format the date according to your system's short date format setting.
DDDDDD	Format the date according to the long date setting recognized by your system.
W	Format the day of the week as a number (1 for Sunday through 7 for Saturday).
WW	Format the week of the year as a number (1 - 54).
M	Format the month as a number without a leading zero (1 - 12).
MM	Format the month as a number with a leading zero (01 - 12).
MMM	Format the month as an abbreviation (Jan - Dec).
MMMM	Format the month as a full month name (January - December).
Q	Format the quarter of the year as a number (1 - 4).
Y	Format the day of the year as a number (1 - 366).
YY	Format the year as a 2-digit number (00 - 99).
YYYY	Format the year as a 4-digit number (100 - 9999).
H	Format the hour as a number without leading zeros (0 - 23).
HH	Format the hour as a number with leading zeros (00 - 23).
N	Format the minute as a number without leading zeros (0 - 59).
NN	Format the minute as a number with leading zeros (00 - 59).
S	Format the second as a number without leading zeros (0 - 59).
SS	Format the second as a number with leading zeros (00 - 59).

Text values:

Character	Description
@	Character placeholder. Display a character or a space.
&	Character placeholder. Display a character or nothing.
<	Force lowercase. Display all characters in lowercase format.
>	Force uppercase. Display all characters in uppercase format.
!	Force left to right fill of placeholders. The default is to fill placeholders from right to left.

Examples:
FORMAT([Date of Diagnosis], "YYMMDD") = if Date value was 18/02/2010 the result of the calculation would be "100218".
FORMAT("HEADache and NAusea", "<") = headache and nausea

2.4.6 LEFT

Returns the specified number of characters from start of a text string.

Syntax: LEFT(text, num_chars)

Text = The whole piece of text to return characters from.
Num_chars = a whole number equaling the specified number of characters beginning from the left of the text.

Example
LEFT("NHSNUMBER", 3) = "NHS"

2.4.7 LENGTH

Returns the length of a text string, i.e. how many characters there are in the text (includes spaces).

Syntax: LENGTH(text),

Example
Length ("Headache and Nausea") = 19

2.4.8 LOWER

Converts a text string to lower case.

Syntax: LOWER(text**)**

Example
Lower(EXAMPLE) = example

2.4.9 PROPER

Converts a text string to proper case.

Syntax: PROPER(text)

Example
Proper(EXAMPLE) = Example
Proper(example) = Example

2.4.10 REPLACE

Replaces all occurrences of one text string (match_text) in the given text string (text) with another text string (subst_text). Note that the match text search is case sensitive.

Syntax: REPLACE(text, match_text, subst_text)

Text = The whole piece of text.
Match_text = all occurrences of this test will be replaced by the substitute text.
Subst_text = The text that will replace the match text.

Example
Replace("Headache and Nausea", "and", "or") = "Headache or Nausea"

2.4.11 RIGHT

Returns the specified numbers of characters from the end of a text string.

Syntax: RIGHT(text, num_chars)
Text = The whole piece of text to return characters.
Num_chars = a whole number equaling that returns the specified number of characters beginning from the right of the text.

Example
Right("NHSNUMBER", 6) = "NUMBER"

2.4.12 SUBSTR

The substring returns a specified number of characters of a given text from a specified start position in the text.

Syntax: SUBSTR(text, start_pos, length)

Text = The whole piece of text
Start_pos = a whole number that refers to the start position of the substring within a text.
Length = The length of the text string to return.

Example
SUBSTR("Headache and Nausea", 5, 15) = "ache and Nausea"
SUBSTR("Headache and Nausea", 10, 15) = "and Nausea"

2.4.13 SUBSTRING

Returns a portion of the given text string from a specified start position to a specified end position.

Syntax: SUBSTRING (text, start_pos, end_pos)

Text = text
Start_pos = a whole number that refers to the start position of the substring within a text.
End_pos = a whole number that refers to the end position of the substring within a text.

Example
SUBSTRING("Headache and Nausea", 10, 12) = "and" (the start_pos '10' refers to the 10th character of the text which equals 'a', the end_pos '12' refers to the 12th character of the text which equals 'd')
SUBSTRING("Headache and Nausea", 5, 8) = "ache" (5th character = "a", 8th character = "e")

2.4.14 TRIM

Removes leading and trailing spaces from the given text string.

Syntax: TRIM(text)

Example
Trim(“ hello ”) = “hello”

2.4.15 UPPER

Converts a text string to upper case.

Syntax: UPPER(text)

Example
Upper(example) = EXAMPLE

2.5 Logical

2.5.1 CHOOSE

Returns argument at the indexed position in the argument list. The index starts from one. The function is used to simplify some long calculations.

Syntax: CHOOSE(index, arg1,[arg2],…)

Index = The first index argument specifies which value to return from the subsequent argument list. An index of one would return the first item in the subsequent argument list, an index of two would return the second, etc.

Arg = Arg1, Arg2, etc. A series of any type of items or expressions.

Example:
CHOOSE([Value], "Jan", "Feb", "Mar", "Apr", "May")

If the value of [Value] was 1 then the function would return "Jan", if the value was 2 then return value would be "Feb", etc.

This function is designed to make expressions easier to write that currently take the form. IIF([Value] = 1, "Jan", IIF([Value] = 2, "Feb", IIF([Value] = 3, "Mar", …)))

2.5.2 CONTAINSCODE

This allows testing for a code in multiple response coded items. Returns a Boolean result: True if a list of codes contained a specified code, False if it does not.

Syntax: CONTAINSCODE(code_list, code, [delimiter])

Code_list = List of delimited codes to test
Code =Single code to search for
Delimiter = Optional: Default is a semi-colon between codes in the list

Examples:
CONTAINSCODE([Presenting Symptom], "2") if the list of Presenting symptoms contains 2 then returns True
CONTAINSCODE([Presenting Symptom], "2") if the list of Presenting symptoms doesn't contains 2 then returns False.

2.5.3 IIF

This function returns an expression depending on the truth value of a test. It may return any type of item depending on the expression given.

Syntax: IIF(test_expr, true_expr, false_expr)

Test_expr = The truth value of this expression determines the outcome of what the function will return.
True_expr = If the Test expression is true then the IIF function will return this expression.
False_expr = If the Test expression is false then the IIF function will return this expression**.** It is sometimes convenient to omit this argument to indicate that the expression returns a null value if the initial argument evaluates to false

Examples:

InfoFlex Data Item	Value for example
Boolean item	True or False
Appointment Date	31/07/2000
Date of Referral	27/07/2000

IIF(Isblank(""), "Yes", "No") = Yes
IIF([Boolean item], [Appointment Date], [Date of Referral])=31/07/2000 when [Boolean item] is true.
IIF([Boolean item], [Appointment Date], [Date of Referral])=27/07/2000 when [Boolean item] is False.
IIF(3 > 2, Yes, No) = Yes

2.5.4 ISBLANK

Returns True if the expression is Null or an empty string and false if the expression has a value.

Syntax: ISBlank(expr)

Expr = Any valid expression.

Examples:

InfoFlex Data Item	Value for example
Next Session in	02 Weeks
Other Presenting Symptoms	

Isblank("") = True
Isblank([Next Session in]) = False
Isblank([Other Presenting Symptoms]) = True

Note: The option described in section **2.8 Formula Options for Blank, Missing and Unknown values** must be switched on when using this function.

2.5.5 ISINCALENDAR

Returns true if the specified date exists in the named calendar. This function uses calendars defined in the Calendars dictionary described in the CalendarDays function in 2.1.2 above.

Syntax: IsInCalendar(Date1, CalendarName)

Date = The date being checked.
CalendarName = The subject in the Calendars dictionary which contains the calendar dates to be checked against.

Examples:
ISINCALENDAR([Date Item], "Public Holidays") = True if the date in the Date Item exists as a non-working day in the Public Holidays calendar.

Notes
If additions are made to the calendar (for example more non-working days are added), the changes are not immediately applied for the data entry user since the dates are cached to enhance performance. Currently the cache is refreshed every 15 minutes, but it is advisable that users log out of InfoFlex and re-login if changes are made to the Calendars dictionary.

If the dictionary name or the subject's primary identifier do not match the parameter value in the expression, the function will return 0.

The ability to update calculated items using the Calendar functions via Direct SQL was introduced in InfoFlex version 5.60.0200. Previously these items should only have been updated using Row by Row.

2.5.6 ISMISSING AND ISUNKNOWN

The result will be True if the given argument is set to Missing or to Unknown (respectively), False otherwise.

Syntax: ISMISSING(value), and ISUNKNOWN(value)

Value = a valid expression
Example
IIF(ISMISSING([1st available appt]) OR ISUNKNOWN([1st available appt]) OR ISBLANK([1st available appt]), NOW(), [1st available appt])
If [1st available appt] was blank, missing or unknown then it will return today's date.

Note: The option described in section **2.8 Formula Options for Blank, Missing and Unknown values** must be switched on when using this function.

2.5.7 LIKE

The result will be True if the pattern is found in the string, False otherwise.

Syntax: LIKE(string, pattern, [case_sensitive])

String = a valid expression
Pattern = A particular pattern that is searched for in the string. The pattern can contain wildcards ("*" and "?"), and character lists "{abc}" or character ranges "{a-d}".
Case sensitive = If set to 0, this can be used to ignore case when comparing two text values, by default the comparison is case-sensitive which is equivalent to 1.

Examples:

LIKE([item], "123{A-Z}") returns True if [item] = "123A", "123B", ….
LIKE([item], "123{abc}") returns True only if [item] = "123a" or "123b" or "123c"
LIKE([item], "{0-9}{a-z}) returns True if first number between 0 and 9 and letter between a-z e.g. "5f" or "4t" or "9e".
LIKE([item], "{0-9}{0-9}{0-9}{a-z}") will return True if any 3 digit number and letter e,g, "354g" or "793d" or "000k"
LIKE([item], "?") would match any one-character length string.
LIKE([item], "A*") would match any string starting with the letter A

There is an optional third argument of 1 for case-sensitive, and 0 for not case-sensitive.
For example

LIKE([item], "123{ab}", 0) would return True if [item] = "123A", "123B", "123a" and "123b".

2.5.8 NOT

This function will return True if expression is False and vice versa. This function can be used in conjunction with functions that return a Boolean result such as **LIKE**, **CONTAINSCODE**, **ISBLANK**, **NOTBLANK**, **ISMISSING**, **ISUNKNOWN**, etc.

Syntax: NOT(expr)

Expr = A valid piece of text.

Examples:

NOT(CONTAINSCODE([MR Coded Item], "3")) Returns False if coded item equals 3 and True if equal to another value.
NOT(ISBLANK([Item])) Returns False if item is blank and True if item has a value. *** this is the same as NOTBLANK([Item]).
NOT(NOTBLANK([Item])) *** this is the same as ISBLANK([Item])
NOT([Item1] = [Item2]) *** this is the same as [Item1] <> [Item2]

2.5.9 NOTBLANK

Returns True if the Expression has a value, False if it is NULL or an empty string.

Syntax: NOTBLANK(expr)

Expr = Any valid expression.

Examples:

InfoFlex Data Item	Value for example
Date of this Radiotherapy	31/07/2000
Planned Number of Courses	12
Other Presenting Symptoms	

Notblank([Date of this Radiotherapy]) =True
Notblank([Planned Number of Courses]) =True
Notblank([Other Presenting Symptoms]) =False

Note: Please see Section **2.8 Formula Options for Blank, Missing and Unknown values** when using this function.

2.5.10 VALIDNHSNUMBER

This validates v2 NHS numbers. These must be 10 digits, with no other characters or spaces and the check digit must satisfy the specification. The function returns True or False.

Syntax: VALIDNHSNUMBER()

Example:
VALIDNHSNUMBER("1111111111") = True
VALIDNHSNUMBER("123456789") = False

2.6 Information

Function Syntax

2.6.1 COMPUTERNAME

Returns the computer name that InfoFlex is running on.

Syntax: COMPUTERNAME()

Example:
COMPUTERNAME() = MyComputer

2.6.2 USER

Returns the Infoflex user name

Syntax: USER()

Example;
USER() = system

2.6.3 USERGROUPS

Returns a comma-separated list of the Infoflex user's group(s)

Syntax: USERGROUPS()

Example:
USERGROUPS() = Administrators,CWT,Heart

2.6.4 WINUSER

Returns the Windows user name

Syntax: WINUSER()

Example:
WINUSER() = jellis

2.6.5 USERFULLNAME

Returns the InfoFlex user's full name

Syntax: USERFULLNAME()

Example:
USERFULLNAME() = Jane Smith

2.6.6 USERDEPARTMENT

Returns the InfoFlex user's department

Syntax: USERDEPARTMENT()

Example:
USERDEPARTMENT() = Information Department

2.6.7 EXTERNALUSERDATA

Returns data about the currently logged in external user of InfoFlex Web. In InfoFlex v5 this function returns a blank string or the default value, if one defined.

Syntax: ExternalUserData(ParameterColName, v5 defaultValue)

name the external user name
default optional default value to return if the requested value is not found

If it is run from an External User InfoFlex Web application, it returns the data that is in the **ParameterColName** column of the Ifx_ExternalUsersData table for the currently logged on user (often this would be 'Parameter1'). This data is defined via the Data Portal Add-in.

If it is run in InfoFlex v5 it returns the **v5defaultValue** for any user.

ExternalUserData is also available as a default value in Query Design Manager Filter Definition. If selected the user will be presented with `ExternalUserData("Parameter1","")` in which they can overtype the parameter name and default value for the function.

2.7 Miscellaneous

Function

2.7.1 NEXTVALUE

Returns the next number in a defined sequence. It provides a new value every time the function is called. The start number can be selected, and it can be concatenated with other text or numbers to provide a unique label across the whole domain.

To use this feature, create a value item or text item and set up a calculation. The item should be set to **Default Calculation** and with a value item there should be a mask set on the **Advanced** tab. The item needs to be a default calculation, because the value does not need to be updated once it has been generated. If it was in an ordinary calculated item the function would be called each time the calculated items in the event were refreshed, and so the value would keep incrementing.

Running the function, even testing it from the Edit Formula dialog, will always increment the next number, so a warning has been put in Test Formula dialog - this allows you to cancel the test if you do not want the number to be incremented.

To add text or further numbers around the generated number, you can use the NextValue function within a longer expression and concatenate it with other text.

Syntax: NEXTVALUE([valuename], [minvalue], [owner])

Valuename =The first argument is a reference field controlling each instance of the use of the nextvalue function (e.g. radiotherapy and chemotherapy might have different but parallel sequences of numbers needing a next value). Note that you do not see this valuename in the generated number, it is an internal reference so that different series of numbers can be kept separate. It can be text or an expression e.g. format(Now(), "yyyy") which gives the four-digit current year. This can be used if the sequence needs to restart at the beginning of each year. The valuename should be unique for each series of numbers you want to generate.

Minvalue, =The second argument is the initial value of the sequence. The first time the function with the same Valuename is called, this initial value is returned, and each subsequent time, the value returned is incremented. The initial value must not be a negative number.

Owner = The third argument is optional and is not currently used.

Example:
NextValue("Rad", 1) = 1,2,3…..

When using the NextValue function in a text item, you can ensure that it has a set length by using the format function, e.g.
Format(NextValue("Chem", 1), "0000")
This would ensure that the values returned were four digit, i.e. "0001", "0002", "0003", etc.

It is also possible to concatenate the sequence to add a prefix to the number.
Concat(Format(NOW(), "yyyy"), NextValue(Concat("RAD", Format(NOW(), "yyyy")), 6000))
This generates a series of numbers prefixed with the current year and starting at 6000. E.g. 20086000, 20086001, etc. The valuename of the series concatenates RAD with the current year, so the valuename during 2008 is "Rad2008". When 2009 is reached, the valuename changes to"Rad2009" and the numbering starts at 6000 again for this new series, i.e. 20096000, 20096001, etc.

2.7.2 NEWID

Generates a unique identifier. When called it generates a unique reference number called a GUID (globally unique identifier). This is displayed as 32 hexadecimal digits with hyphens. Consequently the result needs to be stored in an InfoFlex text item.

The function returns a new GUID every time it is evaluated. It can be used where a unique reference number is required, but be aware that if the value needs to be created once and then not updated, it will need to be set up as a default calculation (similar to the NextValue function), otherwise it would change every time the event is evaluated.

Note that currently it is necessary to refresh **NewID** calculations using row-by-row refresh rather than Direct SQL.

Syntax: NewID()

Example:
NewID() = 6575E161-65D9-427C-AF11-031AD80C6F32

2.7.3 HASHVALUE

Encrypts an existing value according to a specific algorithm. Any value can be encrypted.

The result is a hexadecimal value and so needs to be stored in an InfoFlex text item. Hashvalue always returns the same value for the same inputs.

Hash-values use the underlying stored value in the InfoFlex data item, so that the value does not change if different formats are applied in data entry. For example:

Text items	HashValue is case-sensitive. The hash value is based on the case stored when the value was saved to the database, even if the format chosen to display the data item is different.
Date items	The hash value is based on the date formatted in the following way: dd/mm/yyyy hh:nn:ss, even if the item has a different date format in data entry.
Boolean items	The hash value is based on 1 and 0 even if the boolean is formatted as True/False, On/Off etc.
Value items	The hash value is based on the actual value stored in the database.
Coded/Dictionary Lookup items	The hash value is based on the code not the meaning.

If necessary, functions such as **Format** and **Upper/Lower/Proper** can be used in the calculation to ensure that a particular format or case is applied to the value used in the hashvalue function.

Note that currently it is necessary to refresh **HashValue** calculations using row-by-row refresh rather than Direct SQL.

Syntax: HASHVALUE(arg1, algorithm)

arg1 = any value (e.g. an InfoFlex data item)
algorithm = any of the following

- MD2
- MD4
- MD5
- SHA1
- SHA256
- SHA512

Example
HashValue([Surname], MD2) = 5C555612C071D9777A89EB8D31C35201
HashValue(Upper([Surname]), MD2) = 3DCFCA726994D93B675BDAFC7CEC33F3

2.7.4 CONFIGVALUE

Returns values from the Web configuration in .NET to be used in InfoFlex data expressions. The function is designed to allow a different value to be returned when the expression is evaluated on an InfoFlex web application, compared to evaluating the expression within InfoFlex v5, allowing the possibility of distinguishing between events saved from InfoFlex v5 and events saved from an InfoFlex web application.

In InfoFlex v5 the function returns either blank or the default value specified as an optional argument.

Syntax: CONFIGVALUE(name, [default])

name the value name in the configuration
default optional default value to return if the requested value is not found

If it is run from InfoFlex v5, the return value is the **default** value specified as the second (optional) argument. If this value does not exist then a blank value is returned.

On InfoFlex Web, the **name** argument should be added as a key in the AppSettings.config, and the value that should be returned when ConfigValue(name) is evaluated on the web application is the value for that key.

2.7.5 NEWGUID FUNCTION

The current NEWID function removes hyphens from the generated Guid. This NEWGUID function leaves in the hyphens and returns a true GUID.

Syntax: NEWGUID()

Example:
NEWGUID() = 6575E161-65D9-427C-AF11-031AD80C6F32

2.8 Formula Options for Blank, Missing and Unknown values

Where an expression includes items whose values are set to blank, missing or unknown, then the default behaviour is for the expression to return a blank, missing or unknown value respectively. If it is required that an entry of blank, missing or unknown must not affect the result of the expression, then the option described below can be used to ensure that a value is returned where a contributory item is blank, missing or unknown.

Example 1

The following calculation uses value items. If the option is switched on, a value is still returned where one of the contributory items is blank, missing or unknown. If the option is switched off, a value is only returned if both the contributory items have numeric values.

[Number1] + [Number2]

[Number1] value	[Number2] value	Result with option switched on	Result with option switched off
100	25	125	125
100	[blank]	100	[blank]
100	[missing]	100	[missing]
100	[unknown]	100	[unknown]
[blank]	[blank]	0	[blank]
[missing]	[missing]	0	[missing]
[unknown]	[unknown]	0	[unknown]

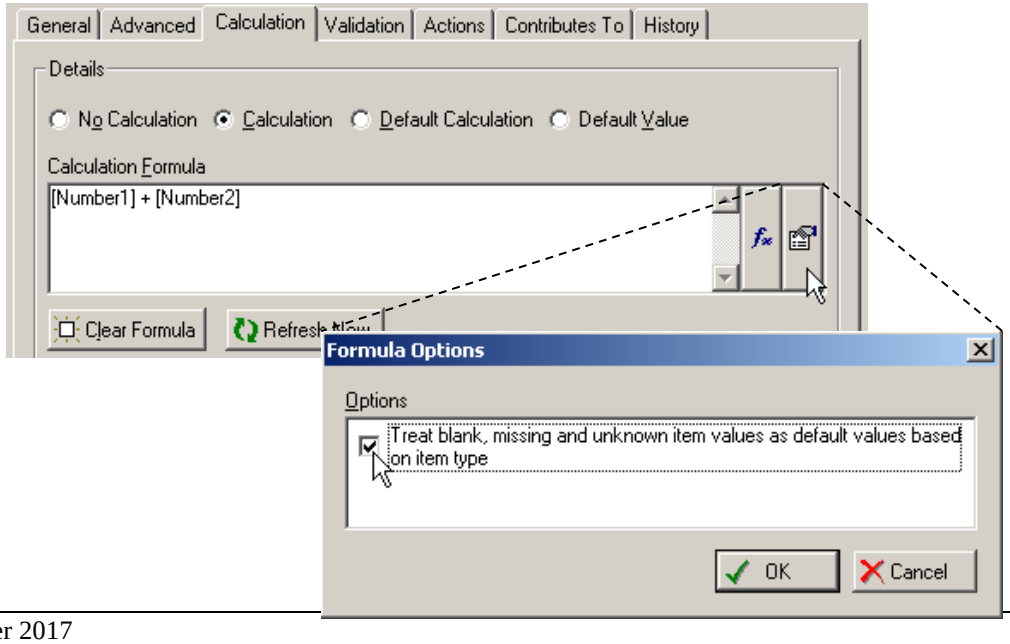
Example 2

The following calculation uses text items. If the option is switched on, a value is still returned where one of the contributory items is missing or unknown. If the option is switched off, a value is not returned if either of the contributory items is missing or unknown.

Concat([TextItem1],[TextItem2])

[TextItem1] value	[TextItem2] value	Result with option switched on	Result with option switched off
Jane	Smith	JaneSmith	JaneSmith
Jane	[blank]	Jane	Jane
Jane	[missing]	Jane	[missing]
Jane	[unknown]	Jane	[unknown]

The option in question is **Treat blank, missing and unknown item values as default values based on item type** and it can be set from the **Edit Options** button on the **Calculation** tab, or from the **Options** button on the **Edit formula** dialog.



Setting the option

No Tick Expression evaluation behaves as usual (i.e. the expression will evaluate to blank, missing or unknown if any of the items it references are set to blank, missing or unknown).

Tick Blank, missing or unknown values in an expression will be substituted with a zero or blank according to the item type.

ISBLANK, ISMISSING and ISUNKNOWN functions

Where the functions ISBBLANK, ISMISSING and ISUNKNOWN are used in an expression, the option must be ticked. Otherwise blank, missing and unknown items are not passed into the expression for evaluation.

Example

Expression	[Item1] value	Result with option switched on	Result with option switched off
ISBLANK([Item1])	Text	false	false
ISBLANK([Item1])	Blank	true	true
ISBLANK([Item1])	Missing	true	false
ISBLANK([Item1])	Unknown	true	false
ISMISSING([Item1])	Text	false	false
ISMISSING([Item1])	Blank	false	false
ISMISSING([Item1])	Missing	true	false
ISMISSING([Item1])	Unknown	false	false
ISUNKNOWN([Item1])	Text	false	false
ISUNKNOWN([Item1])	Blank	false	false
ISUNKNOWN([Item1])	Missing	false	false
ISUNKNOWN([Item1])	Unknown	true	false

Note that for the ISBLANK function, blank, missing and unknown values are all treated as blank values when the option is switched on.