Appendix
Working with formulas in FLIR Thermal Studio

Table of contents

1 Legal disclaimer

1.1 Legal disclaimer

All products manufactured by FLIR Systems are warranted against defective materials and workmanship for a period of one (1) year from the delivery date of the original purchase, provided such products have been under normal storage, use and service, and in accordance with FLIR Systems instruction.

Products which are not manufactured by FLIR Systems but included in systems delivered by FLIR Systems to the original purchaser, carry the warranty, if any, of the particular supplier only. FLIR Systems has no responsibility whatsoever for such products.

The warranty extends only to the original purchaser and is not transferable. It is not applicable to any product which has been subjected to misuse, neglect, accident or abnormal conditions of operation. Expendable parts are excluded from the warranty.

In the case of a defect in a product covered by this warranty the product must not be further used in order to prevent additional damage. The purchaser shall promptly report any defect to FLIR Systems or this warranty will not apply.

FLIR Systems will, at its option, repair or replace any such defective product free of charge if, upon inspection, it proves to be defective in material or workmanship and provided that it is returned to FLIR Systems within the said one-year period.

FLIR Systems has no other obligation or liability for defects than those set forth above.

No other warranty is expressed or implied. FLIR Systems specifically disclaims the implied warranties of merchantability and fitness for a particular purpose.

FLIR Systems shall not be liable for any direct, indirect, special, incidental or consequential loss or damage, whether based on contract, tort or any other legal theory.

This warranty shall be governed by Swedish law.

Any dispute, controversy or claim arising out of or in connection with this warranty, shall be finally settled by arbitration in accordance with the Rules of the Arbitration Institute of the Stockholm Chamber of Commerce. The place of arbitration shall be Stockholm. The language to be used in the arbitral proceedings shall be English.

1.2 Usage statistics

FLIR Systems reserves the right to gather anonymous usage statistics to help maintain and improve the quality of our software and services.

1.3 Copyright

1.4 Quality assurance

The Quality Management System under which these products are developed and manufactured has been certified in accordance with the ISO 9001 standard.

FLIR Systems is committed to a policy of continuous development; therefore we reserve the right to make changes and improvements on any of the products without prior notice.

2 Introduction

The FLIR Thermal Studio application comes with a set of predefined formulas. If these formulas do not meet your requirements, you can create your own. Create them from scratch or use one of the predefined formula as a basis.

The formulas are created, edited, imported, and exported in the *Formula editor*. For more information, see the FLIR Thermal Studio User's manual.

3 Basics

3.1 General

Expressions are written in a formal language and provide a powerful and concise way to find complex patterns inside text. The expression supports setting and getting of property values, property assignment, method invocation, accessing the context of arrays, collections and indexers, logical and arithmetic operators, named variables, and retrieval of objects by name. It also supports list projection and selection, as well as common list aggregators.

3.2 Expressions

A variable is a word that represents an object. If you want to refer to a specific property of that object, add the representation of that property to the variable separated by a dot, for example *Image.Description* and *Page.Number*.

For a list of all variable names, property names, and methods, see 6 *Dictionary*, page 10.

Note

- All variables are case sensitive.
- A property that doesn't have a value expression will be evaluated as literal text.

3.3 Null context

If you do not specify a root object, then the expressions evaluated either have to be literal values (i.e. *2 + 3.14*) , refer to static/common methods or properties(i.e. *DateTime.Today, Max, Minimum*), create new instances of objects (*new DateTime(2004,8,14)*), or refer to other objects such as those in the variable dictionary.

4 Language references

4.1 Literal expressions

Supported types of literal expressions are:

- strings
- dates
- numeric values
- Boolean
- null

String are delimited by single quotes. To put a single quote itself in a string, use the back-slash character. The following table shows simple usage of literal expressions.

EXPRESSION	EVALS TO
Hello World	Hello World
6.0221415E+7	60221415
0x7FFFFFFF	2147483647
date('1974/08/24')	24-Aug-74 12:00:00 AM
true	1
null	null

4.2 Properties

Get values by using the following syntax: *Variable.Property*.

Example:

- *Image.Description*
- *Image.Statistics.Min*

It is possible to assign a property value for the current reporting page, for example change color of the object border:

Example:

- *Page.Border = GetColor(255,0,255)*

For a list of all variable names, property names, and methods, see 6 *Dictionary*, page 10.

4.3 Collections

4.3.1 Lists

A list represents object collections, for example measurements in an image. Access to list variables is the same as for properties.

4.3.2 Indexer

Contents of collections is obtained by specifying integer value within the brackets.

Example:

- *Image.Measurements.MeasurementLines[0].Min*

Also, it is possible to access contents from the collections using a literal key value within the brackets (note that you can use literal key as a indexer only in specified collections, see 6 *Dictionary*, page 10).

In the example above we access the measurement by the measurement name. You may also specify non literal values in place of the quoted literal values by using another expression inside the square brackets, such as variable names or static properties/ methods.

Example:

- *Image.Measurements['Sp1']* where Sp1 = measurement name.

The most important objects are accessible with custom pattern, for more details, see 6 *Dictionary*, page 10.

Example:

- *Sp1* where Sp1 is equivalent to *Image.Measurements['Sp1']*.

4.3.3 Defining Arrays, Lists, and Dictionaries inline

Inline lists are defined by enclosing a comma separated list of items within curly brackets.

Example:

- *{1, 2, 3, 4, 5}*
- *{'abc', 'xyz'}*

Dictionary definition syntax is a bit different: you need to use a # prefix to tell the expression parser to expect key/value pairs within the brackets and to specify a comma separated list of key/value pairs within the brackets. Arrays, lists and dictionaries created this way can be used anywhere where arrays, lists and dictionaries obtained from the object graph can be used. Keep in mind that even though the examples above use literals as array/list elements and dictionary keys and values, that's only to simplify the examples – you can use any valid expression wherever literals are used.

Example:

- *#{1 : 'January', 2 : 'February' ...}*
- *#{'key1' : DateTime.Today, 'key2' : 'Value 2'}*

4.4 Methods

Methods use variables (or none) as parameters and return new processed data.

Syntax: *MethodName(Variable1,Variable2...VariableN)*

For proper calculation, insert round brackets on both sides of the method:

Example:

- *'Log(1, 10) - 5*' returns wrong value: 0
- *(Log(1,10)) - 5'* returns correct value: -5

You may also invoke methods on literals or variables.

Example:

- *'flir'.IndexOf('s')* - evaluated result: 0
- *string.Equals('batch', 'batch')* - evaluated result: 1 (true)

4.5 Operators

Every logical evaluated result (true, false) is converted to and treated as integer type.

Example:

- *True* – evaluated result: 1
- *False* – evaluated result: 0

4.5.1 Relational operators

Supported relational operators are (using standard operator notation):

- equal
- not equal
- less than
- greater than

Note When you compare text using the “less than” and “greater than” operators, alphabetical order is enforced.

Example:

EXPRESSION	EVALS TO
2 == 2	1
Date('1974-08-24') != DateTime.Today	1
2 < -5.0	0
'Test' >= 'test'	1
3 in {1, 2, 3, 4, 5}	1
'Abc' like '[A-Z]b*'	1
'Abc' like '?'	0
1 between {1, 5}	1
'efg' between {'abc', 'xyz'}	1
'xyz' is int	0
'5.00' matches '^-?\d+(\.\d{2})?$'	1

4.5.2 Logical operators

Supported logical operators are:

- and
- or
- not

Example:

EXPRESSION	EVALS TO
true and false	0
!false	1
Image.Statistics.IsHotSpotMarkerVisible or !Image.Statistics.IsColdSpotEnabled	1 / 0

4.5.3 Bitwise operators

Supported bitwise operators are:

- and
- or
- xor
- not

Note Logical and bitwise operators are the same. Their interpretation depends on if you pass in integer values or Boolean values.

Example:

EXPRESSION	EVALS TO
1 and 3	3
1 or 3	4
1 xor 3	2
!1	-2

4.5.4 Mathematical operators

Supported mathematical operators are:

- addition
- subtracion
- multiplication
- division
- modulus (%)
- exponential power (^)

The addition and subtraction operators can be used for numbers and dates, while multiplication and division operators can be used for numbers only. Standard operator precedence is enforced.

Example:

EXPRESSION	EVALS TO
1+1	2
Image.Width + Image.Height	544 (for example)
date('1974-08-24') + 5	29-Aug-74 12:00:00 AM
7%4	3
-2 * -3	6
-2^4	16
Image.Measurements.Items[0].Max - Image.Measurements.Items[0].Min	5.4345 (for example)

4.5.5 Ternary operators (If-Then-Else)

Use the ternary operator to perform “if-then-else” conditional logic inside the expression.

Example:

- *false ? 'trueExp' : 'falseExp'*

In this case, the Boolean false result in returning the string value 'trueExp'.

4.5.6 List projection and selection

List projection and selection are very powerful expression language features that allow you to transform the source list into another list by either projecting across its "columns", or selecting from its "rows". In other words, projection can be thought of as a column selector in a SQL SELECT statement, while selection would be comparable to the WHERE clause.

Example:

- *Image.Measurements.Items.![Name]*

The example shown above gets the list of measurements names. As you can see from the example, projection uses *![projectionExpression]* syntax and will return a new list of the same length as the original list but typically with the elements of a different type. On the other hand, a selection which uses *?[projectionExpression]* syntax will filter the list and return a new list containing a subset of the original element list. For example, selection would allow us to easily get a list of measurements where minimum value is greater than 15.

Example:

- *Image.Measurements.Items.?[Min.Value > 15]*

For selecting the first element of the list, use the *^[projExp]* expression, and for selecting the last element of the list use the *$[projExp*].

4.5.7 Collection processors and aggregators

Expressions also support several collection processors as well as a number of commonly used aggregators.

Processors:

- nonNull() - eliminates all null values from the collection.
- distinct() - remove duplicate items in the collection. It can also accept an optional Boolean argument that will determine whether null values should be included in the result. The default is false.
- sort() - sort elements in collection.
- convert(Type) - convert a collection of elements to a given Type.

- reverse() - returns the reverse order of elements in the list.

Aggregators:

- count() - obtain a number of items in a collection.
- sum() - calculate a total for the list of numeric values.
- average() - return average for the collection of numbers.
- min() - return the smallest item in the list.
- max() - return the largest item in the list.

The difference between processors and aggregators is that processors return a new or transformed collection, while aggregators return a single value. Other than that, they are very similar; both processors and aggregators are invoke on a collection node using standard method invocation expression syntax, which makes them simple to use and allows easy chaining of multiple processors.

5 Conditional formula

Conditional formulas are very similar to ternary operators, but instead of performing conditional logic in one expression it is divided into three sections. Every section is an expression evaluator.

1. Expression – defines which condition will be displayed.
2. True condition – if the evaluated result from 'Expression' is equal to 1(*true*) then 'True condition' expression will be processed and displayed.
3. False condition – for every result from 'Expression' that is not equal 1, 'False condition' expression will be processed and displayed.

6 Dictionary

PropertyName (its type)

For example, to access camera model we use *Image.CameraInformation.Model*, and the type of this property is text.

6.1 Image

6.1.1 Properties

1. Width (integer)
2. Height (integer)
3. Description (text)
4. IsFile (bool)
5. IsThermal (bool)
6. TextAnnotations[Key]
7. Meterlinks[Key] (Key is text type)
    - Label (text)
    - Name (text)
    - OutputUnit (text)
    - Precision (integer)
    - ScaleFactor (text)
    - Value (text)
8. Meterlinks (Meterlink collection)

9. Fusion
    - PanX (integer)
    - PanY (integer)
    - Rotation (double)
    - PictureInPictureSettings
        - Name (text)
        - Blending (double)
        - Color (color mode/text)
        - UseBlending (bool)
    - MsxSettings
        - Name (text)
        - Alpha (double)
    - FusionMode
        - Name (text)
    - ThermalFusionAboveSettings
        - Threshold (float)
        - Color (text)
        - Name (text)
    - ThermalFusionBelowSettings
        - Threshold (float)
        - Color (text)
        - Name (text)
    - ThermalFusionIntervalSettings
        - Min (float)
        - Max (float)
        - Color (text)
        - Name (text)
    - VisualSettings
        - Color (text)
        - Name (text)
    - BlendingSettings
        - Color (text)
        - Level (float)
        - Name (text)
    - ThermalOnlySettings
        - Name (text)
10. Scale
    - Level (float)
    - Min (float)
    - Max (float)
    - IsAutoAdjustEnabled (bool)

11. Isotherms[indexer]

    - Appearance (text)
    - Color (Color)
    - ContrastColor (text)
    - Type (IsothermType enumeration)
    - Percentage (float)

    Above/Below isotherm:

    - Threshold (float)

    Interval isotherm:

    - Min (float)
    - Max (float)

12. Isotherms (collection)
13. TemperatureUnit (TemperatureUnit enumeration)
14. DistanceUnit (DistanceUnit enumeration)
15. Palette

    - Colors (Color collection)
    - AboveSpanColor (Color)
    - BelowSpanColor (Color)
    - OverflowColor (Color)
    - UnderflowColor (Color)
    - Isotherm1 (Color)
    - Isotherm2 (Color)
    - Measurement (Color)
    - Version (text)
    - Name (text)
    - Stretch (integer)
    - Method (integer)

16. CameraInformation

    - Filter (text)
    - Lens (text)
    - Model (text)
    - SerialNumber (text)
    - RangeMax (float)
    - RangeMin (float)
    - FieldOfView (float)

17. Histogram[indexer] (float)
18. Histogram

    - Items (float collection)
    - Count (integer)
    - Overflow (float)
    - SampleCount (float)
    - IsScaleTypeAuto (bool)
    - ScaleMax (float)
    - ScaleMin (float)
    - Underflow (float)

19. Parameters
    - AtmosphericTemperature (float)
    - Distance (float)
    - Emissivity (float)
    - ExternalOpticsTemperature (float)
    - ExternalOpticsTransmission (float)
    - ReferenceTemperature (float)
    - ReflectedTemperature (float)
    - RelativeHumidity (float)
    - Transmission (float)
20. ZoomSettings
    - Factor (float)
    - PanX (float)
    - PanY (float)
21. ColorDistribution (ColorDistribution enumeration)
22. CompassData
    - Degrees (float)
    - Pitch (float)
    - Roll (float)
23. GpsData
    - Altitude (float)
    - AltitudeRef (integer)
    - Dop (float)
    - IsValid (bool)
    - Latitude (float)
    - Longitude (float)
    - MapDatum (text)
    - Satellites (text)
    - Timestamp (integer)
24. Statistics
    - Min (Thermal)
    - Max (Thermal)
    - Average (Thermal)
    - HotSpot (Point)
    - ColdSpot (Point)
    - IsHotSpotMarkerVisible (bool)
    - IsHotSpotEnabled (bool)
    - IsAverageEnabled (bool)

25. Measurements['Key'] or Measurements.Items[indexer]- where Key is measurement name and indexer is integer value.
    All measurements have:

    - Name
    - Distance
    - Emissivity
    - ReflectedTemperature
    - IsCustomParametersEnabled

    Spot measurements also have:

    - X (integer)
    - Y (integer)
    - Value (Thermal)

    Line measurements also have:

    - Start (Point)
    - End (Point)
    - Statistics (see bullet **24** above)

    Rectangles and ellipses also have:

    - Location (Point)
    - Width
    - Height
    - ObjectArea
    - Coverage
        - PercentageAbove
        - PercentageBelow
        - PercentageInterval
        - ThresholdAbove
        - ThresholdBelow
        - RangeMax
        - RangeMin
    - Statistics (see bullet **24** above)

    Delta measurements also have

    - ValueMember1
    - ValueMember2
    - Measurement1 (text)
    - Measurement2 (text)
    - Value (Thermal)

    Polygon measurements also have

    - Points (Collection of Points)
    - Coverage (see above)
    - ObjectArea (float)

26. Measurements

    - Items (Collection) - Contains all measurements in image
    - MeasurementEllipses (Collection)
    - MeasurementLines (Collection)
    - MeasurementRectangles (Collection)
    - MeasurementSpots (Collection)
    - MeasurementDeltas (Collection)
    - MeasurementPolygon (Collection)

6.2 Page

6.2.1 Properties

1. Number (integer)
2. PageItems[indexer]
    - BorderThickness (integer)
    - BackgroundColor (CommonColor)
    - BorderColor (CommonColor)
    - Parent (Page Item)
    - Children (page items collection)
    - Height (integer)
    - Width (integer)
    - IsSelected (bool)
    - LastChange (date)
    - Location (Point)
    - Page (Page)
    - IsVisible (bool)
    - IsLocked (bool)
3. PageItems (page items collection)
4. BackgroundColor (CommonColor)
5. BorderColor (CommonColor)
6. BorderThickness (integer)
7. Width (integer)
8. Height (integer)
9. LastChange (date)
10. TopMargin (integer)
11. BottomMargin (integer)
12. LeftMargin (integer)
13. RightMargin (integer)
14. Doc
    - IsTemplate (bool)
    - IsNew (bool)
    - CreatedDateUtc (date)
    - ModifiedDateUtc (date)
    - FilePath (text)
    - Dpi (float)
    - Pages (Page collection)
    - Title (text)
    - Created (date)
    - Summary
        - Doc
        - ImagesDict (Image dictionary)
        - Images (Image list)
        - LastUpdated (date)

6.2.2 Types and their properties

1. Thermal
    - Value (float)
    - State (ThermalState enumeration)
2. Point
    - X (int)
    - Y (int)
    - IsEmpty (bool)

3. CommonColor
    - R - red(int)
    - G - green(int)
    - B - blue(int)
    - A - alpha(int)
    - Argb
4. Enumerations - is a data type consisting of a set of named values from the collection
    - IsothermType
        - Above
        - Below
        - Interval
        - InvertedInterval
    - ColorDistribution
        - TemperatureLinear
        - HistogramEqualization
        - SignalLinear
        - DigitalDetailEnhancement
    - TemperatureUnit
        - Celsius
        - Fahrenheit
        - Kelvin
        - Signal
    - DistanceUnit
        - Meter
        - Feet
    - ThermalState
        - Invalid
        - Ok
        - Overflow
        - Underflow
        - Warning

6.3 Static methods and properties

6.3.1 Methods

Note For proper calculation, insert round brackets on both sides, for example (Equals (2.53, 2.54, 3)).

- GetColor(R, G, B)
    - Returns CommonColor
    - Parameters R(red), G(green), B(blue) are integer type.
- Equals(value1, value2, precision)
    - Compares two floating point values using precision entered by user. Returns '1' if provided values are equal and '0' if not.
    - Parameters:
        - value1 and value2 are floating point type which are compared
        - precision is integer type
        - Equals(value1, value2)
    - Compares two floating point values with precision equal 2. Returns '1' if provided values are equal and '0' if not.
- Abs(value)

- Returns the absolute value of a specified number
- value is floating point type

- Cos(value)
  - Returns the cosine of the specified angle
  - value is floating point type
- Sin(value)
  - Returns the sine of the specified angle.
  - value is floating point type
- Tan(value)
  - Returns the tangent of the specified angle.
  - value is floating point type
- Round(value, precision)
- Rounds a floating point value to a specified number(precision) of fractional digits.
- Exp(value)
  - Returns **e** raised to the specified(value) power.
  - Parameter: value is floating point type
- Pow(value, power)
  - Returns a specified number(value) raised to the specified power.
  - Parameters: value and power are floating point type.
- Log(value, base)
  - Returns the logarithm of a specified number in a specified base.
  - Parameters: value and base are floating point type.
- Log10(value)
  - Returns the base 10 logarithm of a specified number.
  - Parameter: value is floating point type.
- Lg(value), same as Log10(value)
- Ln(value)
  - Returns the natural (base e) logarithm of a specified number.
  - Parameter: value is floating point type.
- Sqrt(value)
  - Returns the square root of a specified number.
- Num(text)
  - Converts the given text to a decimal number.
- Rad(degrees)
  - Converts degrees to radians
- Format(text, obj1, obj2, obj2, … obj n)
  - Formats text (allows text and numbers concatenation).
  - Formula example:
    ```
    Format('Max: {0:0.000} °C, Min: {1:0.00} °C, Average: {2:0.0} °C', Max, Min, Average))
    ```
  - The result (temperatures are just an example):
    Max: 77,268 °C, Min: 28,53 °C, Average: 34,5 °C
  - Please note the number of decimals.
  - For a simple case (one value) just use:
    ```
    Format('Max: {0:0.000} °C', Max))
    ```

6.3.2 Properties

Property	Equivalent to
Measurements	Image.Measurements
TextAnnotations	Image.TextAnnotations (dictionary<text,text>)
Meterlinks	Image.Meterlinks
Statistics	Image.Statistics
Stats	Image.Statistics
Note	Image.Description
Average	Image.Statistics.Average.Value
Avg	Image.Statistics.Average.Value
Maximum	Image.Statistics.Max.Value
Max	Image.Statistics.Max.Value
Minimum	Image.Statistics.Min.Value
Min	Image.Statistics.Min.Value
E	mathematical constant (2.7182...)
PI	mathematical constant (3.14159...)

6.4 Custom access to objects

To access a **measurement** object (and its properties) from the thermal image, use the measurement name.

Example:

- *Li1* – 'Li1' is the name of a line in the thermal image
- *Li1.Max.Value* - is equal to Image.Measurements['Li1'].Max.Value

To access an **isotherm** collection (and its properties) from the thermal image, use isotherm Type property. If specified isotherm Type is not set then provided text will be treated as a literal expression.

Example:

- *Below* – returns Isotherm object from the thermal image.
- *Below.Coverage*
- *Below.Threshold*

TextAnnotations – Value text is available by using this Key.

Meterlinks – Access Meterlink object by using this Label property.

Website
http://www.flir.com

Customer support
http://support.flir.com

Disclaimer
Specifications subject to change without further notice. Models and accessories subject to regional market considerations. License procedures may apply. Products described herein may be subject to US Export Regulations. Please refer to exportquestions@flir.com with any questions.

Publ. No.: T810519
Release: AA
Commit: 63916
Head: 63916
Language: en-US
Modified: 2020-02-18
Formatted: 2020-02-18