Using Corel Query Language (CQL) to Search for Objects in CorelDRAW and Corel DESIGNER Documents

Introduction

Searching for objects with certain properties is a common task while writing macros for CorelDRAW and Corel DESIGNER. Normally, if you need to search for objects of specific size or fill, you need to loop through all the shapes on the page, inspect each shape's properties and determine if it meets the criteria or not.

For example, the macro to delete all objects with no fill and no outline might look like:

```
Sub DeleteInvisible()
    Dim s As Shape
    Dim sr As New ShapeRange
    For Each s In ActivePage.Shapes.FindShapes()
        If s.Outline.Type = cdrNoOutline And s.Fill.Type = cdrNoFill Then
            sr.Add s
        End If
    Next s
    sr.Delete
End Sub
```

For more complex conditions, more checks are required. Note that these checks need to be done for each shape we find on a page.

You may also use a special query expression in the **Shapes.FindShapes** method that allows you to filter the shapes returned by that method. The expression provided uses CQL to specify the set of conditions a shape must meet in order to be included in the search results. Thus, to do the same task of removing objects with no fills and no outlines using this new approach, we can rewrite the above macro like this:

```
Sub DeleteInvisible()
    Dim sr As ShapeRange
    Set sr = ActivePage.Shapes.FindShapes( _
              Query:="@outline.type = 'none' and @fill.type = 'none'")
    sr.Delete
End Sub
```

Conditions that can be used in the **Query** parameter of **FindShapes** can be much more complicated than that. For example, the following macro selects all objects that have a red color anywhere in their fills or outlines:

```
Sub SelectShapesWithRedColor()
    ActivePage.Shapes.FindShapes(Query:="@colors.find('red')").CreateSelection
End Sub
```

CQL Data Types

CQL is an object-oriented query language. This mean that all the data it operates on are objects of some sorts and the objects have certain methods that can be invoked to access properties of those objects. There are a number of predefined data objects which represent numbers, strings, etc. and custom objects which are specialized for the specific task CQL is used for. Since CQL can potentially be used not only for finding shapes in documents, but other things as well (for example, it can be used to look for text characters with certain attributes such as font face, size and color).

The common built-in data objects of CQL include the following. These objects are present in any implementation of CQL:

- **null:** a null object representing a non-existent property of an object.
- **bool:** a Boolean value (either *true* or *false*).
- **int:** an integer value (between -2147483648 ... 2147483647).
- **double:** a floating point value (approximately between 10^{-308} ... 10^{308}
- **string:** a Unicode text string.
- **unit:** a unit of measurement. Currently linear units of measurements such as inches (in), millimeters (mm) and some others can be used.
- **com:** a wrapper to a COM object. This allows to call VBA object model methods for objects such as shapes, fills, outlines, colors, etc. CQL implementation of CorelDRAW and Corel DESIGNER objects don't have all the properties that are available through the object model, so this object can be used to call the object model for additional information about the objects.
- **array:** a list of objects of any CQL type.
- **custom:** any other object type supplied by the particular implementation of CQL. CQL specialization for finding shapes in documents supplies custom objects such as **shape**, **fill**, **outline** and **color**.

CQL Syntax

Each CQL expression is evaluated to produce a single value (object). When looking for a shape in a document, the query expression is evaluated for every shape in the document and the resulting value is converted to a Boolean (true or false) to determine whether the given shape matches the criteria. If the result is 'true', the shape is added to the list of found shapes, otherwise it is ignored.

However, in general, CQL expressions can evaluate to any object type, such as an integer number, a string, an array of objects, etc. Here are a few examples of valid CQL expressions:

- 1 + 2
- 2 * (2.3 + 1.2)
- 'Hello' + 'world'

CorelDRAW and Corel DESIGNER's VBA object model has a few helper methods which allow you to evaluate CQL expressions and trouble-shoot queries. **Application.Evaluate** is one of them; it evaluates a CQL expression and returns the result of the evaluation.

```
Sub TestEvaluate()
    MsgBox Evaluate("2*(2.3+1.2)")
End Sub
```

The above macro will just show a message box containing "7", which is the result of the arithmetic expression. This method is useful to quickly check the validity of a CQL expression or confirm the result value.

Each operand in the expression is an object, even the constants such as 1 or 'Hello'. For constants, the data type is determined automatically. Each object can have its own set of methods and properties. To call a method of an object, the following syntax is used:

- `object.method(parameters)`

Methods that do not require any parameters could be called without using the parentheses. Both of the following expressions are valid:

- `'world'.length()`
- `'world'.length`

Both of the above expressions evaluate to a numeric value of 5, the length of the string 'world'.

Operators

The table below lists the supported operators:

Operator	Data type	Action	Example	Result
+	numeric	addition	1 + 2	3
+	string	string concatenation	'Hello' + ' world'	'Hello world'
-	numeric	subtraction	10 - 4	6

*	numeric	multiplication	2 * 5	10
/	numeric	division	3 / 2	1.5
\	numeric	integer division	3 \ 2	1
^	numeric	power	2 ^ 3	8
>	any	greater than	3 > 2	true
<	any	less than	10 &lt 3	false
>=	any	greater than or equal to	2 >= 2	true
<=	any	less than or equal to	3 <= 10	true
=, ==	any	equal to	2 = 3	false
<>	any	not equal to	2 <> 3	true
=	string	case insensitive equality	'Test' = 'test'	true
==	string	case sensitive equality	'Test' == 'test'	false
<>	string	case insensitive inequality	'Test' <> 'test'	false
&gt, >=, =, <=, &lt	string	case sensitive compare	'Test' >= 'test'	false
&, and	bool	logical AND	2 = 2 and 1 = 1	true
|, or	bool	logical OR	2 = 1 or 1 = 1	true
!, not	bool	logical NOT	not (2 = 2)	false
.	object	object method call	(-1).abs()	1

Operator precedence

As in most programming languages, operators have certain precedence. Operators with higher precedence are evaluated first. Here is the list of operators in the order of their precedence (in descending order of precedence):

- . (object method call)
- ^
- *, /, \
- +, -
- >, <, =, ==, <=, >=, <>
- not, !
- and, or, &, |

Therefore, in the following expression

- 2 * 2 + 3

the multiplication is performed first, and then addition, since multiplication has a higher precedence.

In order to modify the precedence, parentheses () can be used:

- 2 * (2 + 3)

global Object

CQL generally works with objects. All functions are methods of objects. There is one special object which has special meaning - the **global** object. Methods of this object are called directly without specifying the object name. This object provides a set of common functions and constants such as **pi**, **e**, **sin**, **cos**, **min**, **max**, etc.

- "sin(2)" will evaluate to 0.909297426825682 (sine of 2 radians)
- "max(1, 2, 4, 3, 0)" will evaluate to 4 (maximum of the five numbers)
- "pi" will evaluate to 3.14159265358979

A more comprehensive list can be found under the **global** object reference.

Custom Objects

CQL can potentially be used for queries executed on any types of data. Finding shapes in a CorelDRAW and Corel DESIGNER document is just one application. Finding text attributes is another. In the future, new applications of CQL can be implemented (such as searching for files of specific attributes, or email contacts). In each application of CQL, different custom objects are provided that allow you to access the object-specific data. For example, when searching of shapes, the 'shape' object is provided to

CQL that allows you to inspect a shape's width or height, fill and outline properties, name and many other attributes. When searching for text within a text range, a 'text' object is exposed with properties such as font, size, color, style, case, and many more.

To access the methods and properties of the current object in a query, prepend its method name with the '@' character. Thus, when searching for shapes, "@width" would call the 'width' method of the current 'shape' object and return its width.

The methods of the current query object could return values of any of the built-in types or any other custom object type. For example, if the expression is evaluated on a shape and its "fill" property is referenced, the returned object is another type of custom object which provides the fill properties of the shape, so you can use it in a query as follows:

- "@fill.color.name = 'Red'"

Self documentation

CQL supports a special '**help**' method that lists all available methods and properties of any object. For example, to get a list of methods available on the 'int' object, the following expression can be evaluated: "int.help".

It is easiest to use the **Application.Evaluate** method in the Immediate window of VBA editor to get the help on object methods. Go to the **Immediate** window in VBA editor (Ctrl-G) and type:

- `? Evaluate("int.help")`

And hit the **Enter** key. The result would be printed in the window:

```
int.Abs()
int.ToRadians()
int.ToDegrees()
int.Radix(int, [int])
int.ToBool()
int.ToInt()
int.ToDouble()
int.ToString()
int.TypeName() {Static}
int.This()
int.IsNumeric()
int.IsNull()
```

Object Access Block

In many cases, various properties of the same object need to be inspected in an expression. To reduce expression repetition, a special expression can be used to reference methods of the same class repeatedly. A special construct with square brackets [] can be used for this. General syntax of it is the following:

- `"object[.method1 + .method2 * .method3]"`

For example, if you need to inspect several properties of a shape's fill, the following expression can be used:

- `"@fill[.type = 'uniform' and .color = 'red']"`

The above expression is equivalent to the following:

- `"@fill.type = 'uniform' and @fill.color = 'red'`

Here is another example:

- `"'Test'[.toUpper + .toLower]" => returns "TESTtest"`

Handling Units of Measurement

CQL has a built-in support for units of measurement. Linear dimensions such as mm, in, m, cm, pt, etc are handled in a special way. To provide a constant unit of measurement, it must be included in curly braces {} along with the unit value and unit of measurement. For example:

- {1 in} = 1 inch
- {1 mm} = 1 millimeter

Certain math operations can be performed on units of measurement:

- "{1 cm} + {1 mm}" => {1.1 cm}
- "{1 pt} * 3" => {3 pt}

Unit Power

CQL also supports multiplication and division of units of measurement. So, squares, cubes, etc of units are supported as well. Units of different degree can be specified using the following general syntax:

- {value unit^power}

For example:

- {2 mm^2} => 2 sq. mm
- {3 in^3} => 3 cubic inches

Units of power zero ({2 mm^0}) evaluate to regular numeric values, that is, "{2 mm^0}" equals to a numeric value of 2.

Powers of 2 and 3 can be specified by using special characters "²" (U+00B2) and "³" (U+00B3) after the unit of measurement:

- {2 mm²} => 2 sq. mm
- {3 in³} => 3 cubic inches

Multiplying units increases the power of the unit. Dividing them decreases the power:

- "{2 mm} * {2 mm}" => {4 mm²}
- "{4 mm²} / {1 mm}" => {4 mm}

Comparing units can be done only with the units of the same power:

- "{2 mm} * {2 mm} = {4 mm²}" => true (correct)
- "{2 mm} * {2 mm} = {4 mm}" => error: comparing mm² to mm

However different units of the same unit category can be used in expressions:

- "{1 in} = {72 pt}" => true

Comparisons of units of measurement are performed with reduced precision to eliminate rounding error. The precision of linear units of measurement in CQL is normally set to 3 decimal places of the largest unit involved in comparison. For precise comparison, the precise equality operator ("==") can be used. Considering that 1 mm = 2.83464566929134 pt, the following expressions will yield:

- "{1 mm} = {2.835 pt}" => true
- "{1 mm} == {2.835 pt}" => false

Converting units is done with respect to the unit power. To convert values between different units, the 'unit.convert' method must be used:

- "{1 in}.convert('mm')" => {25.4 mm}
- "{1 in²}.convert('mm')" => {645.16 mm²}

The unit object has the following built-in methods:

Method	Example	Result
unit.Category()	{1 in^3}.category	'Linear Dimensions'
unit.Power()	{1 in^3}.power	3
unit.Unit()	{1 in^3}.unit	'in'
unit.DisplayUnit()	{1 in^3}.displayUnit	'in^3'
unit.Value()	{1 in^3}.value	1
unit.BaseValue()	{1 in}.baseValue	254000
unit.Convert(string)	{1 in^3}.convert('mm')	16387.064 mm^3

CQL Class Reference

Base class for most CQL objects

Most of CQL built-in classes derive from a base class that provides some common functionality and methods. All of the objects described below also include the methods of the base class (with the only exception of the 'global' object that doesn't derive from the this base class).

object.ToBool()

Converts the current object to a Boolean value (true or false).

object.ToInt()

Converts the value of the object to Integer

object.ToDouble()

Converts the value of the object to Double

object.ToString()

Converts the value of the object to String

object.TypeName()

Gets the object type name as a string.

object.This()

Returns a copy of the object.

object.IsNumeric()

Returns true if the object can be converted to a numeric value (int or double)

object.IsNull()

Returns true if the object is the null object

null Object

The **null** object doesn't have any additional methods and is used to represent a missing property or object. It is legal to call any method on a null object and the call will succeed and the return value will be null. Thus, **null.somemethod()** will return 'null'.

Comparison operators =, ==, <, >, >=, <= involving a null object will always return false. So, 'null = 5' will be false. The only exception is comparing two null objects with =, ==. In which case the equality operators will return true. The inequality operator <> will always return true when comparing a null object to non-null unless both operands are null.

Example	Result
null = 3	false
null = 'test'	false
null = null	true
null <> 3	true
null <> null	false
3 > null	false
null > null	false
null >= null	false

bool Object

The **bool** object has no additional methods to those inherited from the base class. A Boolean can have a value of 'true' or 'false' and is the default return value of comparison operators (<, >, <>, =, etc).

int Object

int.Abs()

Returns an absolute value of the integer:
(-2).abs => 2

int.ToRadians()

Converts the value of degrees to radians:
180.ToRadians => 3.14159265358979

int.ToDegrees()

Converts the value of radians to degrees:
1.ToDegrees => 57.2957795130823

int.Radix(int, [int])

Converts a number to a different radix (base). The result of the method is a string. The first parameter specifies the new radix of the number and the optional second parameter specifies the size of the length of the string returned. If the actual string is smaller, the number is prepended with zero characters ('0'):
28.radix(16) => '1c'
28.radix(16,4) => '001c'
28.radix(2) => '11100'

double Object

The Double object represents a floating point value. It has the same additional methods as the int object above:

double.Abs()

Returns an absolute value of the double:
(-2.5).abs => 2.5

double.ToRadians()

Converts the value of degrees to radians:
(12.5).ToRadians => 0.218166156499291

double.ToDegrees()

Converts the value of radians to degrees:
pi.ToDegrees => 180.0

int.Radix(int, [int])

Converts a number to a different radix (base). The floating point value is converted to an integer first before the radix is changed.

string Object

The String object represents a Unicode string value which could be empty or contain one of more characters

string.Length()

Returns the length of the string:
'test'.length => 4

string.Empty()

Returns true if the string is empty:
'test'.empty => false

string.ToUpper()

Converts the string to uppercase:
'test'.ToUpper => 'TEST'

string.ToLower()

Converts the string to lowercase:
'TEST'.ToLower => 'test'

string.ToTitle()

Converts the string to Title case:
'TEST'.ToTitle => 'Test'

string.CharCode()

Gets the Unicode character code of the first string character:
'Test'.CharCode => 84

string.SubStr(int, [int])

Gets a substring of a string. The first parameter is a zero-based index of the first character to include in the substring. The second optional parameter is the length of the substring to retrieve. If it is omitted, the whole remainder of the string is returned:
'Hello World'.SubStr(6) => 'World'
'Hello World'.SubStr(6,3) => 'Wor'
The first index can be negative which has a meaning of the index from the end of the string:
'This is a test'.SubStr(-4) => 'test'
If the last parameter is negative, it indicates the last character from the end of the string:
'Some words'.SubStr(5, -1) => 'word'

string.Delete(int, [int])

Removes the specified characters from the string. The meaning of the parameters is the same as those of SubStr function described above:
'abcdefg'.Delete(2, 3) => 'abfg'
'abcdefg'.Delete(-2) => 'abcdeg'

string.Trim([string])

Removes the specified characters from the beginning and end of the string. If the optional string parameter is omitted then a space (code 32) and tab (code 9) characters are removed:
' test '.trim() => 'test'
'aaaaatestbbbccc'.trim('abc') => 'test'

string.LTrim([string])

Similar to the Trim function but removes the characters only from the beginning of the string:
' test '.trim() => 'test '

string.RTrim([string])

Similar to the Trim function but removes the characters only from the end of the string:
' test '.trim() => ' test'

string.Repeat(int)

Repeats the string the specified number of times:
'test'.repeat(3) => 'testtesttest'

string.StartsWith(string)

Returns true if the string starts with the specified string:
'Apple'.StartsWith('A') => true

string.EndsWith(string)

Returns true if the string ends with the specified string:
'Cooking'.EndsWith('ing') => true

string.Contains(string)

Returns true if the string contains the specified string:
'Peaceful world'.Contains('Peace') => true

string.Replace(string, string)

Replaces all instances of the search substring (the first parameter) with the replacement string (the second parameter):
'green pen, blue pen'.Replace('pen', 'pencil') => 'green pencil, blue pencil'

string.Split(string)

Splits the string delimited with the specified characters into an array of strings:
'Design<->Implementation<->Testing'.Split('<->') => array('Design', 'Implementation', 'Testing')

string.CharAt(int)

Gets the character at the given index in the string. The index is zero-based:
'Test'.CharAt(1) => 'e'

string.CharCodeAt(int)

Gets the Unicode character code of the character at the given index. The index is zero-based:
'ABCD'.CharCodeAt(3) => 68

string.IndexOf(string)

Finds a substring and returns the character index of its first occurrence. If it is not found, -1 is returned
'this is a test'.IndexOf('is') => 2

string.LastIndexOf(string)

Finds a substring and returns the character index of its last occurrence. If it is not found, -1 is returned
'this is a test'.LastIndexOf('is') => 5

string.ToCharArray()

Converts the string into an array of characters:
'abc'.ToCharArray => array('a', 'b', 'c')

string.ParseInt([int])

Converts a string into an integer value. The optional parameter specifies the radix of the number to convert from. This method always succeeds and parses the string till the first character it cannot recognize as a valid digit in the given radix.
'1c'.ParseInt(16) => 28

array Object

An array is a list of objects of any type.

array.Count()

Returns the number of elements in the array:
array(1,2,3).count => 3

array.Empty()

Returns true if the array contains no elements:
array(1,2,3).empty => false
array().empty => true

array.Add(object)

Adds an object to the end of the array

array.Item(int)

Gets the object identified by the index. Index is 1-based:
array(1,3,4).item(2) => 3

array.Join(string)

Joins the array elements into a string using the specified delimiter string:
array(1,3,4).join(', ') => '1, 3, 4'

array.ForEach(object, expression, [object])

Evaluates the specified expression for each element of the array and returns the result of the evaluation. This is an iterative process. The result of evaluation of the previous iteration is passed to the current one. The expression being evaluated can use a special iterator object which has these member variables:
$item - the current array element
$index - the 1-based index of the current array element
$lasteval - the result of evaluation of the previous iteration
$data - additional data object passed in by the caller.
The first parameter specifies the initial value passed in through $lasteval variable to the first element iteration. The second parameter is the expression being evaluated for each array element and the last optional parameter is the additional data object passed in as $data variable. If this parameter is omitted, $data variable will contain a null object.
array(1,2,3,4).foreach(0, $lasteval + $item) => 10 (sums up all elements of the array)
array('this', 'is','an', 'example').foreach('Result:', $lasteval + ' ' + $item.ToTitle) => Result: This Is An Example

array.Filter(expression, [object])

Creates a subset of array elements by including only the elements for which the Boolean expression specified evaluates to true. The expression can use the same iterator object as in ForEach method to get the value of the current element and the custom data passed in as the second parameter.
array(1,2,3,4,5).filter($item > 3) => array(4, 5)

array.Convert(expression, [object])

Changes the value of each element of the array to the result of evaluation of the given expression.
array(1,2,3,4).convert($item * 2) => array(2, 4, 6, 8)

array.Find(object)

Finds an object in the array and returns true if it is found
array(1,2,3,4).find(3) => true

array.IndexOf(object)

Finds an object in the array and returns the 1-based index of the array element found, or 0 if not found:
array('apple', 'orange', 'pear').indexof('orange') => 2

array.First

Returns the first element of the array

array.Last

Returns the last element of the array

unit Object

The unit object represent a value with specific units of measurements attached. A unit object has the following methods (see the section on Units above for more details):

unit.Category()

Returns the string describing the unit category

unit.Power()

Returns the power of the unit

unit.Unit()

Returns the unit name as a string

unit.DisplayUnit()

Returns the unit name and possible unit power (e.g. 'mm', 'cm³', etc)

unit.Value()

Returns the value of the unit, in the current units of measurement

unit.BaseValue()

Returns the base value of the unit, in the base units (base units are chosen for the category and all the other units are internally represented in terms of these base units)

unit.Convert(string)

Converts the unit value to another unit of measurement of the same category

global Object

Global object methods are available directly and do not require any object qualification to call. So, specifying 'pi' just executes the 'pi' method of the global object

global.bool()

Returns the type definition of the 'bool' class. Type definition is a special instance of the class that doesn't have the data associated with it. Only static methods (such as TypeName) can be called on type definition objects

global.int()

Returns the type definition of the 'int' class.

global.double()

Returns the type definition of the 'double' class.

global.string()

Returns the type definition of the 'string' class.

global.pi()

Returns a double value of pi

global.e()

Returns a double value of natural logarithm base

global.min(numeric, ...)

Returns the minimum value of the parameters:
min(2,3,4,1,0,13,5) => 0

global.max(numeric, ...)

Returns the maximum value of the parameters:
max(2,3,4,1,0,13,5) => 13

global.sqrt(double)

Returns the square root of the number

global.sin(double)

Returns the sine of the number

global.cos(double)

Returns the cosine of the number

global.tan(double)

Returns the tangent of the number

global.atan(double)

Returns the arctangent of the number

global.exp(double)

Returns the exponent of the number (e^x)

global.log(double)

Returns the natural logarithm of the value

global.iif(bool, expression, expression)

Inline if function. The first parameter is a Boolean expression. If the expression evaluates to true, then iif returns the value of the seconds parameter, otherwise it returns the value of the third parameter:
iif('This is an apple'.contains('apple'), 'apple', 'orange') => 'apple'

global.array([object, ...])

Creates an array of objects specified as parameters:
array('a','b','c').convert($item.repeat($index)).join(',') => 'a,bb,ccc'

global.chr(int)

Returns a character by the specified Unicode character code:
chr(65) => 'A'

global.rnd([int])

Returns a random integer number between 0 and the value specified in the parameter. If the parameter is omitted, 2147483647 (0x7FFFFFF) is assumed.
rnd(100) => returns a random number between 0 and 100.

global.radix(int, int, [int])

Returns a string representation of an integer is the given radix. The first parameter specifies the integer value to be converted, the second is the target radix. The third (optional) parameter specifies the length of the returned string. The string is prefixed with '0':
radix(1023,8,6) => '001777'

global.rgb(int, int, int)

Returns an RGB color with specified components:
@fill.color = rgb(255,0,0)

global.cmyk(int, int, int, int)

Returns a CMYK color with specified components:
@colors.find(cmyk(0,0,0,100))

global.cmy(int, int, int)

Returns a CMY color with specified components:
@colors.find(cmy(0,0,0))

global.hsb(int, int, int)

Returns an HSB color with specified components:
@fill.color.hsb = hsb(90,100,100)

global.hls(int, int, int)

Returns an HLS color with specified components:
@colors.find(hls(0,0,0))

global.lab(double, int, int)

Returns a Lab color with specified components:
@outline.color.lab = lab(100,127,127)

global.yiq(int, int, int)

Returns a YIQ color with specified components:
@colors.find(yiq(0,0,0))

global.vba()

Returns a VBA object that allows to execute VBA functions. See Calling VBA Macros from CQL section below for more details

Shape Object

This is the object that represents a shape in CorelDRAW or Corel DESIGNER document. The shape object is available when using one of the following object model methods:

- Shapes.FindShapes
- Shapes.FindShape
- Shape.Evaluate

When these methods are executed, the current shape being inspected can be accessed by calling one of the following methods prefixed with '@' symbol. For example, to select all the rectangles wider than 2 inches execute the following VBA command:

```
ActivePage.Shapes.FindShapes( _
              Query := "@type = 'rectangle' and @width > {2 in}").CreateSelection
```

The shape object contains the following additional methods and properties in addition to those inherited from the base class:

shape.Name()

Returns the name of the current shape
@name='MyShape'

shape.Type()

Returns a string representing the type of the shape (the value returned will be 'rectangle', 'ellipse', 'curve', etc)
@type = 'polygon'

shape.Fill()

Returns the fill object representing the fill of the shape (see below)
@fill.type = 'uniform'

shape.Outline()

Returns the outline object representing the outline of the shape (see below)
@outline.color = 'red'

shape.Width()

Returns the width of the shape, in the current units of measurements
@width > {1 in}

shape.Height()

Returns the height of the shape, in the current units of measurements
@height = {20 mm}

shape.Left()

Returns the horizontal position of the left edge of the object on the page
@left = {0 mm}

shape.Right()

Returns the horizontal position of the right edge of the object on the page
@right > {8.5 in}

shape.Top()

Returns the vertical position of the top edge of the object on the page
@top = {11 in}

shape.Bottom()

Returns the vertical position of the bottom edge of the object on the page
@bottom < {4 in}

shape.CenterX()

Returns the horizontal position of the center of the object on the page
@centerX = {4.25 in}

shape.CenterY()

Returns the vertical position of the top center of the object on the page
@CenterY = {5.5 in}

shape.Colors()

Returns an array of colors used in the fill and the outline of the shape. This array contains only unique colors, so if both the fill and the outline have the same color, only one color will be returned
@colors.find(cmyk(0,0,0,100)) => finds if the shape contains a CMYK black color
!@color.filter($item.cmyk.k <> 0).empty => Finds if the shape has any color that has non-empty black color component.

shape.COM()

Returns the COM object for the shape. This allows to get access to methods and properties of VBA object model's Shape object that are not directly accessible by CQL:
(@com.transparency.type = 1) => check if the shape has uniform transparency applied to it. Please refer to CorelDRAW VBA object model to see more details on VBA objects and methods

Fill Object

The fill object represents the fill properties of a shape

fill.Type()

Returns the type of the fill as a string. Possible values are: 'none', 'uniform', 'fountain', 'postscript', 'pattern', 'texture', 'hatch'
@fill.type = 'fountain' and @fill.fountain.angle = 45 => selects all the objects with 45 degree fountain fills.

fill.Color()

Returns the color of the uniform fill, or null if the fill is not uniform.
@fill.color = 'blue' => Selects objects with blue uniform fill

fill.Fountain()

Returns the properties of a fountain fill, or null, if the fill is not fountain.

Outline Object

The outline object represents outline properties of a shape

outline.Type()

Returns the string representing the outline type. The possible values are: 'none', 'solid', 'dot-dash', 'enhanced'
@outline[.type = 'solid' or .type = 'dot-dash'] => selects shapes with solid and dashed outlines

outline.Color()

Returns the color of the outline or null if the shape doesn't have an outline

outline.Width()

Returns the width of the outline in points:
@outline > {1 pt}

outline.IsHairline()

Returns true if the outline is hairline

outline.ScaleWithObject()

Returns true if Scale Outline with Object option is set

outline.BehindFill()

Returns true if Behind Fill option is set

outline.LineCaps()

Returns the string representing the line cap type of the outline. Possible values are: 'butt', 'round', 'square'

outline.LineJoin()

Returns the corner joins in the outline, as a string value. Possible values are: 'miter', 'round', 'bevel'

Color Object

The color object represents color properties in CorelDRAW

color.Type()

Returns the color model string. This could be values like 'rgb', 'cmyk', 'hsb' and so on

color.Name()

Returns the name of the color, or 'unnamed color' if the color doesn't have a name

color.Palette()

Return the name of the palette the color is from, or empty string if the color is from a custom palette

color.SourcePalette()

Returns the original palette name for userink colors.

color.PaletteIndex()

Returns the color index in the palette. This method is available only for colors from fixed palettes

color.Tint()

Returns the tint value of a spot color

color.IsSpot()

Returns true if the color is a spot color

color.IsCMYK()

Return true if this is a CMYK color

color.IsGray()

Return true if the color contains only black component

color.IsWhite()

Returns true if the color is pure white

color.IsTintable()

Returns true if the color can have tints

color.COM()

Returns the VBA COM object for the color. This allows to access additional methods and properties of the Color object not directly exposed through CQL

color.RGB()

Converts the color to RGB and returns a special version of CQL color object that has three additional properties - R, G, B to get access to individual color components:
@fill.color.rgb.r = 255

color.CMYK()

Converts the color to CMYK and returns a special version of CQL color object that has four additional properties - C, M, Y, K to get access to individual color components:
@fill.color.cmyk[.c = .m]

color.CMY()

Converts the color to CMY and returns a special version of CQL color object that has three additional properties - C, M, Y to get access to individual color components:
@fill.color.cmy.c = @outline.color.cmy.c

color.Gray()

Converts the color to grayscale and returns the gray level value as an integer:
@fill.gray < 128

color.HSB()

Converts the color to HSB and returns a special version of CQL color object that has three additional properties - H, S, B to get access to individual color components:
@fill.color.hsb.h = 270

color.HLS()

Converts the color to HLS and returns a special version of CQL color object that has three additional properties - H, L, S to get access to individual color components:
@fill.color.hls.s = 0

color.Lab()

Converts the color to LAB and returns a special version of CQL color object that has three additional properties - L, a, b to get access to individual color components:
@fill.color.lab.l = 0 and @fill.color.lab.a = -10

color.YIQ()

Converts the color to YIQ and returns a special version of CQL color object that has three additional properties - Y, I, Q to get access to individual color components:
@fill.color.yiq.y < 10

Calling VBA Macros From CQL

It is possible to call VBA macro functions from CQL expressions. For this, the **vba** property of the **global** object is used which provides access public VBA macro functions by using the following syntax:

- vba.Project.Module.Function(parameters)

For example, you can add the following VBA function to **GlobalMacros > Module1**:

```
Public Function Sum(ByVal x As Double, ByVal y As Double) As Double
    Sum = x + y
End Function
```

Then call it from a query like:

```
MsgBox Application.Evaluate("vba.GlobalMacros.Module1.Sum(2, 3)")
```

which will yield 5.

Only **public** functions in code modules can be called this way. Private functions, functions in class or form modules as well as subroutines cannot be called from queries.
Related

- CQL Query Question!

  

  So I've finally decided to jump into the CQL Query pool and I'm amazed at it's speed. I like it!!! [:D]So I'm going through my code and simplifying it with CQL where ever it can be applied. I'm stuck...
- Search active document

  

  I have this code working txtFIND = "Order#" txtREPLACE = TextBox12 For Each d In Documents 'Loop all the open documents For Each p In d.Pages 'Loop each page p.TextReplace txtFIND, txtREPLACE, True...
- CQL Descripton

  

  Hi, I would like to write a macro which can identify a certain shape and replace it with another one. I guess the CQL provides a very nice way to find shapes, unfortunately I haven't found a...
- Accessing Shape.Properties using CQL?

  

  I would like to use CQL to find shapes based on a custom Property, but I haven't found a syntax that works. If I set the value for the shape like this: Shape.Properties("MyProperty", 1) = "foo" ...

Previewing Staged Changes