UT Light .NET UFL

User Function Library for Barcodes, Images, SQL, Totals of Totals, Memory, Excel, Text, Font, ini Files, Registry, Windows, Time Zones, Geo, Web, HTML, email, ZATCA, JSON, etc.

www.MilletSoftware.com

Version 6.4.9159
(October 2025)

By

Ido Millet

5275 Rome Court, Erie PA 16509
ido@MilletSoftware.com
(814) 825-6009

Notes:
The functions are prefixed with '`ufl`' (or '`fl`' for legacy functions).
To simplify the discussion, the user manual typically ignores that prefix.

Use the 64-bit version with Crystal Reports 2020 or any 64-bit runtime.
Use the 32-bit version with older versions of Crystal Designer or 32-bit Crystal runtime.
You can install and use both the 32-bit as well as the 64-bit versions on the same machine.

Introduction

CUT Light functions are used in Crystal Report formulas/expressions for use cases such as:

1. **E-mail** dynamic-content messages from within any section of a Crystal report via **SMTP** (used by e-mail clients such as Eudora, Outlook, Outlook Express, Netscape Messenger, Pegasus Mail, etc.).

   A variety of options are supported including multiple recipients, CC Recipients, BCC recipients, and attachment files.
   These options can be specified by using the **EmailSet()** function call within a Crystal formula and by appending more elements or more text via follow-up **EmailAdd()** function calls before triggering the email via an ***EmailSend*** function call.

2. **Aggregate/Lookup/Rank Totals** (totals of totals)

3. **Append Content to Text Files**
   Note: this can be used to take snapshots of information each time the report runs (for example, via Visual CUT scheduled processing). Another Crystal report can then use the text file as a data source for information across multiple snapshots.

4. **Read Content of Text/RTF/HTML Files**
   Provide the file path & name as an argument to the ***FileGetText()*** function and get the file content as a string. You can use Crystal's formatting options to interpret the string as RTF or HTML. You can also use Crystal's string search and manipulation functions to lookup values inside the text file.

5. **Check a File Exists (Local or Web)**

6. **Execute SQL statements against any ODBC data source**

7. **Lookup & Set Values in *.ini files**

8. **Lookup Values in the Registry**

9. **Replace Accented Characters with Regular Ones**

10. **Convert GMT/UTC to Local or Specified Time Zone**

11. **Compute Distance between Points** (by zip codes or by Lat/Long)

12. **Trigger another Application via a Command Line**

13. **Trigger Message & Input Boxes Based On Report Content**

14. **Embed Input from User in Command Line Calls**

15. **Trigger Report Processing by Visual CUT or DataLink Viewer**

16. **Convert HTML to RTF for Better Rendering in Crystal**

17. **Convert HEX strings to Values**

18. **Get the 'User Name' & 'PC Name' Running the Report**
    Note: this can be used to impose row-level security or to address data access tracking requirements such as those imposed by HIPAA.

19. **Encrypt/Decrypt Text**

20. **Call Web Services and Return Values**

21. **Resize or Crop Images**

22. **Generate barcodes without special fonts**

23. **Generate Bullet Charts, Sparklines, and advanced Gauges**

24. **Regular Expressions**

25. And more…

Install / Uninstall

Follow the instruction received via email.

After the installation, the formula editor within Crystal should show the new functions under Additional Functions, COM and .NET UFLs.

Windows & Miscellaneous

uflBatchFileRun()

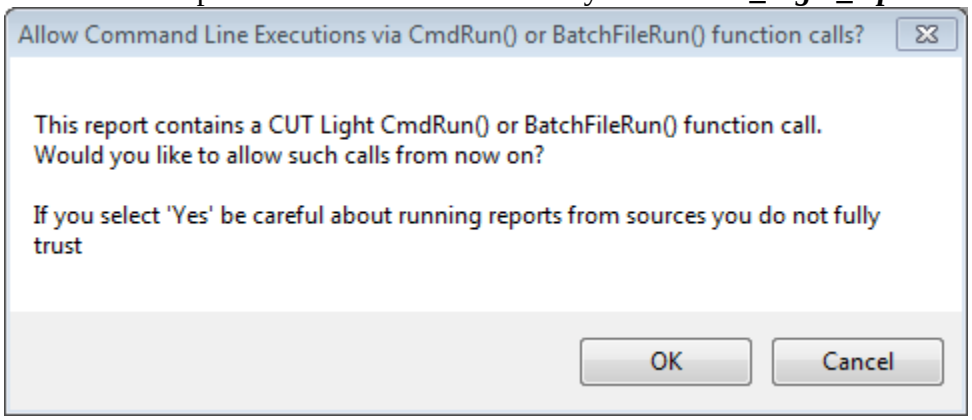
Arguments: (**BatchFilePath**, **Arguments**, **Visibility**)

This function triggers batch file passing to it optional arguments and controlling the visibility of the batch file window. If Visibility is set to "Hidden" the process window is hidden. Otherwise, it is visible.

Note: if there's a chance your arguments might contain special characters that interfere with passing them to the batch file (these include commas, spaces, <, >, ^, |) pass the arguments by enclosing them in single quotes within the double quotes like this:
`uFLBatchFileRun("C:\Batch\test.cmd", '"Jane Doe"', "Normal")`
Then, handle the remaining double-quotes on the receiving side.

Returns: 'Done' or error message.

Note: use of **CmdRun()** or **BatchFileRun()** requires a 1-time user permission to trigger such functions. The permission is stored as an entry in the ***CUT_Light_Options.ini*** file.



uflBitWiseAnd()

Arguments: (Integer1, Integer2)

Returns: The result (as Integer) of a BitWise AND operation on the two input integers.

For example: **BitWiseAnd(13, 6)**
Returns: **4**

uflClipboardSetText()

Arguments: (TextToSet)

Returns: The text specified as an argument.

For example: **ClipboardSetText** (ToText({Invoice.Invoice_N},0,")

uflCmdRun()

Arguments: (CommandLineArguments)
This function triggers Cmd.exe passing to it a command line.
Returns: 'Done' or error message.

See comments above **1st-time permission** and **special characters** in section about **BatchFileRun()** above.

This function is very similar to EXERun except that the EXE is Cmd.exe and you don't need to figure out the path to that executable and how to trigger it without leaving a command window open.

For example, the following Crystal formula:
```
uFLCmdRun("del c:\temp\*.tmp")
```
would delete all files with .tmp extensions in the temp folder.

uflCreateGUID()

Arguments: none

Returns: a GUID as string.

Example: CreateGuid()
returns: **{E0DDC73A-E7FA-484A-A640-4DC79315AA16}**

uflCultureInfoName()

Arguments: none
Returns: the name of the computer's culture (aka Locale).

For example, if the computer is locates in the USA, uflCultureInfoName() returns 'en-US'

ufl**EXERun()**

Arguments: (ExePath, CommandLineArguments)

This function triggers another application (EXE or Bat or Cmd file), passing to it a command line.

Returns: 'Done' if the executable was found in the specified path. Otherwise, returns an error message.

This function is very similar to VisualCutRun, except that it is free to call any application (not just Visual CUT). You can still use the ExeRun to call Visual CUT. For example, the following Crystal formula:

```
ExeRun ("C:\Program Files\Visual CUT\Visual CUT.exe", "-e
""C:\Program Files\Visual CUT\Visual_CUT.rpt""
""Parm1:1996""")
```

Triggers processing of the Visual_CUT.rpt sample report, overriding the saved parameter value with a value of 1996.

Note that each double (") quotes in the command line, must be duplicated ("") within the formula so it is recognized as such.

ufl**FormatValue()**

Arguments:

Value (Dbl): the number to convert/format as string. Use cDbl() if input type is not Double

FromType (string): only '**Millimeters**' or '**Inches**' are currently supported

ToType (string): only **'Imperial' is currently supported**

Denominator (integer): control precision (2, 4, **8**, 16, 32, or 64) of fractions such as 5' 11 3/**8**"

Separator (string): " " for no separator produces 5' 11 5/8" while "-" produces 5'-11 5/8".

Options (string): Leave as blank ("").

Returns: a string formatting the value or an error message starting with "*** "

For example, this formula expression:

```
uFLFormatValue (71.6, "Inches", "Imperial", 8, " ", "");
    returns: 5' 11 5/8"
and
uFLFormatValue (42, "Millimeters", "Imperial", 16, " ", "");
    returns:    1 11/16"
```

uflGetDiskSerialNumber()

Arguments: (FormatAsHex)

Returns: the serial number of the current disk drive.
If the FormatAsHex argument is set to True, the number is returned formatted as HEX.

Returns: a String displaying the serial number of the current disk drive
as a number (e.g. **-1808600113**) or as Hex (e.g. **9432F3CF**)

uflGetEnvironmentVar()

Arguments: (VariableName)

Returns: The environment variable value for the specified variable name.

Note: typical environment variable names include:

AppData
LOCALAPPDATA
PATH
ProgramFiles
CommonProgramFiles
SystemDrive
WinDir
UserDomain
UserProfile

For example, the following Crystal formula:
GetEnvironmentVar ("AppData")
returns a value such as:
C:\Users\ido\AppData\Roaming

uflGetMachineIPAddress()

No arguments.
Returns: a String with the IP Address of the machine where the Crystal report is running.

uflGetMachineName()

No arguments.
Returns: a String with the PC name where the Crystal report is running.

uflNormDist()

Arguments: (x, mean, std)

Returns: the cumulative probability for the value x under a normal distribution with the specified mean and standard deviation.

For example: NormDist(7, 5, 2) = 0.8413

uflGetRegisteredCompanyName()

No arguments.
Returns: a String with the Windows Registered Company Name.

uflGetUser()

No arguments.
Returns: a String with the ID of the user logged to the PC.

Note: among other things, this can be used to address data access tracking requirements such as those imposed by ***HIPAA*** (*Health Insurance Portability and Accountability Act*). By using ***GetUser()*** and ***FileAddText()*** you can log to a text file information about who accessed what patient information and on what date.

uflActiveDirectoryGetProperty()

Arguments: User_ID, Property_Name, Options (currently, use just "")

Returns:
The property value as a string or an error message starting with **"Error: "**

Example: uFLActiveDirectoryGetProperty(uFLGetUser, "mail", "")
returns: jane.doe@acme.com

ufl**NewKey()**

Arguments: (KeyString)

Returns:
True if the string is a new key (a NewKey() call hasn't been issued for it within the same report preview session). In which case the KeyString gets added to an internal set of keys that is reset only when a new report preview is triggered.
False, if the KeyString is already in the internal set of keys.

This function can be used to **avoid duplicate processing** and to compute **Distinct Sums**.

Avoiding Duplicate Processing (New Approach)

Crystal might evaluate the same formula multiple times, as it renders the page content (particularly when Keep Together properties cause shifting of page content from one page to another). To avoid duplicate processing of CUT Light function, you can use the NewKey() above to ensure the same process is not triggered twice. Here is an example:

```
WhilePrintingRecords;
IF ufl NewKey({Product_Type.Product Type Name}) Then
    FileAddText("c:\temp\testNewkey.txt",
         {Product_Type.Product Type Name}, False, True)
```

This formula writes to text file only if the NewKey() function confirms that the current Product Type Name hasn't been processed yet.

ufl**KeySetNewItem()**

Arguments (sKey)
Returns: "DUP" if the key is already in the key set (no duplicates allowed in the set).
"OK" if the key was added to the key set because it is not a duplicate
or error message if unexpected failure occurs.

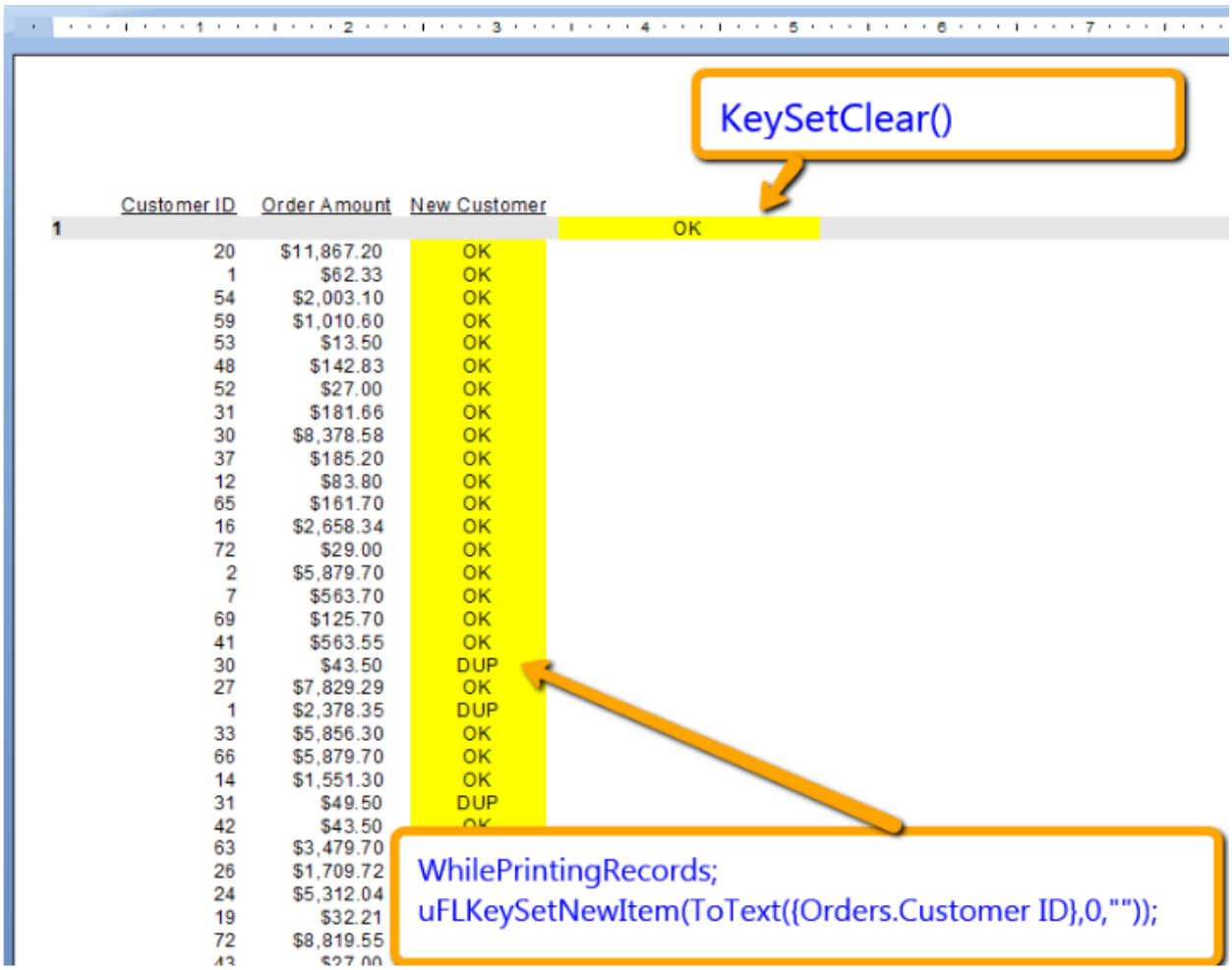
This function is similar to the ufl**NewKey**() function but it is faster, consumes less memory, and can be combined with ufl**KeySetClear()** to reset the key collection.

uflKeySetClear()

Arguments (no arguments)
Returns: "OK" or error message if unexpected failure occurs.

This function resets the unique key set populated by ufl**KeySetNewItem**() calls.

For example, this image below shows that in the Group Header (for Employee ID) we reset the key Set using ufl**KeySetClear()**. Then, in the detail section, we use ufl**KeySetNewItem**() to differentiate between encountering each customer id for the first time ("OK") as opposed to repeat encounters ("DUP").

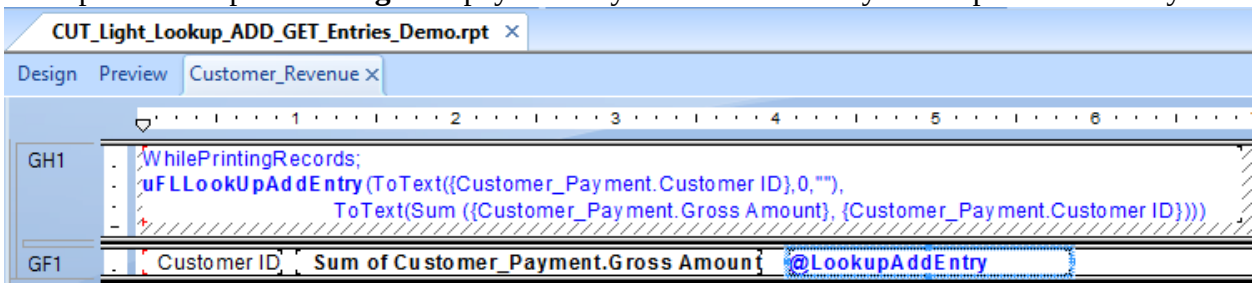


uflLookupAddEntry()

Arguments (sKey, sValue)
Returns: "OK" if the Key-Value pair was successfully added to memory.

Use ToText() in your formula to convert non-string Key or Value data
If the Key already exists, the entry gets replaced with the new Value.

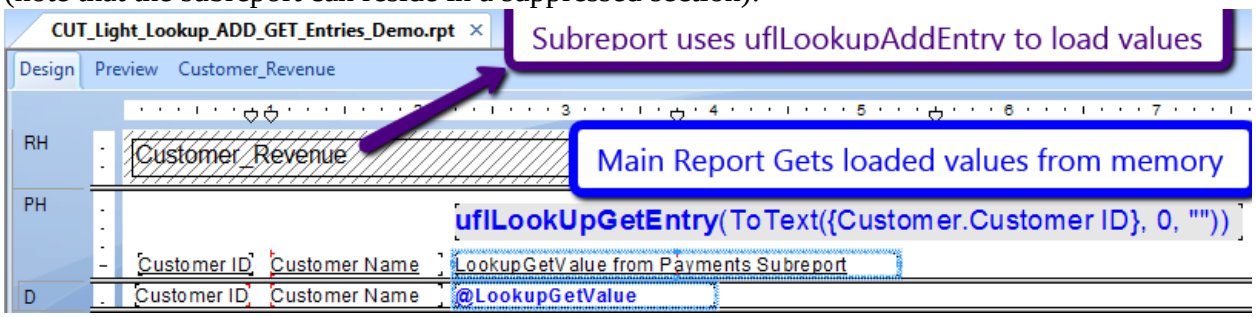
Example of subreport **loading** total payments by Customer into Key-Value pairs in memory:



uflLookupGetEntry()

Arguments (sKey)
Returns: The Value from matching Key-Value pair found in memory or "Entry Not Found".

Example of a main report **getting** values that were loaded by the subreport above
(note that the subreport can reside in a suppressed section):



Customer ID	Customer Name	LookupGetValue from Payments Subreport
65	Platou Sport	$59,950.02
66	Piccolo	$28,717.41
67	Paris Mountain Spo	$19,903.36
68	Magazzini	$49,991.57
69	Furia	$46,712.75
70	Folk och fä HB	$49,249.77
71	BBS Pty	$49,375.20
72	Cycle City Rome	$61,214.37
73	Tienda de Bicicleta	$71,957.07
74	Fahrkraft Räder	$57,350.60
75	Belgium Bike Co.	$48,280.09
76	Canal City Cycle	$67,742.95
77	Warsaw Sports, Inc	$39,973.33
78	Greenlane Bicycles	Entry Not Found
79	Helsinki Bicycle	$989.55
80	France Sports	$35.70

ufl**LookupResetEntries()**

Arguments: none
Returns: "OK" or error message

Used to reset all entries previously set via LookupAddEntry() calls.
This is useful in cases where a user runs multiple reports without closing the reporting software (Crystal, DataLink Viewer, …) between reports.

ufl**VisualCutRun()**

Arguments: (VisualCUTExePath, CommandLineArguments)

This function triggers processing of another report by Visual CUT. Visual CUT is a Crystal Report Manager package developed by Millet Software (www.MilletSoftware.com).

A typical scenario for using this functionality is running a report on Crystal Enterprise (or another software package), and **using the viewing of that report as a trigger mechanism for exporting, printing, and/or e-mailing of information in another report.**

Returns: 'Done' if the Visual CUT executable was found in the specified path. Otherwise, returns an error message.

For example, the following Crystal formula:

```
VisualCutRun ("C:\Program Files\Visual CUT\Visual CUT.exe", "-e ""C:\Program Files\Visual CUT\Visual_CUT.rpt"" ""Parm1:1996""")
```

Triggers processing of the Visual_CUT.rpt sample report, overriding the saved parameter value with a value of 1996.

Note that each double (") quotes in the command line, must be duplicated ("") within the formula so it is recognized as such.

ufl**Sleep()**

Arguments: (Milliseconds as Integer)

This function injects a delay of the specified number of milliseconds.
Returns: 'OK'

For example, the following Crystal formula tries to confirm access to a remote machine. If it fails, it tries up to 2 more times, injecting a 3-seconds delay between each try.

```
Local NumberVar Tries := 0 ;
Local StringVar Result := uFLPing ("74.125.67.100", "") ;
While result = "FALSE" AND Tries < 2 DO
(
   uFLSleep (3000) ;
   Result :=  uFLPing ("74.125.67.100", "") ;
    Tries := Tries + 1 ;
);
Result ;
```

Images

ufl**GetImageProperties()**

Arguments: (image file path & name)

Returns: Width/Height (in pixels) string.

Example: `GetImageProperties("c:\temp\MyLogo.jpg")`
returns: **90**/**111**

Notes:
- If image file in not found, the function returns "File Not Found"
- Supported image types are: JPEG, JPG, GIF, BMP and PNG
- If image type is not supported, the function returns "Image Type Not Supported"

uflImageResize()

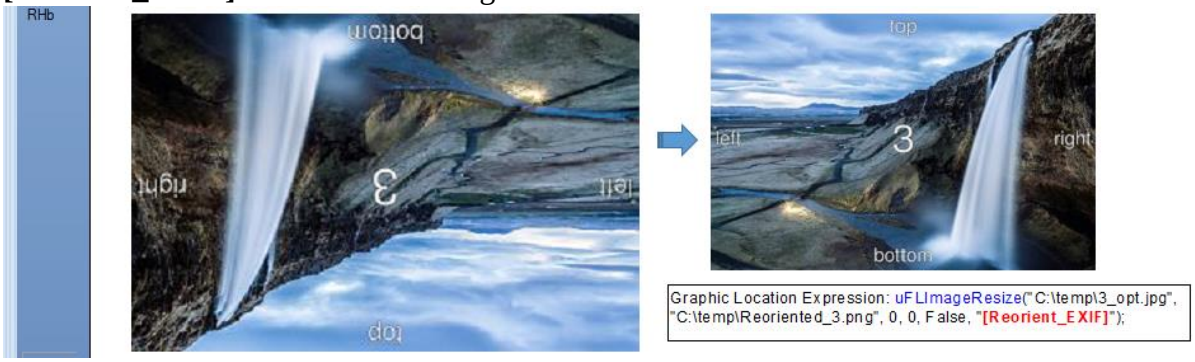
Arguments: (**url** or **image file path & name**, new image file path & name, Width, Height, MaintainProportion, Options)

Returns: The path to the new resized image file.

Example: uflImageResize("c:\Logo.jpg","c:\temp\Logo.png",180, 222, TRUE, "[AvoidAntiAliasing]")
returns: "c:\Temp\Logo.png" (the path the new resized image file

Notes:

- Width & Height in pixels. For no change in size, set to **0, 0**
- Image file path and name **can be a URL**. For example:
  uFLImageResize("http://acme/Glider.jpg", "c:\temp\Glider.jpg", 1600, 900, False, "")
- If ***MaintainProportion*** is set to TRUE, the image is resized to best fit within the specified dimensions.
- You can **convert on the fly between image formats** by specifying a different file extension. For example to convert from jpg to png:
  uflImageResize("c:\temp\A.**jpg**", "c:\temp\B.**png**", 180, 222, True, "")
  Supported Source Formats: BMP, JPEG, PNG, TIFF, GIF
  Supported Output Formats: BMP, JPEG, PNG
- Options: leave the Options argument blank ("") or include some of these options:
  - **[Center]** to center a proportionally resized image within the width & height using white as the fill color for the remaining vertical or horizontal margins. note: this stops Crystal from distorting the image when '***Can Grow***' is False.
  - **[AvoidAntiAliasing]** to avoid anti-aliasing
  - **[Reorient_EXIF]** to reorient the image based on embedded EXIF code:



uflImageReorient()

Arguments: (**url** or **image file path & name**, new image file path & name,
Reorient_Directive, Options)

Returns: The path to the new resized image file (or an error starting with ***)

Rotations are clockwise. Reorient_Directive can be one of these:
"Rotate90FlipNone", "Rotate180FlipNone", "Rotate270FlipNone"
"RotateNoneFlipX", "Rotate90FlipX", "Rotate180FlipX", "Rotate270FlipX"

Example:
uflImageReorient("https://www.milletsoftware.com/images/DataLinkViewer/Data_Viz2.jpg",
"c:\temp\DLV_Data_Vizualizer.png", "Rotate90FlipNone", "")

returns: " c:\temp\DLV_Data_Vizualizer.png " (the path the new resized image file

Notes:
- you can convert between image formats by specifying a different file extension.
Supported Source Formats are: BMP, JPEG, PNG, TIFF, GIF
Supported Output Formats are: BMP, JPEG, PNG

- Leave the Options argument blank ("")

uflImageFix()

This function is typically used to fix the orientation and noise in scanned images.

Arguments: (**url** or **image file path & name**, new image file path & name,
Reorient_Directive, **DeSkew**, **DeSpeckle**, Options)

Returns: 'OK' or an error starting with ***

DeSkew (True or False) automatically detects the rotation angle of the image and un-rotates it. It. It can handle rotation angels in the range of -40º-40º.

Reorient_Directive can be one of these (clockwise):
"Rotate90FlipNone", "Rotate180FlipNone", "Rotate270FlipNone"
"RotateNoneFlipX", "Rotate90FlipX", "Rotate180FlipX", "Rotate270FlipX"

DeSpeckle (True or False) should typically be left as False. It is an attempt to get rid of small groups of isolated pixels, assuming they are noise (e.g. scanning artifacts).

Leave options as blank ("")

Example: uflImageFix("c:\temp\scan.png", "c:\temp\scanFixed.png"
"Rotate90FlipNone, **True**, **False**, "")

Notes:

- For images that are rotated more extremely, use uflGetImageProperties() to compare height to width and apply a Reorient_Directive. The Reorient step occurs first, allowing the finer -40º-40º auto-DeSkew step to handle the rest.
- you can convert between image formats by specifying a different file extension.
  Supported Source Formats are: BMP, JPEG, PNG, TIFF, GIF
  Supported Output Formats are: BMP, JPEG, PNG

uflImageCrop()

Arguments: (**url** or **imagefile path & name**, new image file path & name,
TopLeftX, TopLeftY, Width, Height, Options)

Returns: The path to the new resized image file.
If error occurs, the path is to an image of the error text.

Example: to crop a bullet chart where the scale was hidden, the following call:
uFLImageCrop("c:\temp\Bullet_" & {Employee.Last Name} & ".bmp",
"c:\temp\Bullet_" & {Employee.Last Name} & "_Cropped.bmp", 20, 5, 310, 35, "");
converted this image

To this cropped image:

This allowed generating the report shown in this image.

Notes:

- To auto-crop an image (remove white margins):
  - set TopLeftX and TopLeftY to **-999**
  - set Width & Height to **-999** if after-crop content should be returned as is. Otherwise, the after-crop content would be resized.
- Auto-crop default to a threshold of 250 (range 0 to 255) below which a pixel is considered not white. To make the auto-cropping more aggressive, you can override the default value by including a threshold directive in Options. For example, **"[Threshold:110]"**
- TopLeftX and TopLeftY indicate the top-left corner where the cropped content should be copied from in pixels. Width & Height indicate the width and height of the cropped content to be copied, starting from the top left corner. In pixels. With 75 DPI resolution, 75 pixels = 1 inch.
- Image file path and name can be a URL. For example: "https://acme/Glider.jpg"
- You can convert on the fly between image formats by specifying a different file extension for the new image.
- Include **[AvoidAntiAliasing]** in Options text to avoid anti-aliasing

Crop to Circular Image

Include **[Circular:255,255,255]** in Options text to get a circular image framed in white.
Set the 3 numbers (RGB) to the desired frame color. Or including the Opacity element (0 to 255) before the RGB elements. For example, **[Circular:0,255,255,255]**

Circular QR Code Example

The sample report preview below uses *CUT Light* to
a) generate a QR barcode, and
b) crop it to a circular image with a light gray frame (to match the section background)
using the following Graphic Location expression:

```
uflImageCrop(
    uFLBBarcodeQR({Product_Type.Product Type Name}, "", False, "L", 112, 112, 0,
                  "c:\temp\" & {Product_Type.Product Type Name} & "_QR.png"),
"c:\temp\" & "Circular_" + {Product_Type.Product Type Name} & "_QR.png",
0,0,112,112,"[Circular:225,225,225]")
```

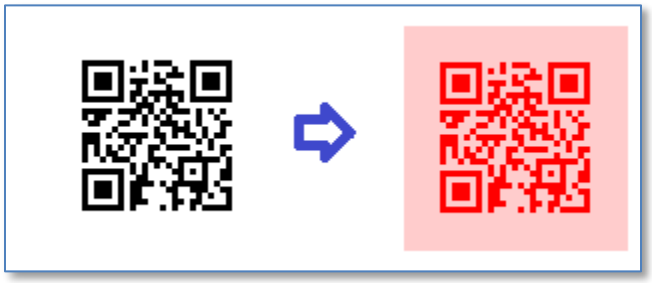
uflImageColorRemap()

Arguments: (**url** or **imagefile path & name**, new image file path & name,
FromToColors,
Options)

Returns: The path to the new re-colored image file or the text of an error message.

The FromToColors argument is constructed as an array of RGB pairs, like these:
"0,0,0>>255,0,0" (convert black to red)
"255,255,255>>255,204,204" (convert white to light red)
- or, including the Opacity element before the RGB elements:
"255,0,0,0>>255,255,0,0" (convert black to red)
"255,255,255,255>>255,255,204,204" (convert white to light red)

Either of these alternatives result in the following transformation:



Here is an example of a Crystal formula that generates the QR Code (in Black & White) and then transforms the image to red on light red:

uFLBBarcodeQR({@TextContent}, "", False, "L", 224, 224, 0, "c:\temp\" &
{@ProductType} & "_QR.png");

uFLImageColorRemap("c:\temp\" & {@ProductType} & "_QR.png",
"c:\temp\" & {@ProductType} & "_QR_Red.png",
["0,0,0>>255,0,0","255,255,255>>255,204,204"],
"");

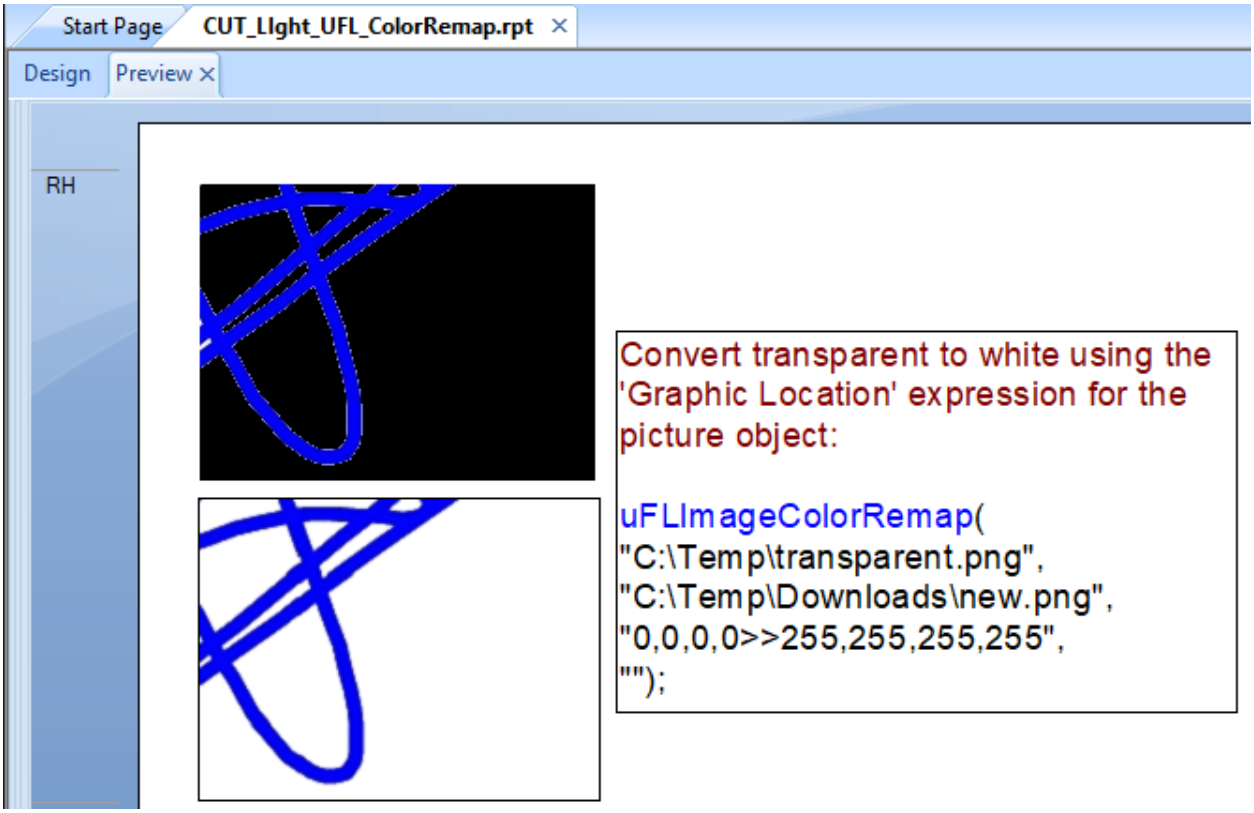
Notes:

- Including the Opacity argument is needed only if you wish to change opacity.
- Leave the Options argument blank (""). It is included to support future functionality

Fixing black background in png files

A typical problem in using png files in a Crystal report is that transparent background shows up as black.

The image below demonstrates how changing transparent and black to opaque and white solves the problem:



ufl**File2Image()**

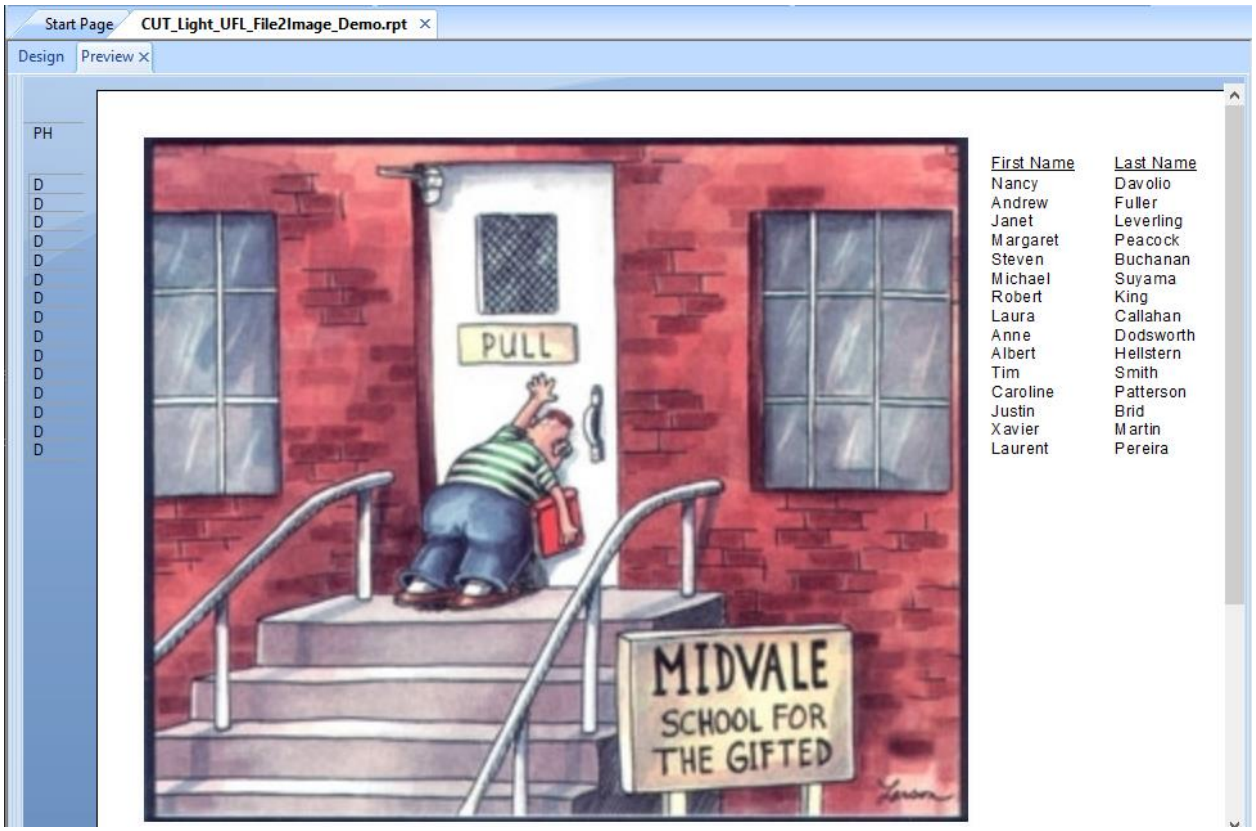
Arguments: (

`SourceFile:` currently, only PDF files are supported,
`PageN`: the page number to target for conversion to image,
`LocalFilePathName`: the path and name of the image file (png, jpg, jpeg, or bmp),
`Width` (pixels), **`Height`** (pixels),
`MaintainProportion`: set to True to maintain the source page proportions,
Options: leave blank (“”)

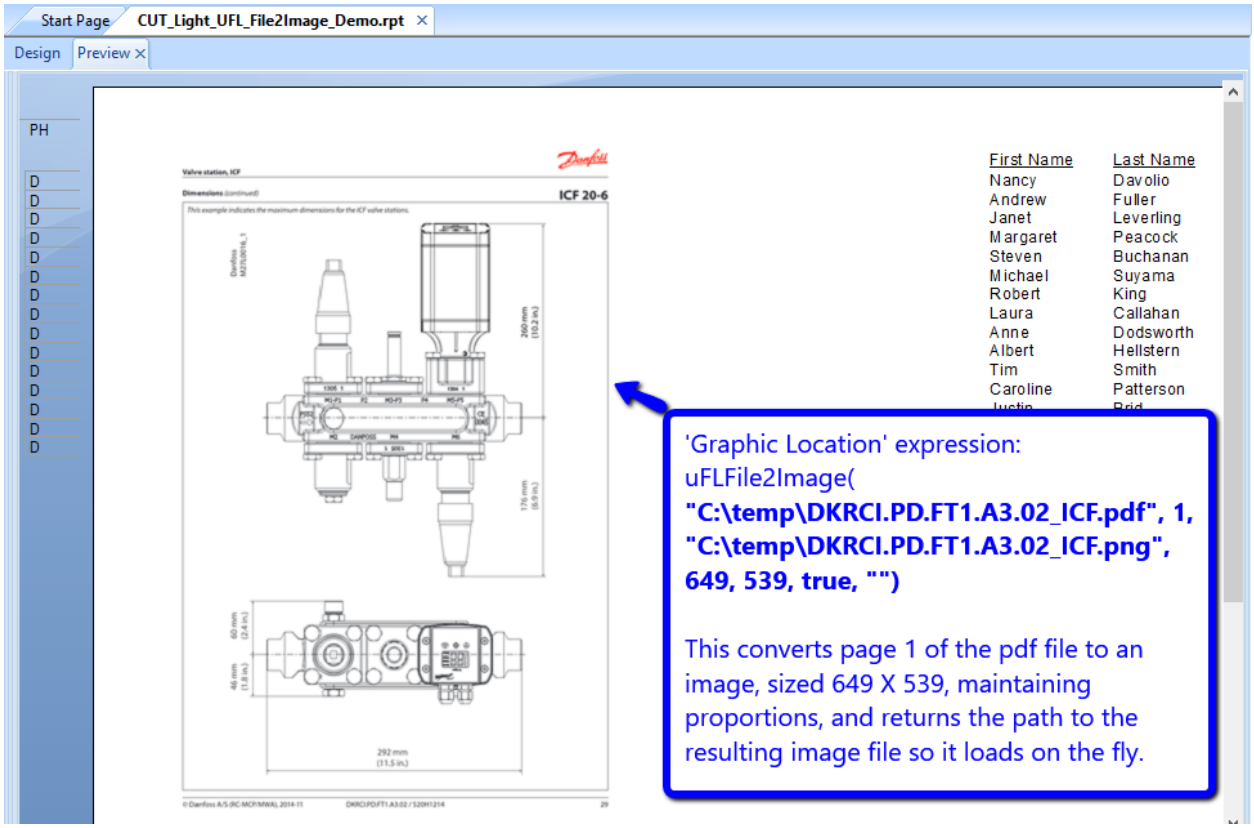
Returns: The path to the new resized image file.
If error occurs, the path is to an image of the error text.

This allows you to simply use this function in the ‘Graphic Location’ expression for a static image, so Crystal replaces it on the fly with the path to the generated image.

Example: Crystal report before Graphic Location expression:



And after using the function in the ‘Graphic Location’ expression:



Notes:

- Width & Height indicate the width and height of the resized image.
  Set both to 0 if you wish to maintain the size of the source page.
- Leave the Options argument blank (""). It is included to support future functionality
- To import a multi-page pdf as page images into a Crystal report, you can use uflPageCount() to find the total number of pages. Use multiple sections in the report (e.g. RFa, RFb, RFc…) to accommodate the maximum expected number of pages. Suppress the sections beyond the number of total pages. And use each visible section to bring in the corresponding pdf page.
  See video demo.

ufl**httpToImage()**

This function is described under the Web / HTML / Google / Translate chapter here.

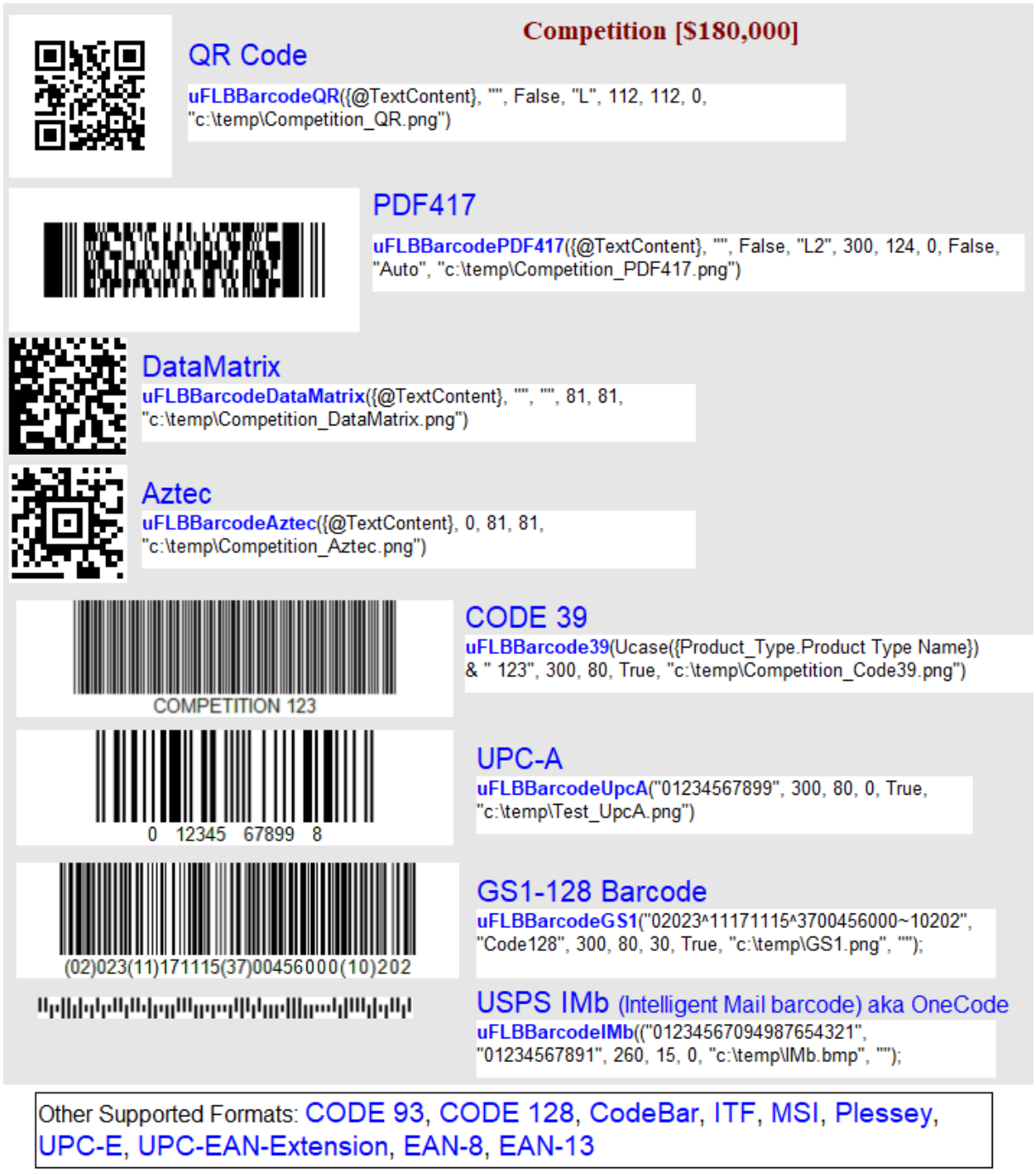
ufl**Decode2ImageFile()**

This function is described under the String/Text/Json chapter here.

Barcodes

CUT Light can generate **18 types of barcodes**. The function generated the image on the fly and return the path to that image file for use in the graphic location expression of a dummy image so it gets replaced by the new image . The process is **very fast** and provides several advantages:

1. **No dependency on special fonts**
2. Can **rotate the image** using uflImageReorient()
3. On failure (e.g. unacceptable value to encode) the barcode image is replaced with an image of the error message. This **simplifies troubleshooting** (e.g. bad value to encode).



Other Supported Formats: CODE 93, CODE 128, CodeBar, ITF, MSI, Plessey, UPC-E, UPC-EAN-Extension, EAN-8, EAN-13

uflBBarcodeQR()

Arguments: (TextContent(), CharacterSet, DisableECI, ErrorCorrectionLevel,
ImageWidth, ImageHeight, AddedMargin, LocalFilePathName)

Barcode version returns "OK" or error message (might be removed in future release)
BBarcode version is easier to use: it returns path to image of barcode or image of error.

This function allows you to generate a QR Code image so you can then load the image into a picture object using a dynamic Graphic Location expression (see example).

TextContent() divides the desired content into an array with 254-charcter segments. You can see example of how content can be chopped into such an array in the section discussing the HTML2Image() function. **If the content is less the 254 characters, you can simply use it as that argument without converting it to an array**.

CharacterSet: leave blank **""** to use the default ("ISO-8859-1"). Use **"utf-8"** for Arabic etc.
DisableECI (Boolean): Use **False**, unless CharacterSet is "**utf-8**" and reading fails
ErrorCorrectionLevel: **"L"** for Low (default), "M" - Medium, "Q" - Quartile, "H" – High
ImageWidth, **ImageHeight**: size (height = width) of the image in pixels.
AddedMargin: set to -1 to remove the white margin (aka "quiet zone") around the barcode.
Set to **0** to generate the default quiet zone around the barcode.
Set to **-999** to avoid any white padding (might cause fuzzy edges due to forced scaling).

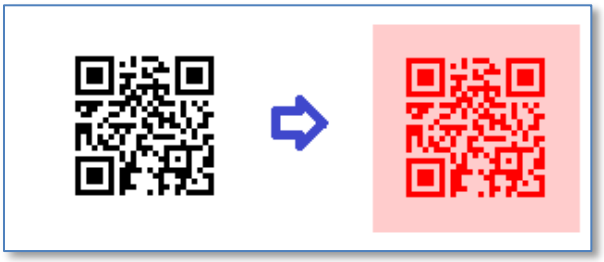
LocalFilePathName specified a unique file path & name of the barcode image file. That path & file name is then used to set the dynamic Graphic Location path to load the image into the Crystal report. File extension must be **.png**, **.bmp**, **.jpg or .jpeg**

The QR barcode shown in the example report image was generated with the following expression in the Graphic Location of the image object:
uFL**BB**arcodeQR({@Text}, "", False, "L", 112, 112, 0, "c:\temp\Competition.png")

- A QR Code can specify the full content of an email message. See link1 and link2.
- With 75 DPI screen resolution, 75 pixels = 1 inch.

Adding Colors

You can add color using uflImageColorRemap() function:

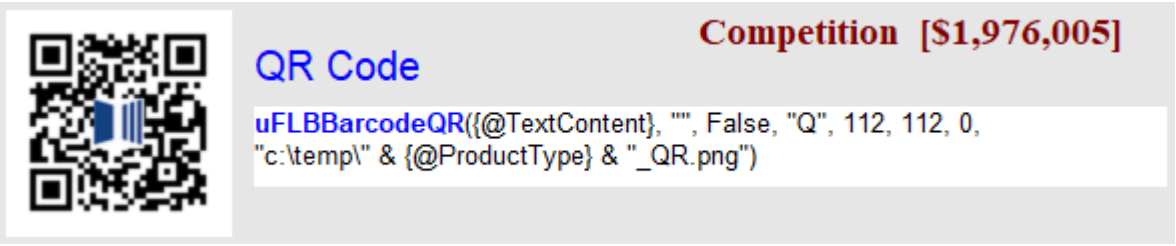


Adding a Logo

When generating the QR Code image, if the target folder contains an image called **QR_Overlay_Logo.png**, CUT Light automatically centers and overlays.

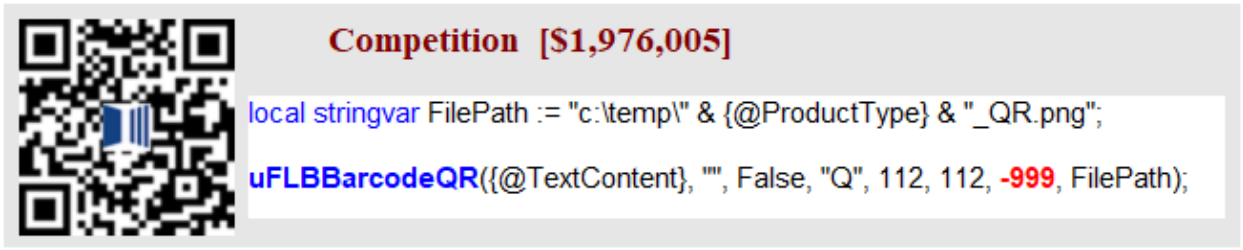
Make sure the Logo image is smaller than the QR Code.

Also, increase the ***ErrorCorrectionLevel*** argument to ensure successful reading. In the example below, the ***ErrorCorrectionLevel*** argument is set to '**Q**'.



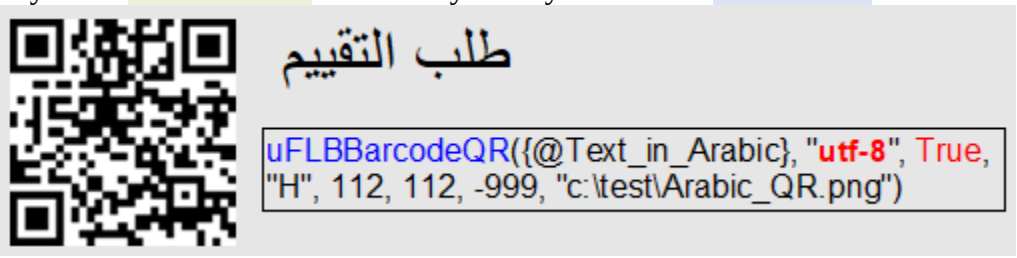
Removing Padding

Even with **AddedMargin** set to -1, the generated image might have some white padding (because widths must be uniform multiples of pixels). To remove the padding, you can set the **AddedMargin** to **-999** (might cause fuzzy edges due to forced scaling):



Using utf-8 as character set

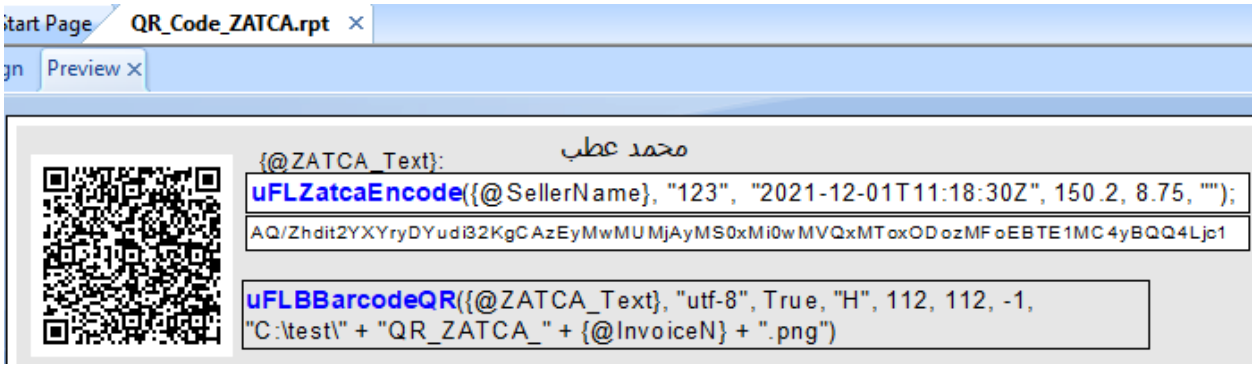
If you set **CharacterSet** to "utf-8" you may need to set **DisableECI** to True:



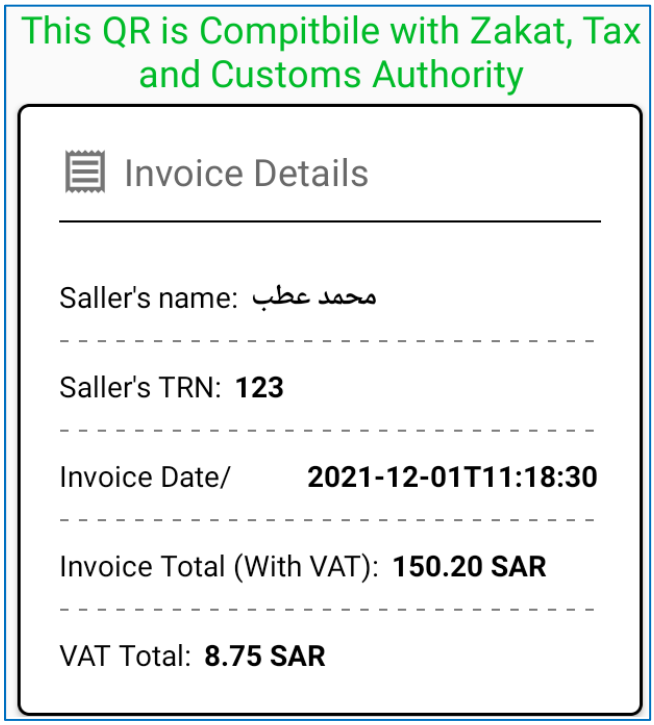
uflZatcaEncode()

Arguments: (SellerName, VatNumber, InvoiceDateTime, InvoiceTotal, Vat, Options)
Returns: encoded Base64 string or, '***' followed by the error message. For example:
uflZatcaEncode ({@SellerName}, "123", "2021-12-01T11:18:30Z", 150.2, 8.75, "") returns:
'AQ/Zhdit2YXYryDYudi32KgCAzEyMwMUMjAyMS0xMi0wMVQxMToxODozMFoEBTE1MC4yBQQ4Ljc1'
Which you can pass as the text content for uflBBarcodeQR().

SellerName a String (for example, "محمد عطب")
VatNumber a String
InvoiceDateTime String as Zulu ISO8601 format. Leave blank ("") for current UTC time.
InvoiceTotal must be passed in as a Double. Use Crystal's CDbl() function if necessary.
Vat must be passed in as a Double. Use Crystal's CDbl() function if necessary.
Options: leave blank. Reserved for future use.



→



uflBBarcodeGS1()

Arguments: (TextContent,**BarcodeType**, ImageWidth, ImageHeight, **AddedMargin**, ShowText, LocalFilePathName, Options)

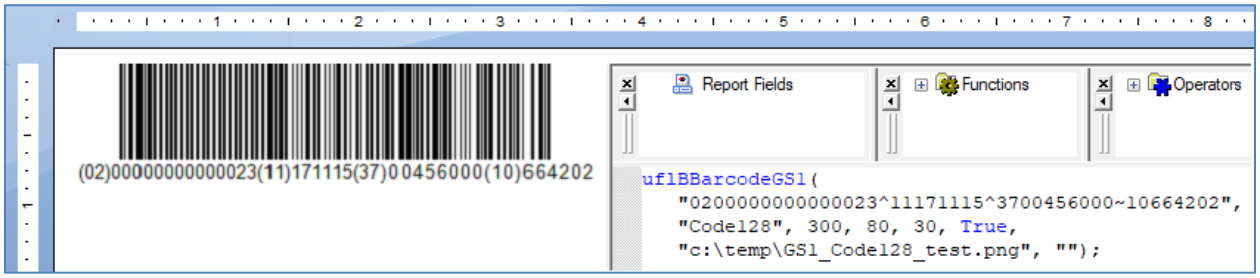
This function allows you to generate a **GS1-128 Barcode** image so you can then load the image into a picture object using a dynamic Graphic Location expression.

TextContent() divides the desired content into an array with 254-charcter segments. You can see example of how content can be chopped into such an array in the section discussing the HTML2Image() function. **If the content is less the 254 characters, you can simply use it as that argument without converting it to an array**.
Within the text content:
~ character indicates where FNC1 character should be inserted as a field separator
^ character indicates where a new field starts without needing the special field separator

BarcodeType: currently, only **"Code128"** is supports.
DisableECI (Boolean): Use **False**, unless CharacterSet is "UTF-8" and reading fails
ImageWidth, **ImageHeight**: size of the image in pixels.
AddedMargin: extra white margin in points. Typically, set to **30**
ShowText: True/False. set to True to show a nicely formatted text below.
LocalFilePathName specified a unique file path & name of the barcode image file. That path & file name is then used to set the dynamic Graphic Location path to load the image into the Crystal report. File extension must be **.png**, **.bmp**, **.jpg or .jpeg**
Options: leave as blank: ""

The GS1 Code 128 barcode in the sample image below was generated with the following expression in the Graphic Location of the image object:

```
uFLBBarcodeGS1("02000000000000023^11171115^3700456000~10664202",
"Code128", 300, 80, 30, True, "c:\temp\GS1_Code128_test.png", "")
```



uflBBarcodePDF417()

Arguments: (TextContent(), CharacterSet, DisableECI, ErrorCorrectionLevel,
ImageWidth, ImageHeight, AddedMargin, Compact, Compaction,
LocalFilePathName)

Barcode version returns "OK" or error message (might be removed in future release)
BBarcode version is easier to use: it returns path to image of barcode or image of error.

This function allows you to generate a PDF-417 barcode image so you can then load the image into a picture object using a dynamic Graphic Location expression (see example).

TextContent() divides the desired content into an array with 254-charcter segments. You can see example of how content can be chopped into such an array in the section discussing the HTML2Image() function. **If the content is less the 254 characters, you can simply use it as that argument without converting it to an array**.

CharacterSet: leave blank **""** to use the default ("ISO-8859-1")
DisableECI (Boolean): Use **False**, unless CharacterSet is "UTF-8" and reading fails
ErrorCorrectionLevel: "L0" to "L8" (**"L2"** is default)
ImageWidth, **ImageHeight**: size of the image in pixels.
AddedMargin: extra white margin in points. Typically, set to **0**
Compact (Boolean): True or False
Compaction: "AUTO", "BYTE", "TEXT" or "NUMERIC". Default is **"AUTO"**

LocalFilePathName specified a unique file path & name of the barcode image file. That path & file name is then used to set the dynamic Graphic Location path to load the image into the Crystal report. File extension must be **.png**, **.bmp**, **.jpg or .jpeg**

The PDF-417 barcode in the example report image was generated with the following expression in the Graphic Location of the image object:

```
uFLBBarcodePDF417({@TextContent}, "", False, "L", 300, 124, 0, False, "", "c:\temp\" &
{@ProductType} & "_PDF417.png")
```

Notes:

- If height > width, the barcode is rendered vertically instead of horizontally.
- With 75 DPI screen resolution, 75 pixels = 1 inch.

uflBBarcodeDataMatrix ()

Arguments: (TextContent(), DefaultEncodation, Shape,
ImageWidth, ImageHeight, LocalFilePathName)

Barcode version returns "OK" or error message (might be removed in future release)
BBarcode version is easier to use: it returns path to image of barcode or image of error.

This function allows you to generate a Data Matrix barcode image so you can then load the image into a picture object using a dynamic Graphic Location expression (see example).

TextContent() divides the desired content into an array with 254-charcter segments. You can see example of how content can be chopped into such an array in the section discussing the HTML2Image() function. **If the content is less the 254 characters, you can simply use it as that argument without converting it to an array**.

DefaultEncodation: leave blank **""** or "ASCII", "Base256", "C40", "EDIFACT", "Text", "X12"
Shape: Leave blank "", or use "Rectangle" or "Square" to force the shape.
ImageWidth, **ImageHeight**: size of the image in pixels.
With 75 DPI screen resolution, 75 pixels = 1 inch.
Listing of sizes: https://www.activebarcode.com/codes/datamatrix_examples
LocalFilePathName specified a unique file path & name of the barcode image file. That path & file name is then used to set the dynamic Graphic Location path to load the image into the Crystal report. File extension must be **.png**, **.bmp**, **.jpg or .jpeg**

The Data Matrix barcode in the example report image was generated with the following expression in the Graphic Location of the image object:

```
uFLBBarcodeDataMatrix({@TextContent}, "", "", 81, 81,
"c:\temp\" & {@ProductType} & "_QR.png")
```

uflBBarcodeAztec()

Arguments: (TextContent(), Layers, ImageWidth, ImageHeight, LocalFilePathName)

Barcode version returns "OK" or error message (might be removed in future release)
BBarcode version is easier to use: it returns path to image of barcode or image of error.

This function allows you to generate an Aztec barcode image so you can then load the image into a picture object using a dynamic Graphic Location expression (see example).

TextContent() divides the desired content into an array with 254-charcter segments. You can see example of how content can be chopped into such an array in the section discussing the HTML2Image() function. **If the content is less the 254 characters, you can simply use it as that argument without converting it to an array**.

Layers: a number -- not Text! For most use cases, set the number to zero (0). This automatically sets the number of layers to the minimum required to render the content.
ImageWidth, **ImageHeight**: size (height = width) of the image in pixels.
LocalFilePathName specified a unique file path & name of the barcode image file. That path & file name is then used to set the dynamic Graphic Location path to load the image into the Crystal report. File extension must be **.png**, **.bmp**, **.jpg or .jpeg**

The Aztec barcode in the example report image was generated with the following expression in the Graphic Location of the image object:

```
uFLBBarcodeAztec ({@TextContent}, 0, 81, 81,
"c:\temp\" & {@ProductType} & "_Aztec.png")
```

uflBBarcodeIMb()

Arguments: (TrackingCode, RoutingCode, ImageWidth, ImageHeight, Margin
LocalFilePathName, Options)

Returns path to image of barcode or image of error.

This function allows you to generate a ***USPS Intelligent Mail barcode***. The ***IMb code*** is also sometimes referred to as a ***One Code Solution***, or ***4-State Customer Barcode***, abbreviated as ***4CB***, ***4-CB*** or ***USPS4CB.***

The barcode is saved to the specified LocalFilePathName, so you can then load it into a picture object using a dynamic Graphic Location expression.

TrackingCode is a string of exactly 20 digits
RoutingCode is a string of 0, 5, 9, Or 11 digits
ImageWidth, **ImageHeight**: size (height = width) of the image in pixels.
With 75 DPI screen resolution, 75 pixels = 1 inch.
Margin: the white padding around the barcode in pixels
LocalFilePathName specified a unique file path & name of the barcode image file. That path & file name is then used to set the dynamic Graphic Location path to load the image into the Crystal report. File extension must be **.png**, **.bmp**, **.jpg or .jpeg**
Options: leave blank ("")

The IMb barcode in the image below:

was generated with the following expression in the Graphic Location of the image object:
uFLBBarcodeIMb("01234567094987654321", "01234567891", 260, 15, 0,
"c:\temp\IMb.bmp", "")

uflBBarcode39()

Arguments: (TextContent,ImageWidth, ImageHeight, ShowText, LocalFilePathName)

Barcode version returns "OK" or error message (might be removed in future release)
BBarcode version is easier to use: it returns path to image of barcode or image of error.

This function allows you to generate a Code 39 barcode image so you can then load the image into a picture object using a dynamic Graphic Location expression (see example).

TextContent is the desired content.

ImageWidth, **ImageHeight**: size of the image in pixels.
With 75 DPI screen resolution, 75 pixels = 1 inch.

ShowText: set to TRUE to display the text below the barcode.

LocalFilePathName specified a unique file path & name of the barcode image file. That path & file name is then used to set the dynamic Graphic Location path to load the image into the Crystal report. File extension must be **.png**, **.bmp**, **.jpg or .jpeg**

The Code 39 barcode shown in this example report image was generated with the following expression in the Graphic Location of the image object:

uFLBBarcode39 (Ucase({Product_Type.Product Type Name}) & " 123", 300, 80, True, "c:\temp\" & {@ProductType} & "_Code39.png")

uflBBarcode93()
Same instructions and arguments as uflBBarcode39()

uflBBarcode128()
Same instructions and arguments as uflBBarcode39()

uflBBarcodeITF()
Same instructions and arguments as uflBBarcode39()

uflBBarcodeCodeBar()
Same instructions and arguments as uflBBarcode39()

uflBBarcodeMSI()
Same instructions and arguments as uflBBarcode39()

uflBBarcodePlessey()
Same instructions and arguments as uflBBarcode39()

uflBBarcodeUpcA()

Arguments: (TextContent,ImageWidth, ImageHeight, AddedMargin, ShowText, LocalFilePathName)

Barcode version returns "OK" or error message (might be removed in future release)
BBarcode version is easier to use: it returns path to image of barcode or image of error.

This function allows you to generate a UPC-A barcode image so you can then load the image into a picture object using a dynamic Graphic Location expression (see example).

TextContent is the desired content.

ImageWidth, **ImageHeight**: size of the image in in pixels.
With 75 DPI screen resolution, 75 pixels = 1 inch.

AddedMargin: extra margin to add to the sides of the barcode

ShowText: set to TRUE to display the text below the barcode.

LocalFilePathName specified a unique file path & name of the barcode image file. That path & file name is then used to set the dynamic Graphic Location path to load the image into the Crystal report. File extension must be **.png**, **.bmp**, **.jpg or .jpeg**

The UPC-A barcode shown in this example report image was generated with the following expression in the Graphic Location of the image object:

```
uFLBBarcodeUpcA ("01234567899", 300, 80, 0, True,
"c:\temp\" & "Test" & "_UpcA.png")
```

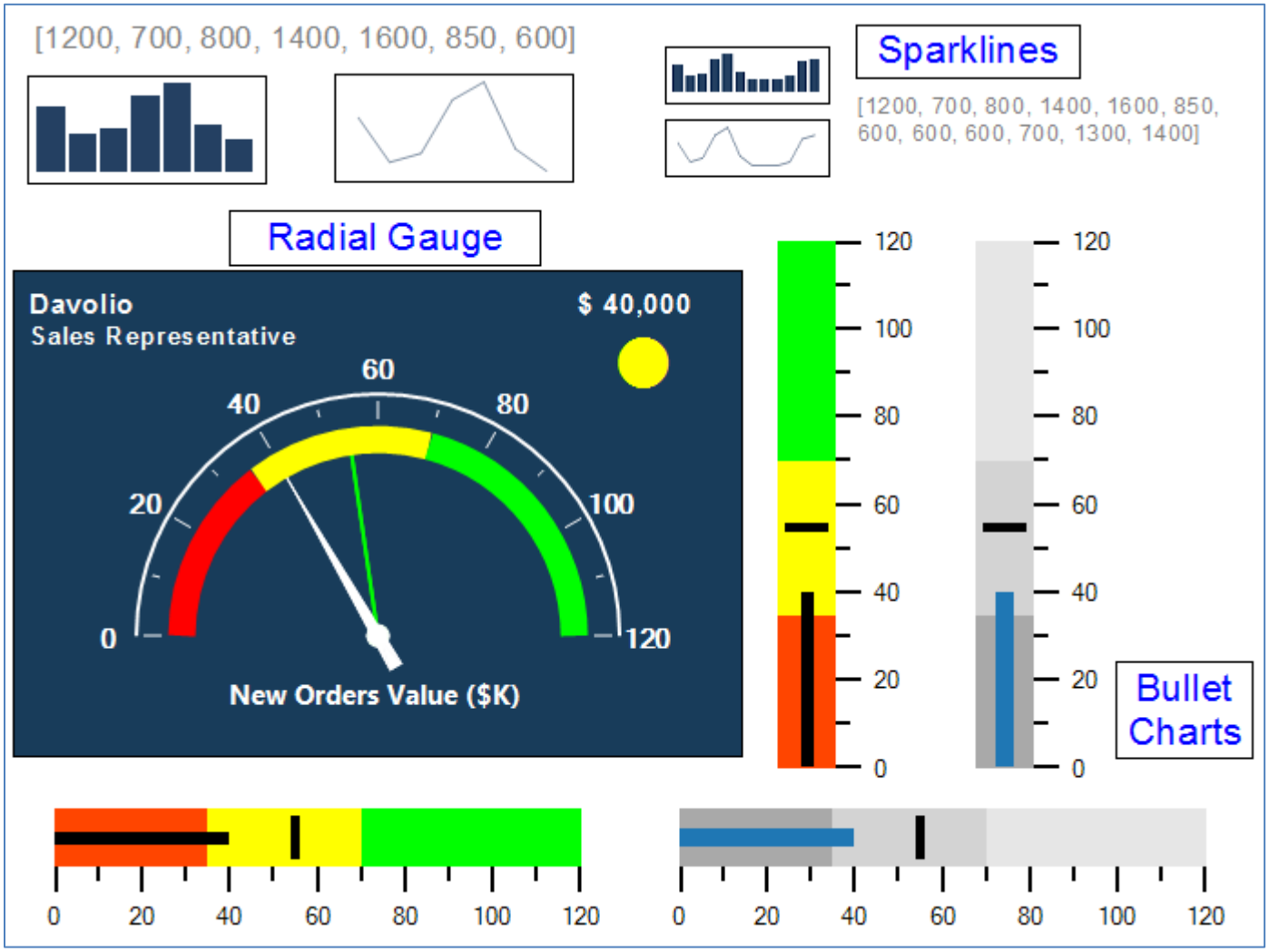
uflBBarcodeUpcE()
Same instructions and arguments as uflBBarcodeUpcA()

uflBBarcodeUpcEanExtension()
Same instructions and arguments as uflBBarcodeUpcA()

uflBBarcodeEAN8()
Same instructions and arguments as uflBBarcodeUpcA()

uflBBarcodeEAN13()
Same instructions and arguments as uflBBarcodeUpcA()

Charts



The functions in this section generate Gauge, Bullet Charts, or Sparklines as images that gets inserted on the fly into the report via the 'Graphic Location' expression of a picture object.

Each function takes a set of arguments and returns the path to the generated image. If an error occurs, the generated image provides the text of the error message.

The image above shows examples of these charts.

uflBulletChart()

Arguments: (ImageWidth, ImageHeight, Caption, RangeValues, MeasureValue, ComparativeValue, Style, LocalFilePathName, Options)

Returns LocalFilePathName to image of chart or image of error (facilitates troubleshooting).

Generate a bullet chart image for loading via Graphic Location expression.

ImageWidth, **ImageHeight**: in pixels. if height > width -> vertical orientation.
With 75 DPI screen resolution, 75 pixels = 1 inch.

Caption (typically, should leave blank and use Crystal text/formula object for caption).

RangeValues: start value + 3 values for end of each color/shade range. Like [0, 35, 70, 120]

MeasureValue: the **KPI value** to plot as a long bar

ComparativeValue: value for **short reference bar**

Style: currently, "Color1" or "Gray1"

LocalFilePathName: a unique file path & name of the chart image file to be loaded into the Crystal report via a picture *Graphic Location* expression. **.png**, **.bmp**, **.jpg or .jpeg**

Options: "[ReverseColors]" reverses the order of colors ("Good" is lower numbers).
"[HideScale]" causes the scale to be hidden.
Combined with **uflImageCrop()** this allows reports like the image shown below:

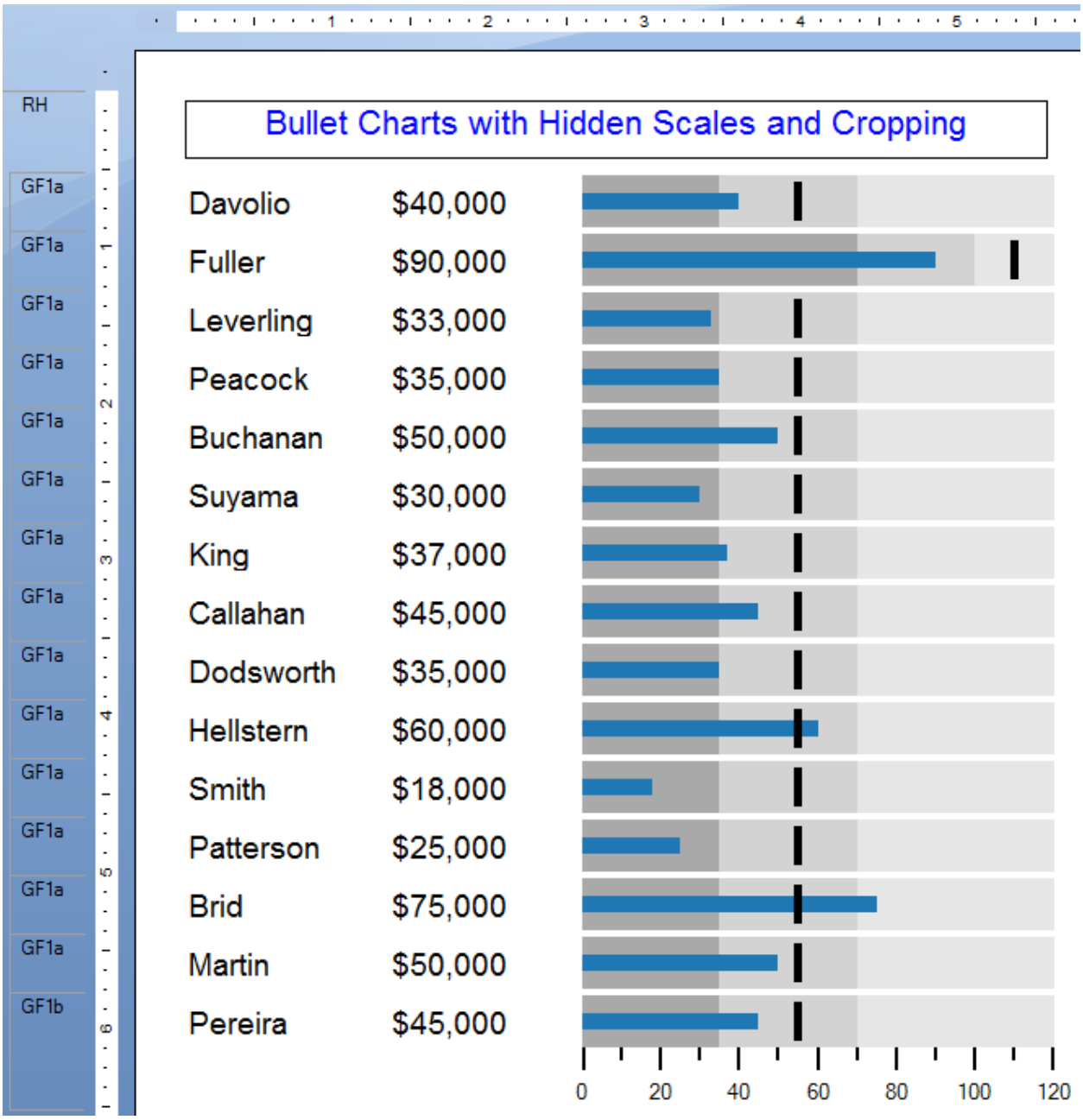
The bullet charts shown in this image were generated with 'Graphic Location' expressions such as:

```
uFLBulletChart(80, 350, "", [0, 35, 70, 120], cdbl({Employee.Salary}/1000), 55, "Color1",
        "c:\temp\" & {Employee.Last Name} & ".bmp", "");
```

The cropped bullet charts in the image below use this 'Graphic Location' expression:

```
Global numbervar array BulletRangeValues := [0, 35, 70, 120];
Local numbervar ComparisonValue := 55;
IF instr({Employee.Position}, "President") > 0 Then (BulletRangeValues := [0, 70, 100, 120];
   ComparisonValue := 110);  // Fuller, the president gets different ranges and benchmark.
//  Blank caption and hidden scale.
uFLBulletChart(350, 80, "", BulletRangeValues, cdbl({Employee.Salary}/1000),
        ComparisonValue, "Gray1",
"c:\temp\HGBullet_" & {Employee.Last Name} & ".bmp","[HideScale]");
// Crop the image to allow tight side-by-side layout
```

uFLImageCrop("c:\temp\HGBullet_" & {Employee.Last Name} & ".bmp", "c:\temp\HGBullet_" & {Employee.Last Name} & "_Cropped.bmp", 20, 5, 310, 35, "");
In the report below, the cropped version of the image is shown in GF1a. An uncropped version with scale visible is in GF1b. Using onlastrecord logic, GF1a is suppressed and GF1b is visible for the last group in the report.



uflGaugeRadial()

Arguments: (Caption, RangeValues, NeedleValues, XML options file,
LocalFilePathName, Options)

Returns LocalFilePathName to image of chart or image of error (facilitates troubleshooting).

Generate a Radial Gauge image for loading via Graphic Location expression.

Caption: can be left blank if you wish to overlay a label using a Crystal field/formula. Can insert new line character for multi-line caption.

RangeValues: an array of numeric values specifying start of each color band + the end value.

Needle Values: an array of numbers or a single value for the main needle. Last number is always considered to be the value for the main needle.

XML options file: an XML file provides **image size** and many **design options**.
Contact Millet Software for consulting to generate the XML file with other options.
Or you can tweak the XML file yourself (e.g., using notepad).

LocalFilePathName: a unique file path & name of the chart image file to be loaded into the Crystal report via a picture *Graphic Location* expression. **.png**, **.bmp**, **.jpg or .jpeg**

Options: for future use. leave blank

The radial gauge shown in the image at the start of this chapter was generated with the following expression in the 'Graphic Location' of the picture object:

```
Global numbervar array RangeValues := [0, 35, 70, 120];
Local numbervar array NeedleValues := [55, Cdbl({Employee.Salary}/1000)];
uFLGaugeRadial("New Orders Value ($K)",
        RangeValues, NeedleValues, "C:\TEMP\Gauge3.xml",
        "c:\temp\" & {Employee.Last Name} & ".bmp", "");
```

uflSparkLine()

Arguments: (ImageWidth, ImageHeight, Values, Type, Style, LocalFilePathName, Options)

Returns LocalFilePathName to image of chart or image of error (facilitates troubleshooting).

Generate a sparkline image for loading via Graphic Location expression.

ImageWidth, **ImageHeight**: in pixels. if height > width -> vertical orientation.
With 75 DPI screen resolution, 75 pixels = 1 inch.

Values: an array of numeric values.

Type: "Line", "Column" or "WinLoss"

Style: currently ignored. Leave as blank text ("")

LocalFilePathName: a unique file path & name of the chart image file to be loaded into the Crystal report via a picture *Graphic Location* expression. **.png**, **.bmp**, **.jpg or .jpeg**

Options: for future use. leave blank

The sparklines shown at the top of the image above were generated with 'Graphic Location' expressions such as:

```
uFLSparkLine(130, 58, [1200, 700, 800, 1400, 1600, 850, 600], "Column", "",
        "c:\temp\Sparkline_Column.bmp", "");
```

String/Text/JSON

uflPopulateTemplate()

Arguments: (Template, ArgumentsArray, EmptyReplacement)

Returns: The result of substituting placeholders ({0}, {1}, {2), …) in the Template string with the corresponding entries in the ArgumentsArray.
Null or empty arguments are replaced with the EmptyReplacement string. If the result is longer than 510 characters: `"Result is Longer than 510 characters"`

Assume, for example, that you wish to build the syntax for an HTML table row with 3 elements: First Name, Last Name, and Middle Initial. The following formula:

PopulateTemplate("<TR><TD>**{0}**</TD><TD>**{1}**</TD><TD>**{2}**</TD></TR>",
["Ido", "Millet", ""], "** **")

would return the following string:

<TR><TD>Ido</TD><TD>Millet</TD><TD> </TD></TR>

Note: instead of specifying an array of strings, you can pass in an array string variable. For example:

PopulateTemplate("<TR><TD>**{0}**</TD><TD>**{1}**</TD><TD>**{2}**</TD></TR>",
MyStringArrayVar, "** **")

Note: if a null/blank is replaced by a "", double spaces are replaced with a single space (only if the template did not contain double spaces). This helps in cases such as [First Initial Last].

uflExpandStringwithEnvironmentVar()

Arguments: (InputString)

Returns: The input string after expanding any references to environment variables within it to their dynamic values.

For example, the following Crystal formula:

ExpandStringwithEnvironmentVar("%UserName% -> %temp%")

returns the following on my PC (where my user id is ixm7):

ixm7 -> C:\Users\ixm7\AppData\Local\Temp

uflReplaceAccentedChars()

Arguments: (String)

This function replaces all accented characters in the input string with their non-accented versions (for example, ê/ë/è/é → e). Upper & lower case are preserved.

Returns: the converted string.

uflEncode()

Arguments:

TextContent() as String Array
EncodeType as String (currently, only "Barcode128", "Base64", or "Hex" are supported)
EncodeOptions (to encode an integer value, set to "Integer_x2") where x2 is the format.
SegmentN as integer

This function combines the string array into a single string, encodes, and returns the requested segment.

Returns: a string with the encoded string or an error message starting with "*** "

Sample Code:

```
Local StringVar TextContent := "Text To Encode";
Local StringVar array MyStringArray;

IF Len(TextContent) = 0 Then
(redim MyStringArray [1];
 MyStringArray[1] = "";)
Else
(   Local numbervar segments := RoundUp(Len(TextContent)/254);
    redim MyStringArray [segments];
    Local numbervar index ;
    for index := 0 to segments - 1 step 1 do
        (MyStringArray[index + 1] := mid(TextContent, 1 + (index * 254) , 254)) ;
);

Local StringVar sResult;
NumberVar i;
i := 1;
// Keep appending segments of 254 characters until we reach the end
while uFLEncode(MyStringArray, "Barcode128" , "", i) <> "" Do
( sResult := sResult + uFLEncode(MyStringArray, "Barcode128" , "", i);
i := i + 1
);
sResult;
```

uflDecode()

Arguments:

TextContent() as String Array
EncodeType as String (currently, only `"Base64"` or `"Hexadecimal"` are supported)
EncodeOptions (leave as blank `""`)
SegmentN as integer

This function combines the string array into a single string, decodes, and returns the requested segment. For example, **uflDecode**`({"VGVzdDEyMzQ="}, "Base64", "", 1))` returns `Test1234`

Returns: the decoded string or an error message starting with `"*** "`

Sample Code:

```
Local StringVar TextContent := "Text To Decode";
Local StringVar array MyStringArray;

IF Len(TextContent) = 0 Then
(redim MyStringArray [1];
 MyStringArray[1] = "";)
Else
(   Local numbervar segments := RoundUp(Len(TextContent)/254);
    redim MyStringArray [segments];
    Local numbervar index ;
    for index := 0 to segments - 1 step 1 do
       (MyStringArray[index + 1] := mid(TextContent, 1 + (index * 254) , 254)) ;
);

Local StringVar sResult;
NumberVar i;
i := 1;
// Keep appending segments of 254 characters until we reach the end
while uFLDecode(MyStringArray, "Base64" , "", i) <> "" Do
( sResult := sResult + uFLDecode(MyStringArray, "Base64" , "", i);
i := i + 1
);
sResult;
```

ufl**Decode2ImageFile()**

Arguments:

TextContent() as String Array
EncodeType as String (currently, only `"Base64"` or `"Hexadecimal"` are supported)
LocalFilePathName as String (e.g. `"c:\temp\test.png"`)
Options (leave as blank `""`)

This function is typically used to convert Base64 or Hexadecimal content stored in the database to an image file for dynamic loading into a report. The loading is typically done via the 'Graphic Location' expression of a dummy image. If image resizing is desired, use the ImageResize() function.

Returns: the path to the newly created image file.
If an error occurs, the image contains the text of the error fitted to an image sized 200x200.

Sample Code:

```
Local StringVar TextContent := {Product.Image};
Local StringVar array MyStringArray;

IF Len(TextContent) = 0 Then
(redim MyStringArray [1];
 MyStringArray[1] = "";)
Else
(   Local numbervar segments := RoundUp(Len(TextContent)/254);
    redim MyStringArray [segments];
    Local numbervar index ;
    for index := 0 to segments - 1 step 1 do
       (MyStringArray[index + 1] := mid(TextContent, 1 + (index * 254) , 254)) ;
);

uFLDecode2ImageFile(MyStringArray, "Base64", "c:\temp\" + {Product.ID} + ".png", "");
```

ufl**Hex2Ascii()**

Arguments: (String)

This function takes hex string eg "ed0972ba628b29f0ec" and returns its ascii equivalent

Returns: the ascii string

ufl**Hex2Number()**

Arguments: (String)

This function takes hex string (for example "000130D") and returns its numeric value (4877 in this case)

Returns: the numeric value of the hex string.

uflConvertRTF2Text()

Arguments: (RTFText() as String Array, SegmentN as integer)

This function converts RTF Text to Plain Text and returns the Nth 254-character segment from the result. See sample image.

RTFText argument divides the RTF string into an array with 254-charcter segments. In the example below, the {@RTF} content gets chopped into a MyStringArray using the code in red. The code in blue then calls **uFLConvertRTF2Text(MyStringArray, i)** in a loop until no more segments are returned.

```
Local StringVar array MyStringArray;
IF Len({@RTF}) = 0 Then
(redim MyStringArray [1];
 MyStringArray[1] = "";)
Else
(   Local numbervar segments := RoundUp(Len({@RTF})/254);
    redim MyStringArray [segments];
    Local numbervar index ;
    for index := 0 to segments - 1 step 1 do
       (MyStringArray[index + 1] := mid({@RTF}, 1 + (index * 254) , 254)) ;
);

Local StringVar sResult;
NumberVar i;
i := 1;
// Keep appending segments of 254 characters until we reach the end
while uFLConvertRTF2Text(MyStringArray, i) <>"" Do
(
sResult := sResult + uFLConvertRTF2Text(MyStringArray, i);
i := i + 1
);
sResult;
```

uflConvertRTFFileToText()

Arguments: (RTFFile, TextFile)

This function takes an RTF File and converts it to a plain text file.

Returns: "OK" or failure message (e.g. "RTF File not found").

uflJsonGet()

Arguments: (JsonText() as String Array, JsonPath as String, RequestType as String, SegmentN as integer, Options as String)

Returns: This function locates the content found at the specified path. It returns the Nth 254-character segment from the result. Errors are returned following a '* ' prefix.

The JsonText() argument divides the JSON content into an array with 254-charcter segments. Or you can simply pass the full JSON content as a single string.

The JsonPath argument specifies the path to the desired content. See examples here.
The path "**mixture.arrayA[2].fruit**" locates a top-level element called "mixture",
within that a sub-element called "arrayA",
within that, the 3rd array object (indexing is zero-based, so 2 points to the 3rd element),
and within that, the value of the "fruit" element. So it returns "kiwi".

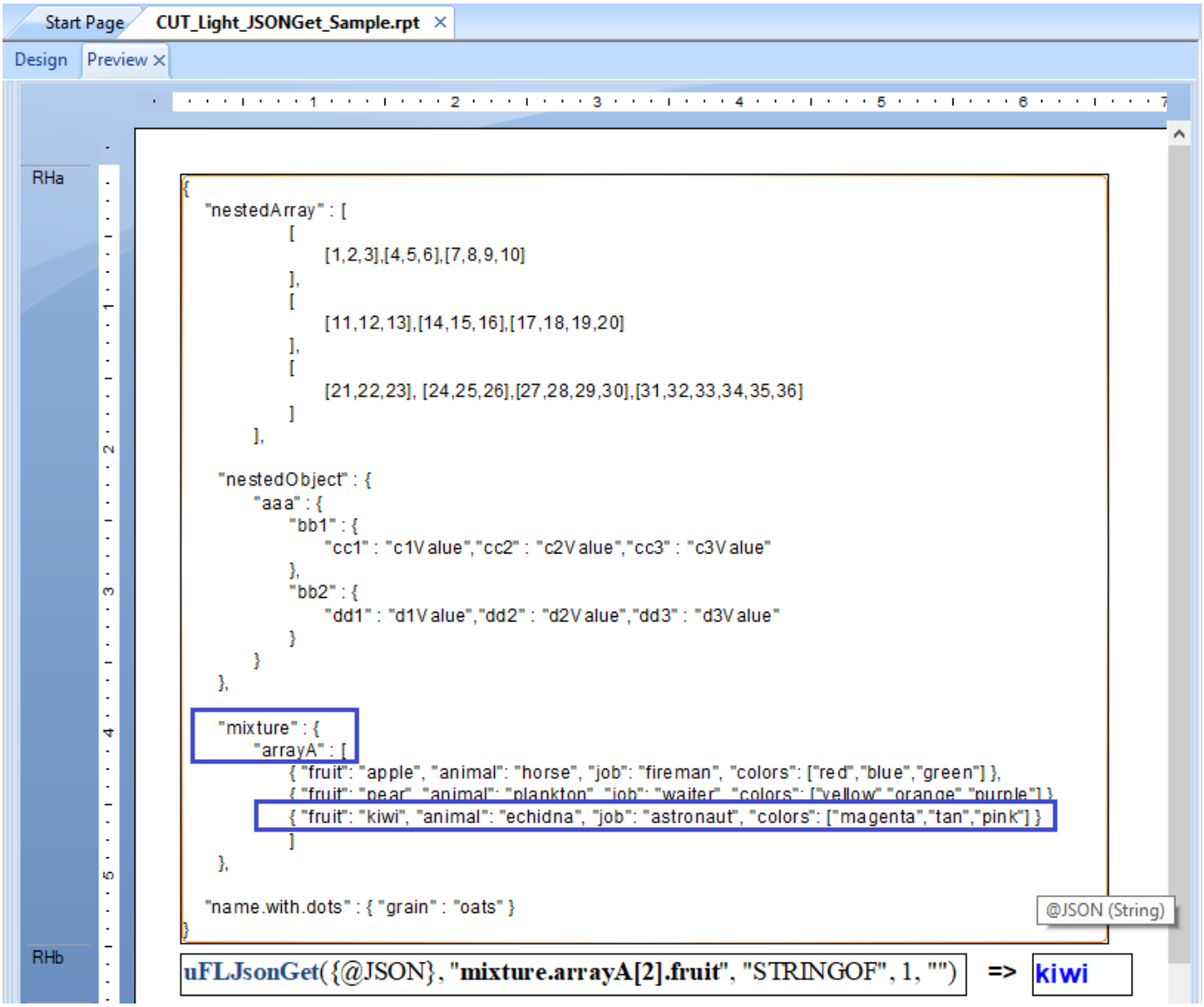
RequestType can be: "STRINGOF" (returns the targeted content as flattened string)
or "SIZEOFARRAY" (returns number of elements in the targeted array)

Simple Sample (no String Array, Getting just the 1st output segment):
uFLJsonGet({@JSON}, "mixture.arrayA[2].fruit", "STRINGOF", 1, "")

Complex Sample (with String Array as input and collecting output segments:
Calls **uFLJsonGet(MyStringArray, i, "")** in a loop until no more segments are returned.

```
Local StringVar myString := {@JSON};  Local StringVar array MyStringArray;
local stringvar ls_return := "";      Local stringvar sResult := "";
local NumberVar i;      Local numbervar segments;

IF Len(myString) = 0 Then ( redim MyStringArray [1];  MyStringArray[1] := "";)
Else ( segments := RoundUp(Len(myString)/254);      redim MyStringArray [segments];
      for i := 0 to segments - 1 step 1 do
       (MyStringArray[i + 1] := mid(myString, 1 + (i * 254) , 254)); "OK");
// Keep appending segments of 254 characters until we reach the end
i := 1;
local stringvar ls_return := uFLJsonGet(MyStringArray, "mixture.arrayA[2].fruit",
"STRINGOF", i, "");
IF Len(ls_return) > 0 Then (
  IF Left(ls_return, 1) = "*" Then sResult := ls_return
  ELSE (
    while ls_return <> "" Do (
     ls_return := uFLJsonGet(MyStringArray, "mixture.arrayA[2].fruit", "STRINGOF", i, "");
        sResult := sResult + ls_return;  i := i + 1);
    sResult;)););
```

This image demonstrates the logic:



Handling Anonymous Arrays

Your JSON data might have no name for the top node, as in this example:

```
[  { "fruit": "apple" },
   { "fruit": "pear" },
   { "fruit": "kiwi" }
]
```

In such a case, you may need to add a name on-the-fly, using the formula logic.
For example, to get the fruit value for the second object in the anonymous array above, embed it in a top json object named "root" (or any name of your choice):

```
Local StringVar myString := {@JSON};
myString := "{""root"": " + myString + "}"
Local StringVar array MyStringArray;
local stringvar ls_return := "";
Local stringvar sResult := "";
local NumberVar i; Local numbervar segments;

IF Len(myString) = 0 Then ( redim MyStringArray [1];  MyStringArray[1] := "";)
Else ( segments := RoundUp(Len(myString)/254);
      redim MyStringArray [segments];
      for i := 0 to segments - 1 step 1 do (MyStringArray[i + 1] := mid(myString, 1 + (i * 254)
, 254)); "OK");

// Keep appending segments of 254 characters until we reach the end
i := 1;
local stringvar ls_return := uFLJsonGet(MyStringArray, "root[1].fruit", "STRINGOF", i,
"");
IF Len(ls_return) > 0 Then (
  IF Left(ls_return, 1) = "*" Then sResult := ls_return
  ELSE (
    while ls_return <> "" Do (
        ls_return := uFLJsonGet(MyStringArray, "mixture.arrayA[2].fruit", "STRINGOF", i,
"");
        sResult := sResult + ls_return;
        i := i + 1);
    sResult;);
```

Regular Expressions

In the following 3 functions, strInput() can be either a simple string or an array of strings that divides long text into an array with up to 254-charcter elements.
You can see example of how content can be chopped into such an array in the section discussing the HTML2Image() function.

uflRegExpIsMatch()

Arguments: (strInput(), Pattern)
This function takes an input string and matches it to Regular Expression pattern,

Returns: True if the input string matches the RegExp pattern.

For example, the following formula returns True because the string matches the RegExp pattern for valid emails:

```
uFLRegExpisMatch("ido@MilletSoftware.com",
  "[\w+-]+(?:\.[\w+-]+)*@[\w+-]+(?:\.[\w+-]+)*(?:\.[a-zA-Z]{2,4})")
```

uflRegExpMatch()

Arguments: (strInput(), Pattern)

Returns: This function searches the strInput and returns the first substring that matches the pattern.

For example, the following formula returns "ido@MilletSoftware.com" because it's the first substring matching a valid email address pattern:

```
uFLRegExpMatch("His email address is ido@MilletSoftware.com.",
                "[\w+-]+(?:\.[\w+-]+)*@[\w+-]+(?:\.[\w+-]+)*(?:\.[a-zA-Z]{2,4})")
```

uflRegExpReplace()

Arguments: (strInput(), Pattern, Replacement)

Returns: This function searches the strInput and replace strings that match a pattern with a replacement string.

For example, the following formula returns
"His Social Security Number is *-**-****"**
because every digit was replaced with a '*':

```
uFLRegExpReplace("His Social Security Number is 123-45-6789", "\d", "*")
```

File

uflFileAge()

Arguments: (FileName)
Returns: Age of file in **minutes**.
- 1 if the given file path & name doesn't exists.

uflFileAge2()

Arguments: (FileName, Type)
Type can be: '***CreationTime***', '***LastAccessTime***', or '***LastWriteTime***'
Returns: Age of file in **minutes**.
- **1** if the given file path & name doesn't exists.
- **10** if the Type argument is not recognized.
-100 for Unauthorized Access Exception
-200 for Path Too Long exception

uflFileCompare()

Arguments: (file1, file2)
Returns: False if the 2 files are not the same (or are missing/inaccessible).
Trues: if the content of the 2 files is the same.

uflFileCopy()

Arguments: (FileToCopy, DestinationFile)
Returns: "OK", "File Not Found", "Destination File Already Exists", or error message.
Note: use FileExists()/FileDelete() to ensure the destination file doesn't already exist.

uflFileDelete ()

Arguments: (FileToDelete)
Returns: "OK", "File Not Found", or Error message if the delete fails

uflFileExists()

Arguments: (FileName)
Returns: If the given file path & name exists, returns TRUE. Otherwise, returns FALSE.

uflFileRename()

Arguments: (FileToRename, NewName)
Returns: "OK", "File Not Found" or Error message if rename fails.
Note: use FileExists()/FileDelete() to ensure the new file name doesn't already exist.

uflFileJustName()

Arguments: (FileName)
Returns: file **name** without the path.

ufl**FileJustPath()**

Arguments: (FileName)
Returns: file **path** without the name.

ufl**FilePageCount()**

Arguments: (SourceFile, Options)
Returns: file **path** without the name.
Returns: the **number of pages** in the file **as a string**.
- or -
Error message (starting with ***)

SourceFile: the file path & name of the file
Options: leave blank ("")

NOTE: currently, **only pdf files** are supported.
A typical use case is to extract each pdf page as an image using uflFile2Image().
See video demo.

ufl**FileListFromWildCards()**

Arguments: (FileName(s), Delimiter, Segment_N, Recursive)

Notes:

- the FileName argument can include one or more path and wild card patterns separated by the specified Delimiter. The second element in such a delimited list can drop the path if it uses the path of the element before it. For example, **c:\Mail\Attach*.xls;*.pdf**
- If Recursive is set to True, the search process recurses down into subfolders

Returns: A delimited list of all files matching the specified FileName(s) patterns. Breaking the file list content into segments of 254 characters each, the function returns the segment number specified by the Segment_N argument.
If the specified file path & name doesn't exist, an empty string is returned.

In Crystal, you can load the entire file list into a string variable using the following code:

```
StringVar sFile;
NumberVar i;
i := 1;
// Keep appending segments of 254 characters to the sFile string until
// we reach the end of the file List
while FileListFromWildCards("c:\mail\attach\*.xls;*.sav",";", i ,False )  <>"" Do
(
sFile := sFile + FileListFromWildCards("c:\mail\attach\*.xls;*.sav",";", i ,False );
i := i + 1
);
// Return the resulting string
sFile;

// to show each file on a new line, replace delimiter with NewLine + CarriageReturn.
//Replace(sFile, ";", Chr(10) + Chr(13));
```

To check for the existence of files matching a wild card expression, you can use a Crystal formula like this:
IF Len(**FileListFromWildCards**("c:\temp*.zip", ",", 1, False)) > 0 Then True Else False

ufl**FileUnzip()**

Arguments: (FileToUnzip, DestinationFolder, Password, Type, Options)
Returns: "OK", "File Not Found", "ZIP Open Problem", "Password Problem" or error message.

If the **DestinationFolder** doesn't exist, the function takes care of creating it.
Password argument is optional. Set it to **""** if the zip file is not password protected.
Set the **Type** argument to **"ZIP"** (though currently it is assumed to be that)
Leave the **Options** argument as **""** (it is reserved for future enhancements).

Example without password:
uFLFileUnzip("c:\temp\test.zip", "c:\temp\", "", "ZIP", "");

Example with password:
uFLFileUnzip("c:\temp\test_Pass.zip", "c:\temp\", "abc123", "ZIP", "");

ufl**GetTempFolder()**

Arguments: None

Returns: The path to the user's temp folder (without a closing "\")

Text File

uflFileAddText()

Arguments: (FileName, TextToAdd(), DeleteFileBeforeAdd, CarriageReturnAfterAdd)
Returns: TRUE if successful, otherwise FALSE.

Adds text to a given file (file path and name).
If the file doesn't exist it gets created automatically.
If the directory doesn't exist, it gets created automatically.
Note: use **CHR(34)** in ***TextToAdd*** argument to generate **double quotes** in the text output.

TextToAdd() argument can be either simple string (up to 254 characters) or an array of such strings. You can see example of how long test can be chopped into such an array in the first example below. The array elements are combined to a single line in the target text file.

If **DeleteBeforeAdd** is TRUE – deletes the file before appending the text.
Note: the deleted file is moved to the recycle bin where it can be recovered.

If **CarriageReturnAfterAdd** is set to FALSE, follow-up calls to this function would add content to the same line (allowing "wide" exports beyond the 254 character limitation of String variables).
You can use the ***FileAddText()*** function to generate your own log or export files from within Crystal formulas.

A specific example would be a situation where after using the functions below to electronically burst and e-mail customers their invoices, you want to update the database with information about which invoices were emailed. You can write to a text file the invoice numbers that were included in the operation and use that file to update a Status column in the INVOICE table using an Update query (WHERE INVOICE_N IN the set of invoice numbers…).

An example of using the string array approach to write a very long line in a single call:

```
StringVar MyText := {@LongTextLine};
local stringvar array MyStringArray;
IF Len(MyText) = 0 Then
(redim MyStringArray [1];
 MyStringArray[1] = "";)
Else
(   Local numbervar segments := RoundUp(Len(MyText)/254);
    redim MyStringArray [segments];
    Local numbervar index ;
    for index := 0 to segments - 1 step 1 do
       (MyStringArray[index + 1] := mid(MyText, 1 + (index * 254) , 254)) ;
);

FileAddText (TargetFile, MyStringArray, False, True);
```

Here is an example of using the older approach (no string array for content) whereby the content is broken to multiple calls:

```
StringVar MyText := {@LargeText};
StringVar TargetFile := "c:\temp\test.txt";
// chop and write the text in 254 character segments
While Len(MyText) > 254 Do
   (
   FileAddText (TargetFile, Left(MyText, 254), False, False);
   MyText := Mid(MyText, 255);
   );
// write the remaining small segment
FileAddText (TargetFile, MyText, False, False);
```

Here is an example of using FileAddText to write to a log file:

```
// The FileAddText() can be used to create any customized text logging/export
// you can embed this functionality inside IF THEN logic to log information
// only when certain conditions in the report are met.

FileAddText(
// File to add text to
"c:\temp\My_Log.txt",
// Text added to the file
"Product Type: " + {Product_Type.Product Type Name} + ", " + Cstr(CurrentDateTime),
// FALSE = Don't delete this file before adding the text
FALSE,
// TRUE = add a CrLf at the end of the added text, so next call starts on new line.
TRUE);
```

ufl**FileAddText2()**

Arguments: (FileName, TextToAdd(), DeleteFileBeforeAdd, CarriageReturnAfterAdd, Encoding, EmitBOM, Options)

Returns: TRUE if successful, otherwise FALSE.

This function is same as uflFileAddTest() except for 3 additional arguments:

Encoding (string): any of the Character Encoding names listed here. For example, “utf-8”

EmitBOM (Boolean): If True, then the BOM (also known as a preamble), is emitted for charsets that define a BOM (such as “utf-8”, “utf-16”, and “utf-32”).

Options (String): set to blank text (""). This is reserved for future functionality.

ufl**FileAddTextKey()**

Arguments: (FileName, TextToAdd, DeleteFileBeforeAdd, CarriageReturnAfterAdd, **Key2AvoidDuplication**)

Returns: TRUE if successful, otherwise FALSE.
Same as FileAddText above but requires a unique String value in **Key2AvoidDuplication** argument. If that value was already used in a prior call within the same report preview, the process is skipped. The idea is to avoid duplication in cases where report pagination causes the formula evaluation to be duplicated.

Note: **FileAddTextKey()** is just a convenient alternative to using a **NewKey()** test before calling **FileAddText()**.

Note: use uflKeySetClear() to reset the keys between consecutive report previews.

ufl**LookupText()**

Arguments: (FileName, String2Match, ExactMatch)

The file name should include the **full path** to a text file with Key|||Value pairs like this:

```
Key1|||Value1
Key2|||Value2
…
```

Returns: the Value from the first line where the Key matches the Strings2Match argument.

ExactMatch is a Boolean argument: if it is True, the Key must match the String2Match on all characters (though the matching is not case sensitive). If ExactMatch is False, the process assumes a match if the key is found anywhere within the String2Match.

This function can be used to let end users maintain branching logic for a string formula without needing to change the formula within Crystal.

ufl**FileGetText()**

Arguments: (FileName, Segment_N)
Note: the file name should include the **full path** to the file.

Returns: Breaking the file content into segments of 254 characters each, the function returns the segment number specified by the Segment_N argument. If the specified file path & name doesn't exist, an empty string is returned.

In Crystal, you can load the entire file content into a string variable using the following code:

```
StringVar sFile;
NumberVar i;
i := 1;
// Keep appending segments of 254 characters to the sFile string until
// we reach the end of the file
while FileGetText( "c:\temp\test.ini", i) <>"" Do
(
sFile := sFile + FileGetText( "c:\temp\test.ini", i);
i := i + 1
);
// Return the resulting string
sFile;
```

Note: if the file contains HTML or RTF text (rather than plain text) you can take advantage of Crystal's formatting options to **interpret the string returned by the formula as HTML or RTF**. Also, be sure to use Crystal's "**Can Grow**" formatting option where appropriate.

ufl**FileGetTextUTF8()**

Same as FileGetText() except that it works with text files with **UTF-8 encoding**.

ini/registry

uflGetINIValue()

Arguments: (FileName, SectionName, KeyName)

Returns: The String value found in the ini file, under the given section for the specified key. Returns "Failed INI Lookup" if the lookup fails.

uflGetINIValueSegment()

Arguments: (FileName, SectionName, KeyName, SegmentN, FailedLookupMessage)

This function is just like GetINIValue() (see detail above) but it **allows you to retrieve strings that are longer than 254 characters by breaking the operation into segments**.

It also allows you to specify the string to return if the ini entry was not found.
"" (as shown in the examples below) would return blank text

Returns:
Breaking the resulting string into segments of 254 characters each, the function returns the segment number specified by the SegmentN argument. In Crystal, you load the entire result into a string variable using the following code:

```
WhilePrintingRecords;
StringVar sResult;
StringVar sSegment;
NumberVar i;
i := 1;
// Keep appending segments of 254 characters to the sResult string until reaching the end
sSegment := uflGetINIValueSegment("c:\test\text.ini","Options","Long_Greeting", I, "");
while sSegment <>"" Do
(
  sResult := sResult + sSegment;
  i := i + 1;
  sSegment := uflGetINIValueSegment("c:\test\text.ini","Options","Long_Greeting", I, "")
);
// Return the resulting string
sResult;
```

ufl**SetINIValue()**

Arguments: (FileName, SectionName, KeyName, KeyValue)

Returns: TRUE if Successful – FALSE if failed.

Writes the String value specified in "KeyValue" to the specified ini file, Section, and Key. If the path exists but the ini file doesn't – the function creates the ini file. If the section and/or key don't exist, they get created.

ufl**INISectionDelete()**

Arguments: (iniFile, SectionName2Delete)

Returns: "TRUE" if Successful
"***Missing INI File" if ini file wasn't found
Error message (starting with ***) if the process failed due to other issue

Deletes the specified section, and all its entries, from the specified ini file path & name.

ufl**GetRegistryString()**

Arguments: (Branch, Key_Name, Value_Name, Default)

Returns: The registry string value if it was found. If the registry value could not be found, the Default value is returned.

Note: possible Branch values are:

HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_CURRENT_CONFIG
HKEY_USERS

For example, the following Crystal formula:

```
GetRegistryString ("HKEY_CURRENT_USER",
                   "Environment", "Temp", "Not Found")
```

returns the following value on my PC:

%USERPROFILE%\AppData\Local\Temp

GMT/Local and Time-Related

uflGMTtoLocalMinutes()

Arguments: None

Returns: The a String providing the number of minutes between Local and Universal (GMT) time, taking into consideration Daylight Saving Time periods and the PC time zone setup. If the function fails to find the value, it returns "Failed"

For example, on my PC (Eastern Standard Time) GMTtoLocalMinutes() returns "240" or "300" depending on Daylight Saving Time.

uflGMTtoLocal()

Arguments: (GMTEpoch)

Returns: Local time as Epoch (number of seconds since 1970/01/01).
taking into consideration Daylight Saving Time periods and the PC time zone setup.

Web log files and databases frequently store DateTime information as Greenwich Mean Time (GMT) or Coordinated Universal Time (UTC). In your Crystal reports, you may need to display this DateTime information as Local Time.

Since UFL's don't support DateTime arguments, this function requires that you pass the GMT DateTime argument as Epoch (number of seconds since 1970/01/01). Assuming you have a GMT DateTime field called **{GMTDateTime}** you can convert it to Epoch using this formula:

DateDiff("s", datetime(1970,01,01), {GMTDateTime})

The function returns the Local Time as Epoch as well. Assuming the Formula returning the Local Time as Epoch is called {@LocalTimeEpoch} you can convert the Local Time result back to DateTime format using the following formula:

DateAdd("s", {@LocalTimeEpoch} , datetime(1970,01,01))

You can combine the conversion steps into a single formula such as:

DateAdd("s",
GMTtoLocal(DateDiff("s", datetime(1970,01,01), {GMTDateTime})) ,
datetime(1970,01,01))

Evaluating Report Processing Elapsed Time

Since Crystal evaluates ***CurrentDateTime*** only once for each report, you can't evaluate the time it took a report to process using Crystal functions.

If you pass a zero (or a negative number) to the **GMTToLocal()** function, it returns the current system date & time. This is **useful for timing report processing**.

Place the following formula in the report header to **capture the start time**:

```
WhilePrintingRecords;
DateTimeVar ldt_start;
ldt_start := DateAdd("s", GMTtoLocal(0), datetime(1970,01,01));
```

And place the following formula in the report footer to **display the elapsed time**:

```
WhilePrintingRecords;
DateTimeVar ldt_start;
DateDiff("s", ldt_start, DateAdd("s", GMTtoLocal(0), datetime(1970,01,01)));
```

ufl**GMTtoZone()**

Arguments: (GMTEpoch, ToZone)

Returns: Specified zone's time as Epoch (number of seconds since 1970/01/01).

This function is very similar to the ***GMTtoLocal()*** function described above except that instead of automatically detecting the local time zone on the user's PC, it converts the specified GMT/UTC time to time at the specified time zone.

Possible values for the ***ToZone*** argument are:

Time Zone Name	Offset
Tonga Standard Time	GMT+13:00
New Zealand Standard Time	GMT+12:00
Fiji Standard Time	GMT+12:00
Central Pacific Standard Time	GMT+11:00
E. Australia Standard Time	GMT+10:00
AUS Eastern Standard Time	GMT+10:00
West Pacific Standard Time	GMT+10:00
Tasmania Standard Time	GMT+10:00
Vladivostok Standard Time	GMT+10:00
Cen. Australia Standard Time	GMT+09:30
AUS Central Standard Time	GMT+09:30
Tokyo Standard Time	GMT+09:00
Korea Standard Time	GMT+09:00
Yakutsk Standard Time	GMT+09:00
China Standard Time	GMT+08:00
North Asia East Standard Time	GMT+08:00
Singapore Standard Time	GMT+08:00
W. Australia Standard Time	GMT+08:00
Taipei Standard Time	GMT+08:00
SE Asia Standard Time	GMT+07:00
North Asia Standard Time	GMT+07:00
Myanmar Standard Time	GMT+06:30
N. Central Asia Standard Time	GMT+06:00
Central Asia Standard Time	GMT+06:00
Sri Lanka Standard Time	GMT+06:00
Nepal Standard Time	GMT+05:45
India Standard Time	GMT+05:30
Ekaterinburg Standard Time	GMT+05:00
West Asia Standard Time	GMT+05:00
Afghanistan Standard Time	GMT+04:30
Arabian Standard Time	GMT+04:00
Caucasus Standard Time	GMT+04:00
Iran Standard Time	GMT+03:30
Arabic Standard Time	GMT+03:00
Arab Standard Time	GMT+03:00
Russian Standard Time	GMT+03:00

Time Zone Name	Offset
E. Africa Standard Time	GMT+03:00
GTB Standard Time	GMT+02:00
E. Europe Standard Time	GMT+02:00
Egypt Standard Time	GMT+02:00
South Africa Standard Time	GMT+02:00
FLE Standard Time	GMT+02:00
Israel Standard Time	GMT+02:00
W. Europe Standard Time	GMT+01:00
Central Europe Standard Time	GMT+01:00
Romance Standard Time	GMT+01:00
Central European Standard Time	GMT+01:00
W. Central Africa Standard Time	GMT+01:00
GMT Standard Time	GMT
Azores Standard Time	GMT-01:00
Cape Verde Standard Time	GMT-01:00
Mid-Atlantic Standard Time	GMT-02:00
E. South America Standard Time	GMT-03:00
SA Eastern Standard Time	GMT-03:00
Greenland Standard Time	GMT-03:00
Newfoundland Standard Time	GMT-03:30
Atlantic Standard Time	GMT-04:00
SA Western Standard Time	GMT-04:00
Pacific SA Standard Time	GMT-04:00
SA Pacific Standard Time	GMT-05:00
Eastern Standard Time	GMT-05:00
Central America Standard Time	GMT-06:00
Central Standard Time	GMT-06:00
Mexico Standard Time	GMT-06:00
Canada Central Standard Time	GMT-06:00
Mountain Standard Time	GMT-07:00
Pacific Standard Time	GMT-08:00
Alaskan Standard Time	GMT-09:00
Hawaiian Standard Time	GMT-10:00
Samoa Standard Time	GMT-11:00
Dateline Standard Time	GMT-12:00

Note: ***GMTtoZone()*** properly handles Daylight Saving Time periods at the specified zone.

uflSecondsToTimeString()

Arguments: (Seconds, Format)

Returns: seconds converted to a time string formatted according to the format argument.

For example,
uflSecondsToTimeString(3800, "HH:mm") returns "01:03"
uflSecondsToTimeString(3800, "HH:mm:ss") returns "01:03:20"

Note: use HH rather than hh in the time string in order to avoid cases where 0 hours is formatted as 12 hours.

uflTimeStringToSeconds()

Arguments: (TimeString, formatted as hh:mm:ss)

Returns: time string converted to seconds.

For example,
uflTimeStringToSeconds("01:03:20") returns 3,800.00

uflNumberToDate()

Arguments: (number reflecting date as 6 (yyMMdd) or 8 (yyyyMMdd) digits.

Returns: the corresponding Date.
If input is invalid, returns null date, which Crystal interprets as Dec 30, 1899.

For example,
uflNumberToDate (20210428) returns **4/28/2021**. This is **8-digit case**.
uflNumberToDate (210428) returns **4/28/2021**. This is **6 digit case**.
uflNumberToDate (20212804) returns **12/30/1899.** This is **invalid input case**.

Message/Input Boxes

ufl**MessageBoxOK()**

Arguments: (Message, Title)

This function triggers a message box with an OK button.

Returns: Ignore the return value

A typical scenario for using this functionality is to display a message in a runtime environment that doesn't support Crystal Report alerts.

For example, the following Crystal formula:
IF Sum ({Purchase.Net})> 100000 THEN **MessageBoxOK**("Net Amount Is Greater than $100,000" & Chr(10) & Chr(13) & "Please Contact the Authorities!"", "Alert: Net Amount is Too Big")
Triggers a message box if the grand total of the Net amount is more than 100,000.

ufl**MessageBoxYesNo()**

Arguments: (Message, Title)

This function triggers a message box with a YES and NO buttons.

Returns:
1 if the user clicked YES and
0 if the user clicked "NO"

A typical scenario for using this functionality is to condition further processing in the report on a user response, but only in specific situations (a regular parameter would prompt the user under all conditions).

For example, the following Crystal formula:

```
IF Sum ({Purchase.Net})> 100000 AND MessageBoxYesNo("Do you wish to email an alert to the appropriate manager?", "Alert: Net Amount is Too Big")=1 THEN
(
// the following code block would have detailed information causing an email to be triggered
EmailSet(…);
EmailAdd(…);
EmailSend;
);
```

Triggers an email message if the grand total of the Net amount is more than 100,000 and the user responded positively to the message box.

ufl**InputBox()**

Arguments: (Prompt, Title, DefaultText)

This function triggers an Input Box allowing the user to enter text.

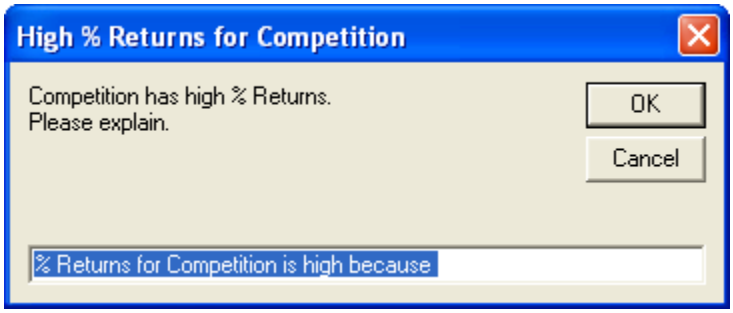
Returns:
Blank text if the user clicks Cancel
The user-entered text (up to 254 characters) if the user clicks OK

A typical scenario for using this functionality is to add comments to areas in the report that show exceptional (good/bad) performance. This is something that standard report parameters cannot do because:
a) parameters always get triggered while InputBox() can be conditionally triggered, and
b) parameters return fixed values, while InputBox() can be triggered multiple times (for example, once for each record or group with an exceptional value).

For example, the following Crystal formula:

```
IF {@L1_Returns}> 0.15 THEN
InputBox({Product_Type.Product Type Name} & " has high % Returns." & chr(13) &
"Please explain.", "High % Returns for " & {Product_Type.Product Type Name},
"% Returns for " & {Product_Type.Product Type Name} & " is high because ");
```

Triggers the following Input Box:



ufl**InputBox2Command()**

Arguments: (Prompt, Title, DefaultText, ExePath, CommandLine1, CommandLine2, CommandLine3, DebugWindow)

This function and the first 4 arguments behaves just like the InputBox function (described above). However, it is used to trigger another executable (just like the ExeRun function described above) and insert the user's input into the command line passed to the executable. The 3 command line arguments allow you to construct a command line that is longer than 254 characters (the function simply combines the 3 arguments into a single command line. The DebugWindow argument (True or False) allows you to request a display of the resulting command line (useful for debugging purposes).

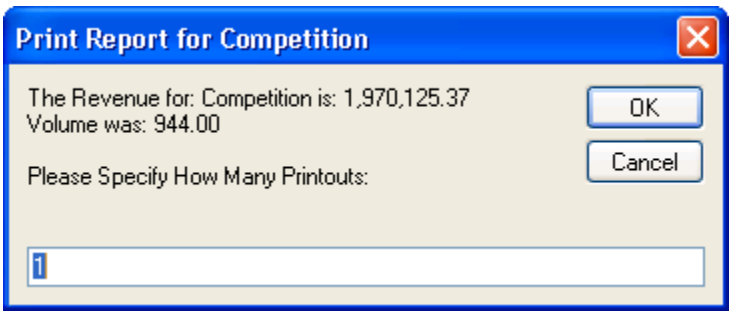
Returns:
Error message if the ExePath doesn't find an exe file in the specified location.
"No Input - Processing Skipped" if the user clicks Cancel or input was blank.
The user-entered text (up to 254 characters) if the user clicks OK

A possible scenario for using this functionality is to trigger printing of information on the report via a call to DataLink Viewer and another report, passing the product name as a parameter and the number of copies as an Input from the user.

For example, the following Crystal formula (placed in a Group Footer for Product Type):

```
InputBox2Command("The Revenue for: " & {Product_Type.Product Type Name}
& " is: " & Sum ({@value}, {Product_Type.Product Type Name}) & Chr(10) &
"Volume was: " & Sum({Orders_Detail.Quantity}, {Product_Type.Product Type
Name}) & Chr(10) & chr(10) & "Please Specify How Many Printouts:",
"Print Report for " & {Product_Type.Product Type Name},  "1",
"C:\Program Files\DataLink Viewer 9\DataLink_Viewer_9.exe",
"-v ""C:\Program Files\DataLink Viewer 9\Product Type Catalog V9.rpt""
""Parm1:" + {Product_Type.Product Type Name} + """",
" ""Printer:Default"" ""Print_Copies:{%Input}""", "", False);
```

Would prompt the user with the following dialog:



It would then replace the {%Input} token in the command line with the value provided by the user, so DataLink Viewer would print the information for that Product Type with the specified number of copies.

Web / HTML / Google / Translate

uflHTMLfile2RTFfile()

Arguments: (HTMLfile, RTFfile)

This function converts an HTML file (the 1st argument) to an RTF file (the 2nd argument).

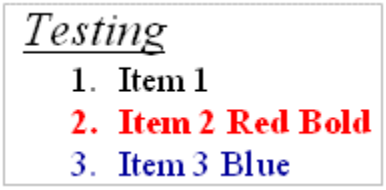
Returns:
"OK" or "Failed"

Since Crystal's support for RTF is more advanced than its support for HTML, a typical scenario for using this functionality is to display HTML files by converting them to RTF and using RTF rather than HTML interpretation for the formula.

For example, the following Crystal formula takes the CUT_Light.htm file, converts it to CUT_Light.rtf, and then uses the **FileGetText()** function to bring in the resulting RTF text.

```
uflHTMLfile2RTFFile ("c:\temp\CUT_Light.htm", "c:\temp\CUT_Light.rtf");

StringVar sFile;
NumberVar i;
i := 1;
// Keep appending segments of 254 characters to the sFile string until done
while uflFileGetText( "c:\temp\CUT_Light.rtf", i) <>"" Do
   (sFile := sFile + uflFileGetText( "c:\temp\CUT_Light.rtf", i);
i := i + 1);
// Return the resulting string
sFile;
```

The left display is HTML with a numbered list shown as RTF in Crystal.
The right display is the original HTML shown as HTML in Crystal.



Note that some aspects of the formatting are lost in the Crystal HTML interpretation:

For a description of RTF rendering limitations in Crystal Reports, see:
https://userapps.support.sap.com/sap/support/knowledge/en/1214798

For a description of HTML rendering limitations in Crystal Reports, see:
https://userapps.support.sap.com/sap/support/knowledge/en/1217084

uflHTMLstring2RTFFile()

Arguments: (HTMLstring, RTFFile)

This function converts an HTML string (the 1st argument) to an RTF file (the 2nd argument).

Returns: "OK" or "Failed"

Since Crystal's support for RTF is more advanced than its support for HTML, a typical scenario for using this functionality is to display HTML string (stored in a database column) by converting it to RTF and using RTF rather than HTML interpretation for the formula. Note: uflHTML2Image() provides a more powerful solution.

For example, the following Crystal formula takes the html string in {Customer.Comments}, converts it to CUT_Light.rtf, and then uses the **FileGetText()** function to bring in the resulting RTF text.

```
uflHTMLstring2RTFFile ({Customer.Comments}, "c:\temp\CUT_Light.rtf");

StringVar sFile;
NumberVar i;
i := 1;
// Keep appending segments of 254 characters to the sFile string until done
while uflFileGetTextutf8( "c:\temp\CUT_Light.rtf", i) <>"" Do
   (sFile := sFile + uflFileGetTextutf8( "c:\temp\CUT_Light.rtf", i);
i := i + 1);
// Return the resulting string
sFile;
```

The left display is HTML with a numbered list shown as RTF in Crystal.
The right display is the original HTML shown as HTML in Crystal.

Note that some aspects of the formatting are lost in the Crystal HTML interpretation:

For a description of RTF rendering limitations in Crystal Reports, see:
https://userapps.support.sap.com/sap/support/knowledge/en/1214798

For a description of HTML rendering limitations in Crystal Reports, see:
https://userapps.support.sap.com/sap/support/knowledge/en/1217084

uflHTMLString2Txtfile()

Arguments: (HTMLString(), Txtfile, Options)

This function converts an HTML string (the 1st argument) to a Text file (the 2nd argument).

Returns:
"OK" or "Failed"

If the HTML string is less the 254 characters, you can simply use it as the HTMLString argument. Otherwise, pass is as an arrays with elements of up to 254-charcter each. You can see example of how content can be chopped into such an array in the section discussing the HTML2Image() function.

The TxtFile argument should specify a path that is recognized by Microsoft Word as a trusted folder.

Leave the Options argument blank (""). It is set a side for future use.

This allows you to convert a large HTML string to plain text.
For example, the following Crystal formula takes a large HTML string, converts it to a text file, and then uses the **FileGetText()** function to bring in the resulting text file.

```
---------------------------------------
NumberVar i;
StringVar HTMLString;
HTMLString := {@HTML_String};

HTMLstring2TxtFile(HTMLString, "c:\temp\HTMLasText.txt", "");

// -- Get the Text File Content
StringVar sFile;
i := 1;
// Keep appending segments of 254 characters to the sFile string until done
while FileGetText( "c:\temp\HTMLasText.txt", i) <>"" Do
   (sFile := sFile + FileGetText( "c:\temp\HTMLasText.txt", i);
    i := i + 1);
// Return the resulting string
sFile;
---------------------------------------
```

ufl**httpFileExists()**

Arguments: (File URL)

This function checks if a specified file exists on the web.

Returns a String:

"No Response from Web Site"	if the web site didn't respond
"TRUE"	if the file exists
"FALSE"	if the file doesn't exist

Note: you may specify the path to the file in various ways:

Without http://
httpFileExists("www.MilletSoftware.com/Download/MyFile.zip")

With http:// or https://
httpFileExists("https://www.MilletSoftware.com/Download/MyFile.zip")

As FTP location
httpFileExists("ftp://ftp.cac.psu.edu/pub/thesis-packages/win/PsuThesiFull.exe")

ufl**httpExists()**

Arguments: (File URL)

This function checks if a specified url exists.

Returns a String:

"TRUE"	if the url exists and is accessible
"FALSE"	or http response Status Code otherwise

ufl**httpFileDownload()**

Arguments: (File URL, LocalFilePathName)

This function download a file on the web to a specified local file path and name.

Returns a String:

Error message	if the download process failed
OK	if the download succeeded

Note: you may specify the path to the web file in various ways, as described in httpFileExists above.

ufl**httpFileDownloadRename()**

Arguments: (File URL, LocalFilePathName, RenameToServerFileName)

This function is useful when
a) the remote file name is unknown (the url responds with a download of an unknown file)
In that case, use a temp file name in the 2nd argument, and set the 3rd argument to TRUE
- or -
b) when you wish to download a known file name to a different file name
In that case, set the 3rd argument to FALSE

Returns a String:

Error message	if the download process failed
Remote File Name	if the download succeeded

Note: you may specify the path to the web file in various ways, as described in httpFileExists above.

uflhttpFileParse()

Arguments: (URL, Targets(), Attribute, delimiter, Segment_N, Options, AuthInfo)

This function is useful when you need to extract text from a web page.
The **URL** argument sets the full path to the web page.
The **Targets()** argument can be a single string or an array of strings with either:
xPath directives for targeting certain element(s) within the web page. OR
From/To Token directives such as: `"[{From_Token}]->[{?To_Token}]"`
The **Attribute** path specifies the attribute within the target element that should be returned.
Use "InnerText" or "InnerHtml" to get those aspects of the targeted node.
The **delimiter** argument sets the separator between multiple parsing results.
The **Segment_N** argument allows handling of results longer than 254 characters.
The ***Options*** and ***AuthInfo*** arguments are not yet in use. Set to empty string ("").

Returns:
Error message (starting with ***) if the process failed.
Otherwise, breaking the resulting string into segments of 254 characters each, the function returns the segment number specified by the **Segment_N** argument. In Crystal, you can load the entire result into a string variable using code like the sample provided for uflExecuteSQLReturnDelimitedSegment().

Example 1: extract from a product catalog web page the path to the product's image.
The information resides in the following div:

```
<div class="product-image"><figure>
    <img src="/images/large/SS-400.jpg" alt="SS-100-1-2BT">
  </figure></div>
```

The following function call will return the text in yellow, using xPath directive, looking for div sections with a class of 'product-detail-image'. Within those, img nodes and their "src" attribute:

```
httpFileParse("https://www.acme.com/catalog?part=SS-400-1",
"//div[@class='product-image']//img", "src", "||", 1, "", "")
```

Alternatively, you can use this From/To token directive:

```
httpFileParse("https://www.acme.com/catalog?part=SS-400-1",
"[{data-src=""}]->[{?impolicy=superZoom}]", "src", "||", 1, "", "")
```

Note that a single literal " in the tokens must be escaped, becoming ""

Example 2: get hyperlink text for all anchor tags within li items within div marked as a class of 'row submenu':

```
httpFileParse("https://www.acme.com/catalog?part=SS-400-1",
"//div[@class='row submenu']//li//a", "InnerText","||", 1, "", "")
```

Web resources provide many tutorials & solutions for xPath logic.
Note: You can use httpFileExists() to check for existence of the web page or image files.

uflhttpToImage()

Use this function to download a web page as an image file. You can then bring the image into the report via dynamic reference to image path.

Arguments: (URL, LocalFilePathName, ImageType, ImageWidth, Wait4Load)

URL**:** can be specified in 3 ways (web url, file:///, or simple file path).
LocalFilePathName: the path and name for the generated image file.
ImageType can be "BPM", "JPG", or "PNG"
ImageWidth width of the browser page (in pixels) used to render. Set to zero (0) to ignore.
Wait4Load in **milliseconds**, allowing the web page to fully load before being captured as image. Set to zero (0) to ignore. For one use case, a value of 1500 worked well.

Returns a String:

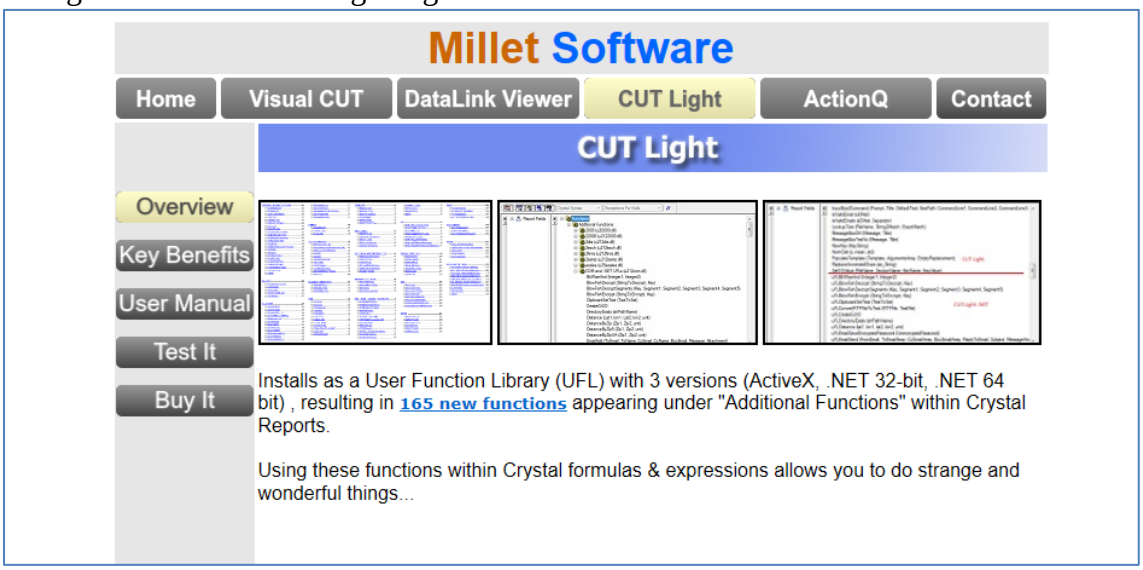
"Invalid URL"	if the url doesn't exist or gets redirected
Error message	if the process failed
"OK"	Otherwise

Note: you can use uflImageCrop() to crop or remove white space.

Web URL Example

uflhttpToImage('https://www.milletsoftware.com/CUT_Light.htm', 'C:\temp\dash.png', 'PNG', 955, 1500);

This generates the following image:



File URL Example

uflhttpToImage('file:///C:/TEMP/ Pivot.htm', 'C:\temp\ Pivot.JPG', 'JPG', 1045, 0);

Revenue & %Late

Year	(All)
Prod_Class	(All)
Prod_Type	(All)

Row Labels	Column Labels Suyama Revenue	% Late	King Revenue	% Late	Dodsworth Revenue	% Late	Leverling Revenue	% Late	Peacock Revenue	% Late	Davolio Revenue	% Late	Total Revenue	Total % Late
USA	$530K	10.9%	$499K	11.9%	$503K	8.4%	$485K	7.4%	$440K	12.1%	$427K	11.1%	$2,884K	10.4%
Canada	$48K	2.6%	$54K	8.9%	$31K	0.0%	$29K	20.0%	$37K	0.0%	$61K	11.4%	$259K	7.2%
Italy	$9K	0.0%	$28K	35.3%	$23K	0.0%	$18K	7.7%	$7K	0.0%	$36K	0.0%	$122K	7.4%
Spain	$20K	0.0%	$13K	0.0%	$12K	37.5%	$6K	0.0%	$14K	0.0%	$7K	16.7%	$72K	6.7%
Germany	$12K	0.0%	$7K	0.0%	$13K	0.0%	$10K	8.3%	$15K	9.1%	$13K	22.2%	$70K	6.8%
England	$6K	0.0%	$7K	33.3%	$10K	0.0%	$22K	0.0%	$19K	0.0%	$5K	0.0%	$69K	3.4%
Netherlands	$7K	0.0%	$24K	0.0%	$18K	0.0%	$1K	0.0%	$18K	0.0%	$0K	0.0%	$68K	0.0%
Grand Total	$632K	9.2%	$631K	11.7%	$609K	7.6%	$573K	7.7%	$550K	10.3%	$550K	10.7%	$3,545K	9.6%

Simple File Path Example

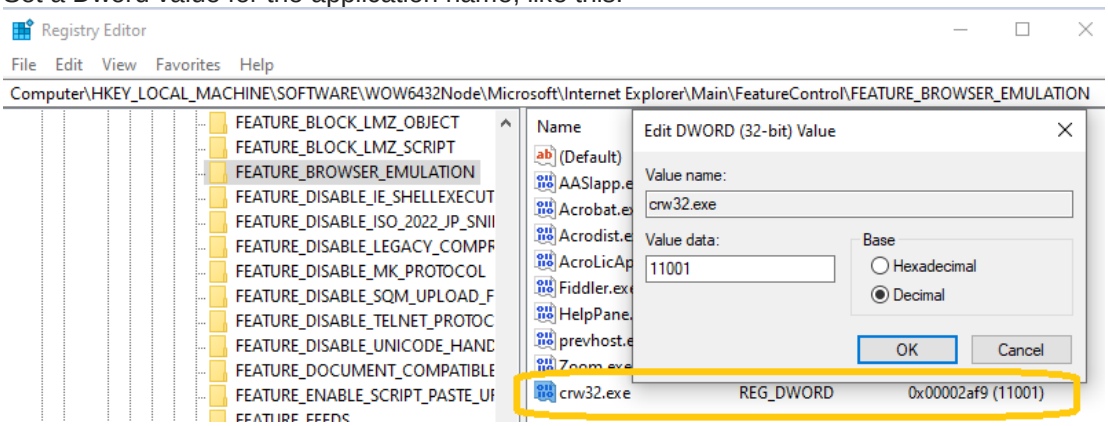
Same as above; CUT Light takes care of converting to file:///):
uflhttpToImage('C:\TEMP\Pivot.htm', 'C:\temp\ Pivot.JPG', 'JPG', 1045, 0);

Fixing 'Old Browser' error

For some websites, the image might show an error text indicating that 'You appear to be using a very old browser that doesn't allow our site to work properly'. The solution is to set a value in the registry to indicate the application should default to a later version of Browser Compatibility.
Warning: changing registry values should be done by an Administrator!

For a **32-bit application**, such as **Crystal 2016**, open:
Computer\HKEY_LOCAL_MACHINE\SOFTWARE**WOW6432Node**\Microsoft\Internet Explorer\Main\FeatureControl\FEATURE_BROWSER_EMULATION
For a **64-bit application** such as **Crystal 2020**, open:
Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Internet Explorer\Main\FeatureControl\FEATURE_BROWSER_EMULATION

Set a Dword value for the application name, like this:



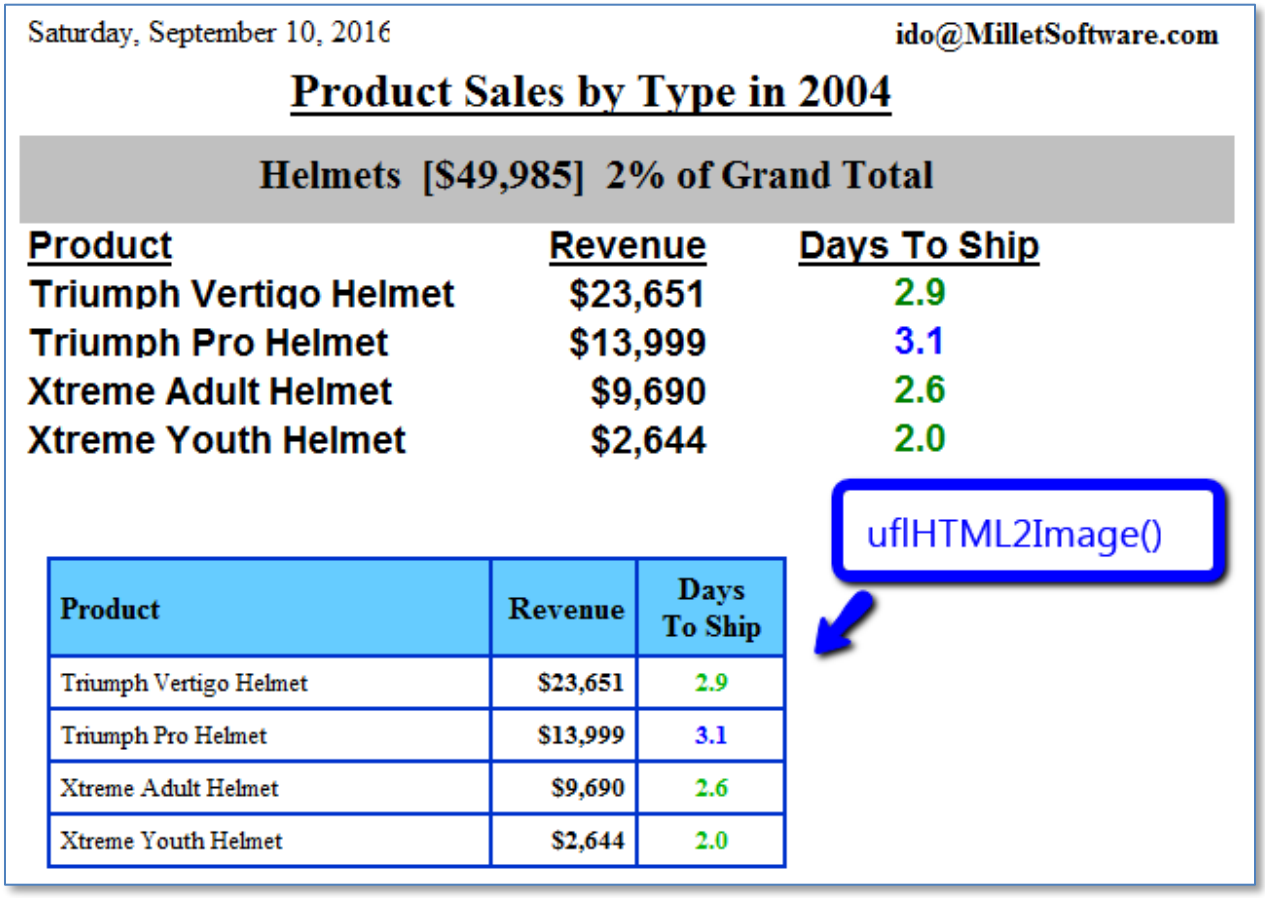
uflHTML2Image()

Arguments: (HTML(), LocalFilePathName, ImageType, ImageWidth)

Returns a String:

Error message	if the process failed
OK	Otherwise

This function overcomes limitations with how Crystal interprets HTML. Instead, it converts the HTML to an image so you can then load the image into a picture object using a dynamic Graphic Location expression:



LocalFilePathName specified the file path and name of the image file. Typically, the same path and file name is then used to set the dynamic Graphic Location path to load the image into the Crystal report.

ImageType can be "BPM", "JPG", or "PNG"

ImageWidth allows you to control the width (in pixels) of the browser page used to render the HTML content before it gets converted to an image. HTML elements specified as percent

of web page width would adapt to that width. Set to zero (0) to ignore.

HTML() argument divides the HTML string into an array with 254-charcter segments. In the example below, the HTML_Table string gets chopped into such an array using the code in red.

```
// Note: Content of HTML Table is established by other formulas
Stringvar HTML_Table ;
// Convert the HTML to an array with 254-character text segments
local stringvar array MyStringArray;
IF Len(HTML_Table) = 0 Then
(redim MyStringArray [1];
 MyStringArray[1] = "";)
Else
(   Local numbervar segments := RoundUp(Len(HTML_Table)/254);
    redim MyStringArray [segments];
    Local numbervar index ;
    for index := 0 to segments - 1 step 1 do
        (MyStringArray[index + 1] := mid(HTML_Table, 1 + (index * 254) , 254)) ;
);

uFLHTML2Image(MyStringArray, "c:\temp\" & {@ProductType} & ".bmp", "BMP", 400)
;

"c:\temp\" & {@ProductType} & ".bmp" ;   // Graphic Location set to the resulting image file
```

ufl**httpCallServiceGetTokens()**

Arguments: (URL, Tokens)

Calls a web service, such as SMS, passing argument values in the url. The Tokens argument (containing the names of the tokens separated by '||' allows the function to return the values of those tokens found in the XML returned from the web service.

Returns a '||' delimited string with the values of the named tokens
or Error message if failed

Example, for triggering a message via an SMS service:

```
uFLhttpCallServiceGetTokens("https://sveve.no/SMS/SendMessage?user=user1&passwd=shh&from
=TK-Helpdesk&reply=false&to=94271234&msg=Test%20of%20uflhttpCallServiceGetTokens()",
"msg_ok_count||id"
```

This call (if dummy password and phone number are replaced with real values) returns: 1||5554836
where 1 is the count of successful messages, and 5554836 is the id.

ufl**GoogleSentiment()**

Arguments: (sourceText(), contentType, Options)
Returns a String: with 2 numbers (***score*** and ***magnitude***) separated by '||' (e.g. -0.6||1.2)
Error message (starting with ***) if the process failed. Or just "||" is API key is restricted to certain IP addresses and the machine is not one of them.

See sample image showing a report with results for several text scenarios.

1. ***Score*** reflects overall sentiment ranging from -1 (super negative) to +1 (super positive).
2. ***Magnitude*** reflects total emotional content.

A neutral **score** (around 0.0) indicates a low-emotion document, or may indicate mixed emotions, with both high positive and negative values which cancel each out. Use **magnitude** values to tell the difference between these cases. **A truly neutral documents will have a low magnitude value, while mixed documents will have higher magnitude values**. For more detail about interpreting these numbers, see this Google documentation.

The Google service is free for the first 5,000 requests per month. If a request contains more than 1,000 characters, each 1,000 character segment counts as a request.
The cost for going beyond the free quota is very low and described here:
https://cloud.google.com/natural-language/pricing

The **language of the source text is auto-detected**. A list of supported languages is provided here: https://cloud.google.com/natural-language/docs/languages

To enable the service, you need to follow Google's Instructions to create a cloud project, enable billing, and get your API Key. You then need to set an entry in **CUT_Light_Options.ini.** For the 32-bit version of CUT Light, create that file here:
C:\Program Files (x86)\Millet Software\CUT_Light_NET_32\CUT_Light_Options.ini
The entry should look like this (use your own API key of course):

[Options]
GoogleAPI_NaturalLanguage=AIzaSyC4C22Af4XQ…
---

sourceText() divides the text you wish to translate into an array with 254-charcter segments. You can see example of how content can be chopped into such an array in the section discussing the HTML2Image() function.
As shown in the sample image, i**f the content is less the 254 characters, you can simply use it as-is without converting it to an array**.

contentType supports only two options: "HTML" or "Plain_Text"

Options: should be left blank (""). It is not yet used.

uflGoogleTranslate()

Arguments: (sourceText(), sourceLanguage, targetLanguage, Segment_N)
Returns a String:
Error message (starting with ***) if the process failed
The Nth segment of the translated text otherwise

You can translate any amount of text using the paid ($20 for 1 million characters) Google Translate service. This service supports more than 100 languages.
To enable the service, you need to follow Google's Instructions to create a cloud project, enable billing, and get your API Key. You then need to set an entry in **CUT_Light_Options.ini.** For the 32-bit version of CUT Light, create that file here:
C:\Program Files (x86)\Millet Software\CUT_Light_NET_32\CUT_Light_Options.ini
The entry should look like this (use your own API key of course):

[Options]
GoogleAPI_TranslateKey=AIzaSyC4C22Af4XQ…
No_Change=Millet Software||Cool||DataLink Viewer||Visual CUT

The **No_Change** entry indicates what text elements should be exempt from translation.

sourceText() divides the text into an array with 254-charcter segments. See how content is chopped into such an array in the section discussing the HTML2Image() function.
As shown in the example below, i**f the content is less the 254 characters, you can simply use it as that argument without converting it to an array**.

See example showing English text (in black) translated to Spanish (in blue), using the following formula:

```
---------------------
Local StringVar sResult;
Local StringVar sSegment;
Local NumberVar i;
i := 1;
// Keep appending segments of 254 characters to the sResult string until reaching the end
sSegment := uFLGoogleTraslate({Instructions.Description}, "English", "Spanish", i );
while sSegment <>"" AND Left(sSegment,3) <> "***" DO
(
sResult := sResult + sSegment;
i := i + 1;
sSegment := uFLGoogleTraslate({Instructions.Description}, "English", "Spanish", i );
);
// Return the resulting string
sResult;
---------------------
```

Note: the translation result is HTML-encoded (e.g. '&' -> &) so turn on HTML interpretation for the formula field: right-click, Format Field, Paragraph tab, set the Text Interpretation option to **HTML Text**.

ufl**Number2Arabic()**

Arguments: (**`InputNumber as Double`**, **CurrencyCode** as String)
Returns a String:
Error message (starting with ***) if the process failed
The number converted to Arabic words for specified currency otherwise. See image sample.

Allowed Currency Codes are: `"Syria", "UAE", "SaudiArabia", "Tunisia", "Qatar", "Kuwait", "Egypt, or "Gold"`

To express the fraction value as N/100, follow the currency name with **||V2** like this: "SaudiArabia**||V2**" See sample image.

For example, this formula:
uFLNumber2Arabic(186.87, "SaudiArabia")
Returns this:

فقط مائة وستة وثمانون ريالاً سعودياً و سبع وثمانون هللة لا غير.

And this formula:
uFLNumber2Arabic(186.87, "SaudiArabia**||V2**")
Returns this:

فقط مائة وستة وثمانون ريالاً سعودياً و 87/100 لا غير.

Notes:

- Since the first argument must be of Double data type, currency data types must be converted to double like this: **uFLNumber2Arabic**(**cDbl**({Orders.Order Amount}), "SaudiArabia")
- This functionality is done all locally – no need for internet connectivity.

uflIpDot2Long()

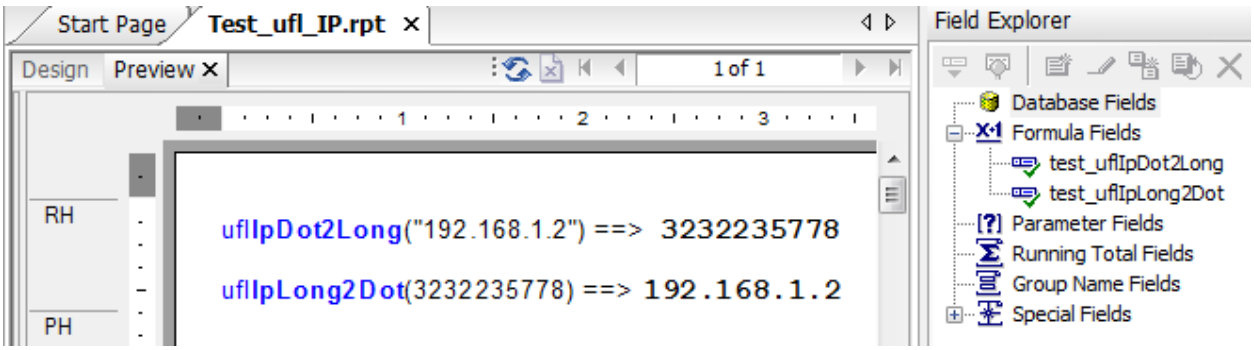
Arguments: (String)

Converts IP address such as "192.168.1.2" to its numeric value (3232235778)

uflIpLong2Dot()

Arguments: (Number)

Converts IP address number such as 3232235778 to its dot notation ("192.168.1.2")



uflPing()

Arguments: (***HostNameOrAddress*** as String, ***Options*** as String)
Note: Leave Options as blank ("")

Returns: If the host is accessible, returns ***TRUE***. Otherwise, returns ***FALSE***.

Examples:
uFLPing ("google.com", "") returns TRUE
uFLPing ("googllllle.com", "") returns FALSE
uFLPing ("74.125.67.100", "") returns TRUE

SQL

uflExecuteSQLCanConnect()

Arguments: (ODBC DSN or OLEDB Connection String, User ID, Password)

This function tests whether a connection can be made to a database: an **ODBC DSN** (even one that is not used by the Crystal report) or any **OLEDB Connection String**.

You may use such a test before calling other ExecuteSQL statements.

Returns:
"TRUE" or an error message if connection to the database failed.

Note: you may refer to a saved encrypted password by its name if you also own Visual CUT and you copy the entry from the DataLink_Viewer.ini file to the CUT_Light.ini file like this:
[Options]
Encrypted_Password_DB=64D26BF23E2C830AE2BE0A8EAC689353159D5618ED8D5D45

uflExecuteSQLNoReturn()

Note: uflSQLNoReturn() provides a newer function with similar functionality.

Arguments: (ODBC DSN or OLEDB Connection String,
User ID, Password, sql1(), sql2(), sql3(), sql4(), sql5())

This function executes a SQL statement against any **ODBC DSN** (even one that is not used by the Crystal report) or any **OLEDB Connection String**. If the SQL statement is longer than 254 characters, break it into segments across up to 5 sql "segments".
The idea is to Update, Delete, or Insert records as a byproduct of viewing a Crystal report.

Returns: "OK" or an error message if failed.

If the sql statement is not properly constructed (e.g., missing single or double quotes) you may get a "Memory Full" message from Crystal. In such a case, examine and adjust the sql statement.
sql1() –to- sql5() arguments can further divide very long sql statements into arrays with up to 254-charcter elements. You can see example of how content can be chopped into such an array in the section discussing the HTML2Image() function. **If the content is less the 254 characters, you can simply use it as that argument without converting it to an array**.

If you email me a request for a sample report, I will email you an rpt file that demonstrates this functionality. Below are two formulas from that report as examples you can follow:

The formula increments the "Last Year's Sales" column in the Customer table by $1:

```
ExecuteSQLNoReturn ("Xtreme Sample Database","","","Update
""Customer"" SET ""Last Year's Sales"" = ""Last Year's Sales""
+ 1 WHERE TRUE" ,"" ,"" ,"" ,"" )
```

This formula sets Customer 7 Address2 column to its Last Year's Sales:

```
ExecuteSQLNoReturn ("Xtreme Sample Database","","","Update
""Customer"" SET ""Address2"" = ""Last Year's Sales"" WHERE
""Customer ID"" = 7" ,"" ,"" ,"" ,"" )
```

This formula demonstrates using information from the current section in the report:

```
whileprintingrecords;
ExecuteSQLNoReturn ("Vision Offline","SYSDBA","sesame","Update
ENTRY SET PRINTED =1 WHERE ENTRY_ID =" +
cstr({ENTRY.ENTRY_ID},0,'') ,"" ,"" ,"" ,"" )
```

Embedding File(s) Content in the SQL Statement

If you need to trigger a large SQL script, you can embed within any of the sql segments references to files using the following token structure:

[[Insert_File:File_Path_and_Name]]

For example, **[[Insert_File:c:\temp\Script1.txt]]**

Such tokens are replaced with the content of the specified files (if such files exists). You can use **as many file tokens as you wish**. If an inserted file has embedded file tokens, they would be replaced as well (the process is **recursive**).

Avoiding Duplicate Processing (old approach)

Since Crystal may evaluate the same formula multiple times, as it renders the page content (particularly when Keep Together properties cause shifting of page content from one page to another), the SQL statements may fire more than once. This can be a problem in cases where an Update statement is incrementing a value.

To guard against duplicate processing, you can leverage CUT Light's ability to read and write ini file values. Here is an example:

```
If GetIniValue("c:\cutlight.ini","PickTicket.rpt",
                        {TKT_HDR.TKT_NO})="Failed INI Lookup"
Or
DateDiff ("s",
          CDateTime(GetIniValue("c:\cutlight.ini","PickTicket.rpt",
            {TKT_HDR.TKT_NO})), CurrentDateTime)>10
then
(
ExecuteSQLNoReturn ("ODBC1","","","Update DB1.dbo.TKT_HDR SET
DB1.dbo.TKT_HDR.TIMES_PRTD = DB1.dbo.PS_TKT_HDR.TIMES_PRTD + 1 WHERE
DB1.dbo.TKT_HDR.TKT_NO = '"+{TKT_HDR.TKT_NO}+"'" ,"" ,"" ,"" ,"" );

SetIniValue("c:\cutlight.ini","PickTicket.rpt",{TKT_HDR.TKT_NO},
       ToText(CurrentDateTime))
)
```

This formula starts by checking that a value for that particular record (ticket number) hasn't yet been written to the ini file or, if it has, it hasn't happened within the last 10 seconds. If that condition is satisfied, the formula then proceeds to execute the Update statenment and record the execution time for that particular ticket in the ini file.

Note: you may refer to a saved encrypted password by its name**. See information about this in the** ExecuteSQLCanConnect() **section.**

uflSQLNoReturn()

Arguments: (ConnectionType, ConnectionString, sql)

This function is a newer version similar to the older uflExecuteSQLNoReturn.
Differences include:

1. The code uses pure .NET implementation, removing the dependency on ADODB
2. SQL statement is passed in as a single string. No need to worry about the 254 character limitation.
3. Currently, the only ConnectionType supported is ODBC
4. If the result has more than one row or column, the function simply returns the value in the first row and first column (converted to string).

Returns:
The number of affected rows (as string) or an error message (starting with ***).

For Connection String examples, see: https://www.connectionstrings.com/

This SQL Server example updates an employee last name by adding '*' to it.
So 'Fuller" becomes 'Fuller*'. It returns the value '1' because only 1 row was affected.

```
uFLSQLNoReturn("ODBC", "Driver={SQL Server Native Client 11.0};
Server=REMOTE151; Initial Catalog=Northwind;Trusted_Connection=yes;",
"Update [dbo].[Employees] SET [dbo].[Employees].[LastName] = 'Fuller' WHERE
[dbo].[Employees].[EmployeeID]=2");
```

uflExecuteSQLReturnValue()

Note: uflSQLReturnValue() provides a newer function with similar functionality.

Arguments: (ODBC DSN or OLEDB Connection String,
User ID, Password, sql1, sql2, sql3, sql4, sql5)

This function executes a SQL statement against any **ODBC DSN** (even one that is not used by the Crystal report) or any **OLEDB Connection String**. If the SQL statement is longer than 254 characters, break it into segments across up to 5 sql "segments".
sql1() –to- sql5() arguments can further divide very long sql statements into arrays with up to 254-charcter elements. You can see example of how content can be chopped into such an array in the section discussing the HTML2Image() function. **If the content is less the 254 characters, you can simply use it as that argument without converting it to an array**.

The SQL statement must be of a type that returns a single value rather than a result set.
A typical use is to look up or get information from a data source that is not included in the report.

Returns:
A string containing the result, or an error message if failed. **You must guard against Null return values** by first checking for count or by using an aggregate (Min, Avg, Max…)

If the sql statement is not properly constructed (e.g., missing single or double quotes) you may get a "Memory Full" message from Crystal. In such a case, examine and adjust the sql statement.

If you email me a request for a sample report, I will email you an rpt file that demonstrates this functionality. Below is a formula from that report as an example you can follow:

The formula returns the number of employees in the EMPLOYEE table (even if that table or even the ODBC data source is not included in the report:

```
ExecuteSQLReturnValue ("Xtreme Sample Database","","","Select
Count(*) from Employee","","","","")
```

Here's another example:

```
ExecuteSQLReturnValue ("Xtreme Sample Database 11","","",
"SELECT sum(A.""Order Amount"") from Orders A WHERE
A.""Customer ID"" = " + Cstr({Orders.Customer ID},0,""),"","""
,"","","")
```

Here's an example that combines both types of SQL Functions:

```
WhilePrintingRecords;
IF ExecuteSQLReturnValue ("Vision
Offline","SYSDBA","myPass","Select PRINTED from ENTRY WHERE
ENTRY_ID ="+cstr({ENTRY.ENTRY_ID},0,'') ,"" ,"" ,"" ,"" ) ="0"

THEN
ExecuteSQLNoReturn ("Vision Offline","SYSDBA","myPass","Update
ENTRY SET PRINTED = 1 WHERE ENTRY_ID
="+cstr({ENTRY.ENTRY_ID},0,'') ,"" ,"" ,"" ,"" )
```

Example with a Connection String

Here's an example where instead of ODBC DSN, a connection string is specified:

```
ExecuteSQLReturnValue ("Provider=sqloledb;Data Source=IP
ADDRESS,1433;Network Library=DBMSSOCN;Initial
Catalog=DB_NAME;","USER","PASSWORD","Select Count(*) from
Location","","","","",""))
```

uflSQLReturnValue()

Arguments: (ConnectionType, ConnectionString, sql)

This function is a newer version similar to the older uflExecuteSQLReturnValue.
Differences include:

5. The code uses pure .NET implementation, removing the dependency on ADODB
6. SQL statement is passed in as a single string. No need to worry about the 254 character limitation.
7. Currently, the only ConnectionType supported is ODBC
8. If the result has more than one row or column, the function simply returns the value in the first row and first column (converted to string).

Returns:
The resulting value (as string) or an error message (starting with ***).
If the result of the SQL is zero rows, the error message is '***Zero Rows'

For Connection String examples, see: https://www.connectionstrings.com/

This **MS Access Example** returns the first employee's last name found:
```
SQLReturnValue("ODBC", "Driver={Microsoft Access Driver (*.mdb, *.accdb)}; Dbq=C:\Temp\Xtreme\xtreme.mdb;", "Select Employee.""Last Name"" from Employee");
```

This SQL Server example returns 'Fuller':
```
uFLSQLReturnValue("ODBC", "Driver={SQL Server Native Client 11.0}; Server=REMOTE151; Initial Catalog=Northwind;Trusted_Connection=yes;", "Select [dbo].[Employees].[LastName] from [dbo].[Employees] WHERE [dbo].[Employees].[EmployeeID]=2");
```

uflSQLReturnValueSegment()

Arguments: (ConnectionType, ConnectionString, [sql], SegmentN)

This function is similar to the previous one but the SegmentN argument allows you to retrieve very long values by fetching and combining 254-character segments:

1. ConnectionType must be set to "ODBC"
2. The code uses pure .NET implementation, removing the dependency on ADODB
3. SQL statement is passed in as a string array. No need to worry about the 254 character limitation.
4. If the result has more than one row or column, the function simply returns the value in the first row and first column (converted to string).

Returns:
Breaking the resulting string into segments of 254 characters each, the function returns the segment number specified by the Segment_N argument. In Crystal, you can load the entire result into a string variable using the following code:

```
WhilePrintingRecords;
Local StringVar sResult := "";
Local StringVar sSegment := "";
NumberVar i;
i := 1;
// Keep appending segments of 254 characters to the sResult string until reaching the end
sSegment := uflSQLReturnValueSegment ("ODBC","DSN=Xtreme;",
[{"Select Emp.""Last Name"" from Emp", "WHERE Emp.""Employee ID"" = 2"}], i );
while sSegment <>"" Do
(
sResult := sResult + sSegment;
i := i + 1;
sSegment := uflSQLReturnValueSegment ("ODBC","DSN=Xtreme;",
[{"Select Emp.""Last Name"" from Emp", "WHERE Emp.""Employee ID"" = 2"}], i );
);
// Return the resulting string
sResult;
```

Returns:
The resulting value (as string) or an error message (starting with ***).
If the result of the SQL is zero rows, the error message is '***Zero Rows'

For Connection String examples, see: https://www.connectionstrings.com/

uflExecuteSQLReturnFile()

Arguments: (ODBC DSN or OLEDB Connection String,
User ID, Password,
FilePathAndName, ConversionType,
sql1, sql2, sql3, sql4, sql5)

This function is similar to ExecuteSQLReturnValue() except for 2 aspects:

1. Instead of returning a value from the database to the report, the function saves the value To a file specified by the FilePathAndName argument.

2. The content is converted according to the ConversionType argument.
Supported options are:
None or "" (no conversion)
RTF2Text (convert RTF content to plain text)
2BMP, 2PNG, **2GIF**, or **2JPEG** convert any image type to bitmap, png, GIF, or jpeg
2BMP_White converts image with **transparent** background to BMP with **white** background

Returns:
"OK" or a failure message.

Note: you can bring the converted/saved content into the report using the **FileGetText**(), or **FileGetTextUTF8**() functions. In the case of image files, they can be brought into the report via dynamic reference to image path.

ufl**ExecuteSQLReturnDelimited()**

Arguments: (ODBC DSN or OLEDB Connection String,
User ID, Password, Delimiter, sql1, sql2, sql3, sql4, sql5)

This function executes a SQL statement against any **ODBC DSN** (even one that is not used by the Crystal report) or any **OLEDB Connection String**. If the SQL statement is longer than 254 characters, break it into segments across up to 5 sql "segments".

sql1() –to- sql5() arguments can further divide very long sql statements into arrays with up to 254-charcter elements. You can see example of how content can be chopped into such an array in the section discussing the HTML2Image() function. **If the content is less the 254 characters, you can simply use it as that argument without converting it to an array**.

The SQL statement can be of a type that returns multiple records. The function takes all the values in the first column of the result set and concatenates them into a single delimited string.

Returns:
A string containing the result, or an error message if failed.

If the sql statement is not properly constructed (e.g., missing brackets, single or double quotes) you may get a "Memory Full" message from Crystal. In such a case, examine and adjust the sql statement.

If you email me a request for a sample report, I will email you an rpt file that demonstrates this functionality. Below is a formula from that report as an example you can follow:

The formula returns the last names of all employees delimited with a semi-colon (";") from the EMPLOYEE table in the Xtreme Sample Database (even if that table or even the ODBC data source is not included in the report:

```
ExecuteSQLReturnDelimited ("Xtreme Sample Database","","",";",
"Select [Last Name] from Employee" ,"" ,"" ,"" ,"" )
```

The result is:

```
Davolio;Fuller;Leverling;Peacock;Buchanan;Suyama;King;Callahan;Dodsworth;Hellstern;
Smith;Patterson;Brid;Martin
```

Note: you may refer to a saved encrypted password by its name. See information about this in the ExecuteSQLCanConnect() section.

uflExecuteSQLReturnDelimitedSegment()

Arguments: (ODBC DSN or OLEDB Connection String,
User ID, Password, Delimiter, sql1, sql2, sql3, sql4, sql5, SegmentN)

This function executes a SQL statement against any **ODBC DSN** (even one that is not used by the Crystal report) or any **OLEDB Connection String**. If the SQL statement is longer than 254 characters, break it into segments across up to 5 sql "segments".

sql1() –to- sql5() arguments can further divide very long sql statements into arrays with up to 254-charcter elements. You can see example of how content can be chopped into such an array in the section discussing the HTML2Image() function. **If the content is less the 254 characters, you can simply use it as that argument without converting it to an array**.

The SQL statement can be of a type that returns multiple records. The function takes all the values in the first column of the result set and concatenates them into a single delimited string.

This function is just like ExecuteSQLReturnDelimited() (see detail in prior page), except that it **allows you to retrieve strings that are longer than 254 characters by breaking the operation into segments**.

Returns:
Breaking the resulting string into segments of 254 characters each, the function returns the segment number specified by the Segment_N argument. In Crystal, you can load the entire result into a string variable using the following code:

```
WhilePrintingRecords;
StringVar sResult;
StringVar sSegment;
NumberVar i;
i := 1;
// Keep appending segments of 254 characters to the sResult string until reaching the end
sSegment := ExecuteSQLReturnDelimitedSegment ("Xtreme Sample Database","","",
Chr(10) + chr(13), "Select [Customer Name] from Customer" ,"" ,"" ,"" ,"", i );
while sSegment <>"" Do
(
sResult := sResult + sSegment;
i := i + 1;
sSegment := ExecuteSQLReturnDelimitedSegment ("Xtreme Sample Database","","",
Chr(10) + chr(13), "Select [Customer Name] from Customer" ,"" ,"" ,"" ,"", i )
);
// Return the resulting string
sResult;
```

Note: if you email me a request for a **sample report**, I will email you an rpt file that

demonstrates this functionality.

Note: you may **refer to a saved encrypted password by its name**. See information about this in the **ExecuteSQLCanConnect()** section.

Totals of Totals

These functions overcome Crystal's inability to aggregate totals. For example, a report may display the Sum of Order Values by month within each year. CUT Light allows you to return aggregate calculations (such as Max, Min, and Rank) on top of these regular monthly sums:

PH		Sum	Lookup Sum	Rank	Max ofSum	Min ofSum	Count N
GF2	2004_01	$211,265.10	211,265.10	8	446,198.19	156,269.27	12
GF2	2004_02	$240,366.85	240,366.85	5	446,198.19	156,269.27	12
GF2	2004_03	$180,967.89	180,967.89	11	446,198.19	156,269.27	12
GF2	2004_04	$202,186.19	202,186.19	9	446,198.19	156,269.27	12
GF2	2004_05	$217,648.93	217,648.93	7	446,198.19	156,269.27	12
GF2	2004_06	$446,198.19	446,198.19	1	446,198.19	156,269.27	12
GF2	2004_07	$313,481.73	313,481.73	2	446,198.19	156,269.27	12
GF2	2004_08	$219,437.06	219,437.06	6	446,198.19	156,269.27	12
GF2	2004_09	$181,894.82	181,894.82	10	446,198.19	156,269.27	12
GF2	2004_10	$255,488.79	255,488.79	3	446,198.19	156,269.27	12
GF2	2004_11	$241,880.36	241,880.36	4	446,198.19	156,269.27	12
GF2	2004_12	$156,269.27	156,269.27	12	446,198.19	156,269.27	12
GF1a	2004	$2,867,085.18					
GF2	2005_01	$253,573.93	253,573.93	2	317,900.02	28,080.43	5
GF2	2005_02	$317,900.02	317,900.02	1	317,900.02	28,080.43	5
GF2	2005_03	$173,797.45	173,797.45	4	317,900.02	28,080.43	5
GF2	2005_04	$183,837.69	183,837.69	3	317,900.02	28,080.43	5
GF2	2005_05	$28,080.43	28,080.43	5	317,900.02	28,080.43	5
GF1a	2005	$957,189.52					

First, establish a unique key for an in-memory table storing the totals. For example:

```
BeforeReadingRecords;  Global Stringvar Key1 := uFLCreateGUID();
```

Then, Accumulate the totals using an in-memory table tied to that key. For example:

```
WhileReadingRecords;   Global Stringvar Key1;
    uFLTotalStore(Key1, "04", "12", "", "", "", 4, "");
    uFLTotalStore(Key1, "04", "12", "", "", "", 6, "");
    uFLTotalStore(Key1, "05", "01", "", "", "", 2, "");
    uFLTotalStore(Key1, "05", "02", "", "", "", 3, "");
    // resulting table: G1 | G2 | G3 | G4 | G5 | Sum | N | Avg | Max | Min
                        "04"  "12"                10    2    5      6      4
                        "05"  "01"                 2    1    2      2      2
                        "05"  "02"                 3    1    3      3      3
```

You can then Lookup a single stored total like this:

```
Global Stringvar Key1;
Cdbl(uFLTotalLookup(Key1, "04", "12",, "", "", "", "Sum", ""));   // resulting value: 10
```

To get a total of totals you can target all stored totals that match a particular pattern.
The example below returns the **maximum** of all **sums** stored for any row whose first key column is "05" (the '*' in the other four key columns match to anything).

```
WhilePrintingRecords;   Global Stringvar Key1;
cDbl(uFLTotalofTotals(Key1, "05", "*", "*", "*", "*", "Max", "Sum", ""));  // resulting value: 3
```

You can also get the **Percentile** values for a set of totals using **uflTotalPercentile**() and the Top/Bottom **Rank** of a given total using **uflTotalRank**().

The following sections describe these functions in more detail.

uflTotalStore()

Arguments:
TotalsKey : string uniquely identifying the in-memory totals table to create/use

G1,G2,G3,G4,G5 : strings for up to 5 Group Levels or Keys for identifying a total.
Can be left blank (**""**)

Value : the number (as Double data type) to accumulate into the total.

Options : blank string (""). Or, if set to **"[Distinct]"** calls with repeat keys are ignored.

Returns: a string with details about the created/updated total
or an error message starting with "*** "

For example, Formula in Report Header to establish a key for a totals table:

```
BeforeReadingRecords;   Global Stringvar Key1 := uFLCreateGUID();
// returns: '4841bd19-07d1-4902-95f7-e311d636b62e'
```

Formula in Detail section to accumulate totals for each year / month combination:

```
WhileReadingRecords;
Global Stringvar Key1;
uFLTotalStore(Key1, {@Year}, {@Month}, "", "", "", 4, "");
// returns: 'Inserted: [Sum4|N=1|Avg=4|Min=4|Max=4]'
```

uflTotalLookup()

Arguments:
TotalsKey : string uniquely identifying the in-memory totals table to lookup

G1,G2,G3,G4,G5 : strings for up to 5 Group Levels or Keys for identifying a total to lookup. Can be left blank (**""**)

Type : string specifying the total type. Acceptable types: **Sum**, **N**, **Avg**, **Max**, **Min**

Options : blank string (""). For future use.

Returns: a string showing the total or an error message starting with "*** "

Example:
```
Global Stringvar Key1;
Cdbl(uFLTotalLookup(Key1, {@Year}, {@Month}, "", "", "", "Sum", ""));
// returns: '4'
```

uflTotalsReset()

Arguments:
TotalsKey : key of totals table to reset and remove.

Options : blank string (""). For future use.

Returns: a string showing success or an error message starting with "*** "

Example:
```
Global Stringvar Key1;
uFLTotalsReset (Key1, "");
// returns: "TotalsKey [4841bd19-07d1-4902-95f7-e311d636b62e] was Reset."
```

uflTotalofTotals()

Arguments:
TotalsKey : key of totals table to scan for matching rows & total the specified summary.

G1,G2,G3,G4,G5 : strings for up to 5 Group Levels or Keys for identifying a total.
Use **"*"** as a wild card to match all values

Type : string specifying the aggregation operation: ***Sum***, ***Avg***, ***Count***, ***Max***, ***Min***, ***StDev***, or ***Var***

ofType : the column in the totals table to target for aggregation: **Sum**, **N**, **Avg**, **Max**, or **Min**

Options : blank string ("") or, to do a **Conditional Total of Totals**, specify a condition like this:
"[Condition:**N > 30**]" (that would total only total table rows that obey the condition)

Returns: a string showing the total of the total, or an error message starting with "*** "

Example:
```
Global Stringvar Key1;
Cdbl(uFLTotalofTotals (Key1, {@Year}, "*", "*", "*", "*", "Max", "Sum",  ""));
// returns: '3'
```

Notes:
When specifying the 5 key columns **You are not restricted to the grouping structure or the grouping hierarchy used in the report**. For example, in a report grouped by
a) {@Year}, and b) {@Month} you can ignore the year and find, for each month, the average sum across several years using something like:
```
Global Stringvar Key1;
Cdbl(uFLTotalofTotals (Key1, "*", {@Month}, "*", "*", "*", "Avg", "Sum",  ""));
```

The Condition can be a composite condition involving any of the columns in the table of totals.

uflTotalPercentile()

Returns the Percentile Value for a set of stored totals.

Arguments:
TotalsKey : key of totals table to scan for matching rows & total the specified summary.

G1,G2,G3,G4,G5 : strings for up to 5 Group Levels or Keys for identifying a total.
Use **"*"** as a wild card to match all values

ofType : the column in the totals table to target for comparison: **Sum**, **N**, **Avg**, **Max**, or **Min**

Percentile : for example, 75 would return the 75th percentile value

Options : blank string (""). For future use.

Returns: a string showing the targeted percentile value, or an error message starting with "*** "

Example:
Global Stringvar Key1;
uFLTotalRank(Key1, {@Year}, "*", "*", "*", "*", "Sum", 75, "")

PH		Sum	Lookup Sum	Rank	Max ofSum	Min ofSum	Count N
GF2	2004_01	$211,265	211,265.10	8	446,198	156,269	12
GF2	2004_02	$240,367	240,366.85	5	446,198	156,269	12
GF2	2004_03	$180,968	180,967.89	11	446,198	156,269	12
GF2	2004_04	$202,186	202,186.19	9	446,198	156,269	12
GF2	2004_05	$217,649	217,648.93	7	446,198	156,269	12
GF2	2004_06	$446,198	446,198.19	1	446,198	156,269	12
GF2	2004_07	$313,482	313,481.73	2	446,198	156,269	12
GF2	2004_08	$219,437	219,437.06	6	446,198	156,269	12
GF2	2004_09	$181,895	181,894.82	10	446,198	156,269	12
GF2	2004_10	$255,489	255,488.79	3	446,198	156,269	12
GF2	2004_11	$241,880	241,880.36	4	446,198	156,269	12
GF2	2004_12	$156,269	156,269.27	12	446,198	156,269	12
GF1a	2004		202,186.19 = 25th percentile of Monthly Sums 255,488.79 = 75th percentile				
GF2	2005_01	$253,574	253,573.93	2	317,900	28,080	5
GF2	2005_02	$317,900	317,900.02	1	317,900	28,080	5
GF2	2005_03	$173,797	173,797.45	4	317,900	28,080	5
GF2	2005_04	$183,838	183,837.69	3	317,900	28,080	5
GF2	2005_05	$28,080	28,080.43	5	317,900	28,080	5
GF1a	2005		176,307.51 = 25th percentile of Monthly Sums 301,818.50 = 75th percentile				

uflTotalRank()

Returns the Top or Bottom rank of a given total compared to a specified peer group of totals.

Arguments:
TotalsKey : key of totals table to scan for matching rows & total the specified summary.

G1,G2,G3,G4,G5 : strings for up to 5 Group Levels or Keys for identifying a total.
Use **"*"** as a wild card to match all values

Type : string specifying the Rank type: ***Rank_Top*** or ***Rank_Bottom***

ofType : the column in the totals table to target for comparison: **Sum**, **N**, **Avg**, **Max**, or **Min**

LookupTotal : the total value for which you want to return the rank.
You can use a regular total or uflTotalLookup() to provide that value.

Options : blank string (""). For future use.

Returns: a string showing the rank of the LookupTotal, or an error message starting with "*** "

Example:

```
Global Stringvar Key1;
uFLTotalRank(Key1, {@Year}, "*", "*", "*", "*", "Rank_Top", "Sum",
             CDbl(Sum ({Orders.Order Amount}, {@YY_MM})), "")
```

PH		Sum	Lookup Sum	Rank
GF2	2004_01	$211,265.10	211,265.10	8
GF2	2004_02	$240,366.85	240,366.85	5
GF2	2004_03	$180,967.89	180,967.89	11
GF2	2004_04	$202,186.19	202,186.19	9
GF2	2004_05	$217,648.93	217,648.93	7
GF2	2004_06	$446,198.19	446,198.19	1
GF2	2004_07	$313,481.73	313,481.73	2
GF2	2004_08	$219,437.06	219,437.06	6
GF2	2004_09	$181,894.82	181,894.82	10
GF2	2004_10	$255,488.79	255,488.79	3
GF2	2004_11	$241,880.36	241,880.36	4
GF2	2004_12	$156,269.27	156,269.27	12

uflTotalNth()

Returns details from the Nth largest or smallest stored totals row in a set of stored totals.

Arguments:
TotalsKey : key of totals table to scan for matching rows & total the specified summary.

G1,G2,G3,G4,G5 : strings for up to 5 Group Levels or Keys for identifying a total.
Use **"*"** as a wild card to match all values

N (integer) : for example, to retrieve details from 2nd highest Sum, you would set the N to **2**.

Type : string specifying ***Largest*** or ***Smallest***

ofType : the target column for Nth comparisons: **Sum**, **N**, **Avg**, **Max**, or **Min**

ReturnInfo : a string with token references to the desired detail from the Nth row.
The function replaces the token references with the values from the target row.
Tokens can be: *{G1}, {G2}, {G3},{G4},{G5}, {Sum}, {N}, {Avg}, {Max}, {Min}*
For example, **"{G1}|{G2}|{Sum}|Avg|"**

Options : blank string (""). For future use.

Returns: The requested ReturnInfo or an error message starting with "*** "

Example:
Global Stringvar Key1;
uFLTotalNth(Key1, {@Year}, "*", "*", "*", "*", "2", "Largest ", "Sum", "{G2}|{Sum}|", "")

2004_07|313481.73

The 2nd largest Sum in 2004 is: 313481.73
This occured in Month: 07

2004_01	$211,265.10	211,265.10	8
2004_02	$240,366.85	240,366.85	5
2004_03	$180,967.89	180,967.89	11
2004_04	$202,186.19	202,186.19	9
2004_05	$217,648.93	217,648.93	7
2004_06	$446,198.19	446,198.19	1
2004_07	$313,481.73	313,481.73	2
2004_08	$219,437.06	219,437.06	6
2004_09	$181,894.82	181,894.82	10
2004_10	$255,488.79	255,488.79	3
2004_11	$241,880.36	241,880.36	4
2004_12	$156,269.27	156,269.27	12

Geo

uflDistance()

Arguments: (lat1, lon1, lat2, lon2, unit_of_measure)
lat1, lon1 are the latitude and longitude of point 1 (in decimal degrees)
lat2, lon2 are the latitude and longitude of point 2 (in decimal degrees)
unit_of_measure is how the result should be provided:
'm' for miles, 'k' for kilometers, 'n' for nautical miles

This function calculates the distance between two points (given the latitude/longitude of those points). Note that south latitudes are negative, east longitudes are positive.

Returns: Distance in the requested units of measure.

Example: **Distance (52.2047, 0.1406 , 53.2047 , 0.1406, "m")** returns **69.09**

uflDistanceByZip5()

Arguments: (Zip1, Zip2, unit_of_measure)
Zip1 and Zip2 are the **Canadian** or **5-digit** US zip code of the two points (e.g., "M1B0A9" and "16509") unit_of_measure is how the result should be provided:
'm' for miles, 'k' for kilometers, 'n' for nautical miles

Returns: Distance in the requested units of measure.

Example: **DistanceByZip5 ("16509", "M1B 0A9", 'm')** returns **4.76** miles

Note: this function requires that the CUT_Light.ini file is installed to the default location of: c:\Program Files\CUT Light\ Or c:\Program Files (x86)\CUT Light\
Since it uses a local ini file, it doesn't depend on a web connection and quota restrictions imposed by DistanceByZip().

uflDistanceByZipUK()

Arguments: (Zip1, Zip2, unit_of_measure)
Zip1 and Zip2 are the UK Outer Codes of the two points (e.g., "AB16" and "AB30")
unit_of_measure is how the result should be provided:
'm' for miles, 'k' for kilometers, 'n' for nautical miles
Returns: Distance in the requested units of measure.

Example: **DistanceByZipUK("AB16", "AB30, "k")** returns **48.92 Kilometers**

Note: this function requires that the CUT_Light.ini file is installed to the default location of: c:\Program Files\CUT Light\ Or c:\Program Files (x86)\CUT Light\

Since it uses a local ini file, it doesn't depend on a web connection and quota restrictions imposed by DistanceByZip().

ufl**DistanceByZip()**

Arguments: (Zip1, Zip2, unit_of_measure)
Zip1 and Zip2 are the zip code of the two points (e.g., "16563" or "16563-1400")
unit_of_measure is how the result should be provided:
'm' for miles, 'k' for kilometers, 'n' for nautical miles

Returns: Distance in the requested units of measure.

Example: **Distance ("16509", "16563-11400, "m")**
returns **5.76** (distance in miles between the two points)

Note: this function receives data from a web site, so it requires the computer to have access to the internet. If the function stops performing, you probably exceeded the daily hit quota for the web site. You should switch to the **DistanceByZip5()** function which uses a local data file but is restricted to Canadian or 5-digit US zip codes.

ufl**GetLatLongFromZip()**

Arguments: (Zip): zip code (e.g., "16563" or "16563-1400")

Returns: Latitude/Longitude string.

Example: GetLatLongFromZip("16563-1400")
returns **42.124621/-79.982133**

Note: this function receives data from a web site, so it requires the computer to have access to the internet. If the function stops performing, you probably exceeded the daily hit quota for the web site. You should switch to the **GetLatLongFromZip5()** function, which uses a local data file but is restricted to Canadian or 5-digit US zip codes.

ufl**GetLatLongFromZip5()**

Arguments: (Zip): **Canadian** or **5-digit US** codes ("16563" or "M1B0A9" or "M1B 0A9")

Returns: Latitude/Longitude string.

Example: GetLatLongFromZip5("M1B 0A9")
returns: **43.807304/-79.179753**

Note: this function requires that the CUT_Light.ini file is installed to the default location of:
c:\Program Files\CUT Light\ Or c:\Program Files (x86)\CUT Light\
Since it uses a local ini file, it doesn't depend on a web connection and quota restrictions imposed by GetLatLongFromZip().

ufl**GoogleAddress2LatLong()**

Arguments: (Address, ApiKey)

Returns: Latitude/Longitude string. Error message (starting with ***) if the process failed

Example: **uFLGoogleGeoAddress2LatLong**("5275 Rome Ct, Erie, PA 16509, USA","")
returns: **42.0961719||-80.0294165**

Download sample report. Or see image.

You need to follow Google's Instructions to create a cloud project, enable billing, and get your **Maps API Key**. You then need to either specify the ApiKey via the argument or, more typically, leave the ApiKey argument blank ("") and set the API key via an entry in **CUT_Light_Options.ini.**
For the 32-bit version of CUT Light, create that file here:
C:\Program Files (x86)\Millet Software\CUT_Light_NET_32\CUT_Light_Options.ini
The entry should look like this (use your own API key of course):

[Options]
GoogleAPI_Geo=AIzaSyDNR6…

ufl**GoogleDrivingTimeDistance()**

Arguments: (Origin, Destination, Units, ApiKey)
Origin/Destination are specified as address or Lat|Long pair ("42.0961719||-80.0294165","")
Units: "Metric" to return distance in **meters**. "Imperial" to return distance in **feet**.

Returns: a string with 4 elements delimited by '||':
1. Driving time (seconds) assuming we depart now (takes into account traffic conditions)
2. Driving time as nicely formatted text
3. Distance in feet if Units argument was set to "Imperial", meters if "Metric".
4. Distance as nicely formatted text
Returns error message (starting with ***) if the process failed

Example: **uFLGoogleDrivingTimeDistance**("5275 Rome Ct, Erie, PA 16509, USA",
"While House, Washington, DC", "Imperial","")
returns: 20405||5 hours 40 mins||594538||369 mi

Download sample report. Or see image.

You need to follow Google's Instructions to create a cloud project, enable billing, and get your **Distance Matrix API Key**. Specify the ApiKey via the argument or set the API key via an entry in **CUT_Light_Options.ini** like this (use your own API key of course):

[Options]
GoogleAPI_Distance=AIzaSyBcu…

Excel

uflGetXLSValue()

Arguments: (Workbook, Worksheet, Cell)

Returns: The text of the excel cell value
Returns "Workbook Not Found" if the file can't be found.
Returns "Worksheet Not Found" if the worksheet can't be found

Example: GetXLSValue **("c:\temp\test.xlsx", "Sales", "B2")**

Notes:
1. If you leave the sheet name blank ("") or as "1" the first sheet in the workbook is used.

uflSetXLSValue()

Arguments: (Workbook, Worksheet, Cell, NewValue)

Returns: OK or error message.
Returns "Workbook Not Found" if the file can't be found.
Returns "Worksheet Not Found" if the worksheet can't be found

Example: SetXLSValue **("c:\temp\test.xlsx", "Sales", "B2", "MyNewValue")**

Notes:
1. If you leave the sheet name blank ("") or as "1" the first sheet in the workbook is used.
2. NewValue is specified inside double quotes, but treated as if typed into the cell, so "100.2" would become a number, while "'100.2" would become text in the spreadsheet due to the single quote.

uflGetXLSOutput()

This function allows you to plug Crystal values as input to an Excel model and get some output back. One typical use scenario is VLOOKUP tables, allowing users to maintain lookup logic in Excel without needing to modify the Crystal report.

Arguments: (Workbook, Worksheet, InputCell, InputCellValue, OutputCell)

Returns: The text of the excel OutputCell after plugging the InputCellValue into InputCell
Returns "Workbook Not Found" if the file can't be found.
Returns "Worksheet Not Found" if the worksheet can't be found

Example: GetXLSOutput **("c:\temp\test.xlsx", "", "B2", {@LookUpValue}, "C4")**

Notes:
1. If you leave the sheet name blank ("") or as "1" the first sheet in the workbook is used.

uflXLSLookUp()

This function allows you to look up a value in one column and return the corresponding value (from the same row) in another column.
The advantages of using this newer function compared to uflGetXLSOutput are:

a) Doesn't require Excel to be installed on the same machine
b) Much faster
c) Doesn't require a VLOOKUP() or similar logic within the spreadsheet

Arguments: (Workbook, Worksheet, Lookup Value, Lookup Column, ReturnFrom Column)
note: all arguments are of String type, even if the value in excel is numeric

Returns: The text of the value found in the corresponding column.
Or "LookUp Value Not Found" if the lookup value is not found within the Lookup Column.
Example: GetXLSLookup**("c:\temp\test.xlsx", "Emails", "7882", "C", "Q")**

Notes: the 'Lookup column' doesn't need to be to the left of the 'ReturnFrom Column'.

uflXlsGetValue()

The advantages of using this newer function compared to uflGetXLSValue() are:

a) Doesn't require Excel to be installed on the same machine
b) Using the SegmentN argument allows you to retrieve values longer than 254 characters
c) The array arguments allow you to get values from multiple sheets & cells

Arguments: (Workbook, SheetName(), Cell(), SegmentN)
note: The SheetName() and Cell() arrays allow you to get values from multiple sheets and cells. The values are delimited by '||' .
If you are targeting a single sheet/cell combination, pass them in as simple string arguments.

Returns: The cell values, delimited by '||' or an error message.

Example: **uFLXLSGetValue("C:\Temp\MyData.xlsx", ["Sheet1","Sheet1"], ["A1","A2"], 1);**
this returns '422||TestA22' which are the delimited values found in Sheet1, cell A1 & A2.

In this case, we know the content would not exceed 254 characters, so we pass in a SegmentN value of 1. If we need to handle longer returned values, we can loop following the logic discussed for the uflGetINIValueSegment() function.

uflXlsSetValue()

The advantages of using this newer function compared to uflSetXLSValue() are:

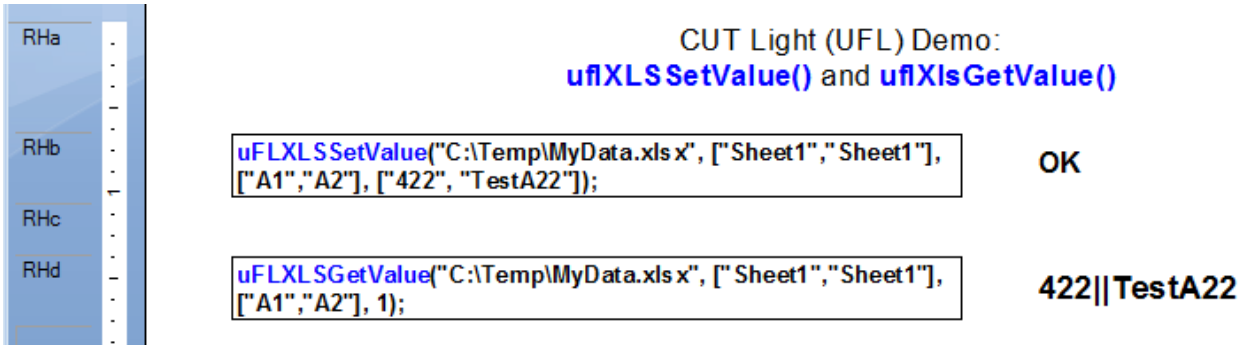
a) Doesn't require Excel to be installed on the same machine
b) Can handle protected sheets
c) The array arguments allow you to set values in multiple sheets & cells in a single pass

Arguments: (Workbook, SheetName(), Cell(), NewValue())
note: The SheetName(),Cell(), and NewValue() arrays allow you to set values to multiple sheets and cells.
If you are targeting a single sheet/cell combination, pass them in as simple string arguments.

Returns: "OK" or error message.

Example: **uFLXLSSetValue("C:\Temp\MyData.xlsx", ["Sheet1","Sheet1"], ["A1","A2"], ["422","TestA22"]);**

Here is a sample report image showing both setting and getting multiple Excel cell values:



Font

ufl**GetTextWidth()**

Arguments: (Text, FontName, FontSize, Bold, Italic, Units)

Returns: The width of the text in specified Units ("Twips" or "Pixels")
-1 if an error occurs

Note: 1440 twips = 1 inch

Examples: GetTextWidth({Employee.Last Name}, "Arial", 10, False, False, "Pixels")
GetTextWidth({Employee.Last Name}, "Arial", 10, False, False, "Twips")

ufl**GetFontSizeToFitText()**

Arguments: (Text, FontName, Available Width, Bold, Italic, Units)
The Units argument reflects the units in which Available Width was specified:
"Inches", "Centimeters", "Twips", or "Pixels".

Returns: The largest font size that would fit the specified text within the available width.
-1 if an error occurs

Example: **GetFontSizeToFitText**("This is Some Text", "Arial", 2, False, False, "Inches")
returns a value of 18 (so font size of 18 is the largest that would fit that text in 2 Inches)

Note: typically used to dynamically control the font size of a field/formula in Crystal.

ufl**GetTextHeight()**

Arguments: (Text, MaxWidth, FontName, FontSize, Bold, Italic)

Returns: The height of the wrapped text in inches, when fitted into the MaxWidth (in inches)
-1 if an error occurs

Examples: GetTextHeight ({@SomeText}, 3.5, "Arial", 10, False, False)

ufl**GetTextNumberOfLines()**

Same as GetTextHeight(), but returns **number of wrapped lines** rather than text height.

Encryption

uflBlowFishEncrypt()

Arguments: (StringToEncrypt, **Key**)

Returns: The encrypted text (as HEX) using the BlowFish algorithm.

For example: **BlowFishEncrypt ("MilletSoftware", "Sesame")**
Returns:
268BA82DCA546F2C4E107E9C8AF16546318A8E275BDE0F8D1C5BBD663C8DF10E

Note: StringtoEncrypt can't exceed 254 characters. If you try to pass a string larger than that, the function returns: ""String to Encrypt Must Be Less Than 255 Characters"

Note however that the returned encrypted string may be longer than 254 characters

uflBlowFishDecrypt()

Arguments: (StringToDecrypt, **Key**)

Returns: The decrypted text using the BlowFish algorithm.

For example:
BlowFishDecrypt ("268BA82DCA546F2C4E107E9C8AF16546318A8E275BDE0F8D1C5BBD663C8DF10E", "Sesame")
Returns:
MilletSoftware

Note: StringtoDecrypt can't exceed 254 characters. If you try to pass a string larger than that, the function returns: ""String to Decrypt Must Be Less Than 255 Characters"
If the string you need to decrypt is longer, use **BlowFishDecryptSegment()** (see next page).

ufl**BlowFishDecryptSegment()**

Arguments: (**Key,** Segment1, Segment2, Segment3, Segment4, Segment5)
Note: Key is the first argument in this function (unlike the prior function).

Returns: The result (as HEX) of decrypting the combined text of all segments using the BlowFish algorithm.

For example: **BlowFishEncryptSegment ("Sesame", "MilletSoftware", "", "", "","")**
Returns:
268BA82DCA546F2C4E107E9C8AF16546318A8E275BDE0F8D1C5BBD663C8DF10E

This function is like **BlowFishDecrypt**() but it **allows you to break the text you need to decrypt into up to 5 segments that are each no longer than 254 characters**.
Here is a Crystal formula sample that automatically breaks a string to such segments

```
StringVar Plain_String := "Long Plain Text or a reference to a Crystal field/formula";
StringVar Encrypted_String := BlowFishEncrypt(Plain_String, "Sesame");
StringVar Decrypted_String;

IF Len(Encrypted_String) <=  254 Then
      Decrypted_String := BlowFishDecryptSegments("Sesame",
      Encrypted_String, "", "", "", "")
Else If Len(Encrypted_String) <= 254 * 2 Then
      Decrypted_String := BlowFishDecryptSegments("Sesame",
      Left(Encrypted_String, 254),
      Mid(Encrypted_String, 254 + 1), "", "", "")
Else If Len(Encrypted_String) <= 254 * 3 Then
      Decrypted_String := BlowFishDecryptSegments("Sesame",
      Left(Encrypted_String, 254),
      Mid(Encrypted_String, 254 + 1, 254),
      Mid(Encrypted_String, (254 * 2) + 1), "", "")
Else If Len(Encrypted_String) <= 254 * 4 Then
      Decrypted_String := BlowFishDecryptSegments("Sesame",
      Left(Encrypted_String, 254),
      Mid(Encrypted_String, 254 + 1, 254),
      Mid(Encrypted_String, (254 * 2) + 1, 254),
      Mid(Encrypted_String, (254 * 3) + 1), "")
Else If Len(Encrypted_String) <= 254 * 5 Then
      Decrypted_String := BlowFishDecryptSegments("Sesame",
      Left(Encrypted_String, 254), Mid(Encrypted_String, 254 + 1, 254),
      Mid(Encrypted_String, (254 * 2) + 1, 254),
      Mid(Encrypted_String, (254 * 3) + 1, 254),
      Mid(Encrypted_String, (254 * 4) + 1))
else Decrypted_String := "Encrypted String is Too Long";
```

Email

uflEmailSetOptionsIniFile()

Arguments: full path and name of the ini file storing processing options.

For example:

SetOptionsIniFile("C:\ProgramData\MilletSoftware\VC_11\DataLink_Viewer.ini")

Returns: `"OK"` if successful, failure message otherwise.

You may point to an existing DataLink Viewer.ini file used by Visual CUT or you may first call this function when the ini file doesn't exist yet. If the ini file doesn't exist, the function automatically creates it and all the entries shown below. you can then:
a) use Notepad to update entries, and
b) call **uflEmailSaveEncryptedPassword**() to encrypt and save the email authentication password (if your SMTP server requires password authentication).

[Options]
Email_Default_SMTP_Server=???
Email_SMTP_Port=???
Email_User_ID=
// the following entry is set only by calling **uflEmailSavePasswordEncrypted()**
Email_Encrypted_Password_CUT_Light= E2484C7AB94FD7C525F177D…
Email_StartTLS=False

Email_Connect_Timeout=40
Email_Connect_Retries=4
Email_Message_Timeout=60
Email_Failure_Notices_To=
Email_Bounce_Address=
Email_Send_Encrypted=False
Email_Send_Signed=False
Email_Outgoing_Folder=
Email_SMTP_Domain=
Email_SMTP_Disconnect_After_Send=True
Email_SocksHostname=
Email_SocksPort=
Email_SocksUsername=
Email_SocksPassword=
Email_SocksVersion=
Check_Email_Addresses=TRUE
Email_POP3_Server=

---

uflSetEmailSaveEncryptedPassword()

Arguments: the unencrypted password.

For example:

SetEmailSaveEncryptedPassword("MyPassword")

Returns: `"OK"` if successful, failure message otherwise.

When successful, the encrypted password is saved to the **Email_Encrypted_Password_CUT_Light** entry in the [Options] section of the ini file established by a prior call to **EmailSetOptionsIniFile**().

While you must call **EmailSetOptionsIniFile**() everytime you wish to send email, you need to call **SetEmailSaveEncryptedPassword**() only when the password changes.

uflEmailSend()

Arguments: (FromEmail, ToEmailArray, CcEmailArray, BccEmailArray, ReplyToEmail, Subject, MessageArray, AttachmentArray, Priority, LogFile)

Returns: `"OK"` if successful, failure message otherwise.

The array arguments allow you to specify multiple email addresses and email attachments.

LogFile must be the path and file name of an existing text file.
The MessageArray allows you to divide a very long text or HTML message into 254-charcter segments. Here is an example:

```
Local StringVar Message := {@emailMessage};
local stringvar array MessageArray;
IF Len(Message) = 0 Then
(
   redim Messagearray [1];
   MessageArray[1] = "";
)
Else
(
   Local numbervar segments := RoundUp(Len(Message)/254);
   redim Messagearray [segments];
   Local numbervar index ;
   for index := 0 to segments - 1 step 1 do
      (
      MessageArray[index + 1] := mid(Message, 1 + (index * 254) , 254)
      );
);
```

Full Example

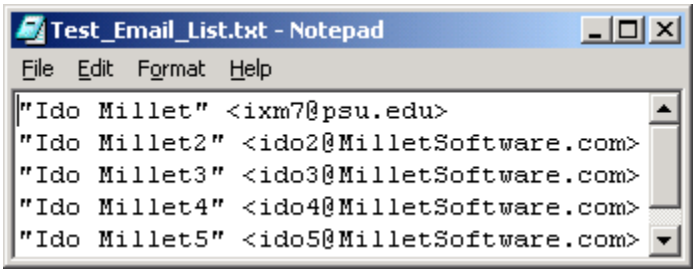
```
Local StringVar Message := {@MyEmailMessage};
// Convert the message to an array with 254-character text segments
local stringvar array MessageArray;
IF Len(Message) = 0 Then
(redim Messagearray [1];
 MessageArray[1] = "";)
Else
(   Local numbervar segments := RoundUp(Len(Message)/254);
    redim Messagearray [segments];
    Local numbervar index ;
    for index := 0 to segments - 1 step 1 do
        (MessageArray[index + 1] := mid(Message, 1 + (index * 254) , 254));
);
// ------------------------------
// Point to the ini file where the email options are stored
uFLEmailSetOptionsIniFile("C:\ProgramData\MilletSoftware\VC_11\DataLink_Viewer.i
ni");

// This is needed (only once) if SMTP server requires a password
uFLEmailSaveEncryptedPassword("Your Secret Password");

uFLEmailSend(""""From Name"" <ido@MilletSoftware.com>",
          """To Name"" <joe@Acme.com>", "", "", "", // to, cc, bcc, ReplyTo
          "My Email Subject", MessageArray, {@Attachments},
          "Normal", // Priority: "Lowest"/"Low"/"Normal"/"High"/"Highest"
          "");  // log file (path & name of an existing text file) if you wish to log email
activity
```

Specifying Multiple (Simple/Composite) Email Addresses

You specify multiple email address destinations in the **To**, **CC**, or **BCC** emailing options, by simply including multiple entries in the array arguments and/or by separating multiple addresses with a **semi-colon (;)** without any spaces.
You can also specify composite (display name as well as address) email destinations.
For example: **"Ido Millet" <ido@MilletSoftware.com>;"Jane Doe" <Jane@aol.com>**

Specifying Email Distribution Lists in Text Files

To facilitate emailing to a long list of recipients, you can specify in the **To**, **CC**, and **BCC** emailing options a **file name containing a distribution list**.
For example, using Notepad you can create a file such as:



You then specify that Visual CUT should use that file as a distribution list by entering
File:c:\temp\test_email_list.txt
in the **To**, **CC**, or **BCC** emailing options.

Notes: you use the word "**File:**" followed by the path & name of the text file containing the email addresses. **Each address should be on a separate line** and it can be specified as an **email address only or as a composite** (display name and address) as shown in the example above.

Specifying Email Distribution Lists in SQL Queries

In cases where you wish to dynamically retrieve the list of email addresses using an SQL query against an ODBC data source, you can specify in the **To**, **CC**, and/or **BCC** emailing options an expression such as:

MS Access Example:

```
ODBC:Customers::User_ID::Password::SELECT [email] FROM
[Contacts] WHERE [Customer_ID] = '{Customer_ID}'
```

SQL Server Example:

ODBC:CONTACTS::sa::xxxxx::SELECT DISTINCT AHD.ctct.c_email_addr FROM AHD.ctct where AHD.ctct.c_email_addr IS NOT NULL

The expression starts with **`ODBC:`** followed by 4 elements separated by **`::`**

1. **`ODBC DSN`** (Note: could be different from the DSN used for the Crystal report)
2. **`User ID`** (use a single space if not needed)
3. **`Password`** (use a single space if not needed)
4. **`SQL Statement`**

Notes:

- If the SQL statement returns multiple rows, Visual CUT takes care of concatenating the email addresses from all the rows to a single string with semi-colon as the delimiter.
- The SQL statement syntax depends on your database. For example, as shown in the samples above, MS Access uses [ ] around field/table names but SQL Server doesn't.
- You can embed dynamic fields/formula values anywhere inside the expression. In the MS Access example, the list of contacts for the Customer in the current bursting cycle would be retrieved. This offers powerful email distribution management options...

Attaching Multiple Files

To attach multiple files to a single email message, simply include multiple entries in the AttachmentArray argument or specify multiple files and separate them with a **semi-colon (;) Without any spaces.**

If all files are under the same folder, it's enough to specify the full path only for the 1st file.
For example: **c:\temp\Sales.pdf;Returns.pdf**

You can also specify file names using **wild cards**.
For example, in a bursting scenario, to send all pdf files in the current month folder under the current customer:
C:\VC_Exports\{customer.customer_id}\{@Month}*.pdf
and to send all files that start with the current Customer ID:
C:\VC_Exports\{customer.customer_id}*.*

Specifying a Different Character Set

In order to override the default character set (iso-8859-1) you should add an entry to the [Options] section of DataLink_Viewer.ini. For example, in order to support Chinese characters, you should add the following entry:

```
Email_Char_Set=big5
```

Queuing Emails & The smtpQ Service

If you are also using Visual CUT, by specifying an outgoing folder (Email_Outgoing_Folder) in the ini file, you can direct outgoing email messages (as eml files) to an outgoing folder. For more detail, see the Visual CUT user manual.

ufl**IsValidEmail()**

Arguments: (email address)

Returns:
True if the email address looks valid. False otherwise.

Note: the logic only ensures correct structure and no invalid characters. It doesn't check the domain and doesn't test whether the email address actually exists.

ufl**IsValidEmails()**

Arguments: (email addresses separated by a separator, The separator)

Returns:
True if all email addresses looks valid. False otherwise.

Note: the logic only ensures correct structure and no invalid characters. It doesn't check the domain and doesn't test whether the email address actually exists.

Example: Isvalidemails("**ido@MilletSoftware.com;ixm7@psu.edu**", "**;**")
Returns True

Legacy Functions

These functions provide substitutes for old UFLs such as ***uflssm.dll***, ***ufldate.dll***, and ***uflbtime.dll***. This allows reports using these functions to run in Crystal 2020 and the 64-bit Crystal runtime.

The function names are the same as in their legacy UFLs except that in CUT Light they are prefixed with '**fl**'. You can use Visual CUT to automate the process of replacing the original name with the same name prefixed with fl. See video demo.

flHoursFromSecsSinceMidnight()

Arguments: SecsSinceMidnight as integer (number of seconds since midnight)
Returns: the **hours** portion after converting the argument to time.
Example: `flHoursFromSecsSinceMidnight(50585);` ➔ `14`
`note: legacy function from uflssm.dll`

flMinutesFromSecsSinceMidnight()

Arguments: SecsSinceMidnight as integer (number of seconds since midnight)
Returns: the **minutes** portion after converting the argument to time.
Example: `flMinutesFromSecsSinceMidnight(50585)` ➔ `3`
`note: legacy function from uflssm.dll`

flSecondsFromSecsSinceMidnight()

Arguments: SecsSinceMidnight as integer (number of seconds since midnight)
Returns: the **seconds** portion after converting the argument to time.
Example: `flSecondsFromSecsSinceMidnight(50585)` ➔ `5`
`note: legacy function from uflssm.dll`

flSHHMMFromSecsSinceMidnight()

Arguments: SecsSinceMidnight as integer (number of seconds since midnight)
Returns: string showing the time formatted as **HH:MM**.
Example: `flHHMMSecondsFromSecsSinceMidnight(50585)` ➔ `02:03`
`note: legacy function from uflssm.dll`

flHHMMPMFromSecsSinceMidnight()

Arguments: SecsSinceMidnight as integer (number of seconds since midnight)
Returns: string showing the time formatted as **HH:MM** AM/PM
Example: `flHHMMPMSecondsFromSecsSinceMidnight(50585)` ➔ `02:03 PM`
`note: legacy function from uflssm.dll`

flHHMMSSFromSecsSinceMidnight()

Arguments: SecsSinceMidnight as integer (number of seconds since midnight)
Returns: string showing the time formatted as **HH:MM:SS**
Example: `flHHMMSSSecondsFromSecsSinceMidnight(50585)` ➔ `02:03:05`
`note: legacy function from uflssm.dll`

flHHMMSSPMFromSecsSinceMidnight()

Arguments: SecsSinceMidnight as integer (number of seconds since midnight)
Returns: string showing the time formatted as **HH:MM:SS AM/PM**
Example: `flHHMMSSSecondsFromSecsSinceMidnight(50585)` ➔ `02:03:05 PM`
`note: legacy function from uflssm.dll`

flDayOfYear()

Arguments: myDate as Date
Returns: a number representing the **day of year** (1-366)
Example: `flDayOfYear(Date(2022,9,14))` ➔ `257`
`note: legacy function from ufldate.dll`

flWeekOfYear()

Arguments: myDate as Date
Returns: the **week number** as integer
Example: `flWeekOfYear(Date(2022,9,14))` ➔ `38`
`note: legacy function from ufldate.dll`

flTIME()

Arguments: Fraction of a day (24 hours) as a Number
Returns: string showing the time formatted as **HH:mm:ss**
Example: `flBTIME(0.66)` ➔ `15:50:24`
`note: legacy Btrieve function from uflbtime.dll`

Update History

Version 6.4.9159 (October 5, 2025):

- uflImageCrop() supports **[Threshold:?]** option to control what color levels (1-255) are considered white enough to be auto-cropped.
- uflImageResize() can now save the resized image over the original one.
- Added uflImageFix() to Orient, automatically DeSkew (un-rotate), and DeSpeckle (remove noise from) images.
- Updated components.

Version 6.4.9150 (January 23, 2025):

- Fixed failure to accept ***Var*** (Variance) as summary operation in Totals of Totals functions.

Version 6.4.9149 (December 15, 2024):

- Functions that download a file from a web server, now avoid getting the file from the cache.
- Added uflActiveDirectoryGetProperty() function for getting property values for a given user.
- Fixed a rare, recently-introduced, failure to reload the UFL after closing the runtime.

Version 6.4.9142 (September 19, 2024):

- Added uflFilePageCount() function for getting the number of pages in a pdf file. You can use this function -- along with uflFile2Image() -- to **display a multi-page pdf as dynamically generated/imported page images in a Crystal report**. See video demo.
- Updated internal component.

Version 6.4.9139 (July9, 2024):

- Added uflBBarcodeIMb() function for generating USPS Intelligent Mail barcodes.

Version 6.4.9138 (May 31, 2024):

- Internal code update to optimize performance and update code protection.

Version 6.4.9137 (May 18, 2024):

- Added a **[Circular:<FrameColor>]** option to uflImageCrop() allowing you to generate circular images framed in a desired color.

Version 6.4.9136 (April 25, 2024):

- Added uflSQLReturnValueSegment() allowing retrieval of a long string using a more modern connection method.

Version 6.4.9124 (December 5, 2023):

- Updated logic of uflPopulateTemplate() to better handle cases where a token is null.

Version 6.4.9133 (October 10, 2023):

- Added a **[Center]** option to uflImageResize() allowing you to center a proportionally resized image within the specified width & height. This stops Crystal from distorting the image when '***Can Grow***' is False.

Version 6.4.9131 (August 6, 2023):

- Updated uflhttpToImage() to avoid 'Invalid URL' results due to JavaScript errors
- Added instructions for Fixing 'Old Browser' error when using uflhttpToImage()

Version 6.4.9127 (July 12, 2023):

- Fixed text display for some Application Identifiers (AI codes) in uflBBarcodeGS1()

Version 6.4.9127 (June 23, 2023):

- Added support for doing Conditional Totals of Totals in uflTotalofTotals()
- Added uflBBarcodeGS1() to **generate GS1-128 barcodes**.
- Included sample report for barcodes with no dependency on a database.
- Sparkline charts now better support negative values.

Version 6.4.9123 (April 8, 2023):

- Fixed a bug and improved performance for uflReplaceAccentedChars()
- Fixed a bug in sizing of overlay logos on QR Codes.
- Updated Excel component.
- Optimized performance for several functions.

Version 6.4.9117 (November 30, 2022):

- Added uflImageReorient() function to allow rotating and flipping images.

Version 6.4.9117 (November 30, 2022):

- Added 10 functions from old UFLs such as ***uflssm.dll***, ***ufldate.dll***, and ***uflbtime.dll***. This allows reports using these functions to run in 32-bit as well as 64-bit (e.g. Crystal 2020) applications. See Legacy Functions
- Updated an internal component.

Version 6.4.9114 (September 9, 2022):

- Fixed a bug in uflNumberToDate().

Version 6.4.9112 (July 2, 2022):

- Added uflJsonGet() function to retrieve JSON content using a specified path.

Version 6.4.9106 (May 12, 2022):

- Added uflPing() function to check access to remote machine.
- Added uflSleep() function to inject a delay (e.g. retry a connection).
- Language translation (via Google Translate) now supports 109 languages.

Version 6.4.9105 (April 18, 2022):

- Added Decode2ImageFile() to convert Base64 or Hexadecimal content to an image file for dynamic loading into a report (via the 'Graphic Location' expression of a dummy image).
- Fixed a recently introduced bug in uflImageCrop().
- uflBulletChart() now supports a ***[ReverseColors]*** option to treat lower numbers as good.

Version 6.4.9103 (March 29, 2022):

- Improved Trim logic for functions that might return errors longer than 254 characters.

Version 6.4.9102 (February 19, 2022):

- Added uflFormatValue() to convert Inches or Millimeters to imperial format.
  For example, a value of **71.6** Inches becomes **5' 11 5/8"**

Version 6.4.9101 (February 8, 2022):

- Fixed a bug in uflhttpFileExists().

Version 6.4.9098 (January 11, 2022):

- Added a From/To tokens alternative to uflhttpFileParse() .

Version 6.4.9098 (December 3, 2021):

- Added uflZatcaEncode() to support QR Codes for electronic ZATCA invoices.
- Added 'Hex' encoding options (for string as well as integers) to uflEncode().

Version 6.4.9095 (October 3, 2021):

- Added 7 functions to support Totals of Totals.
- uflBBarcodeQR() now support adding a logo.
- uflBBarcodeQR() now support auto-cropping to avoid padding.
- uflImageCrop() now supports auto-cropping (e.g. remove padding from QR Barcodes).

Version 6.4.9092 (August 6, 2021):

- uFLRegExpisMatch(), uFLRegExpMatch() and uflRegExpReplace() now accept as input either a single string or an array of strings. This avoids a string length limitation.
- Packaged with a newer component.

Version 6.4.9090 (May 17, 2021):

- Added uflNumberToDate() function.

Version 6.4.9087 (April 23, 2021):

- uflImageResize() now supports "[Reorient_EXIF]" option to reorient the image based on embedded EXIF code.
- uflImageResize() now handles 0, 0 arguments for width & height to indicate no size change.

Version 6.4.9085 (March 23, 2021):
- Added uflImageColorRemap() function to change colors in images and barcodes.
- uflImageResize() now supports "[AvoidAntiAliasing]" option.
- uflEncode() now supports "Base64" encoding.
- Added uflDecode() to decode Base64 or Hexadecimal content.

Version 6.4.9079 (January 7, 2021):
- Added uflhttpFileParse() function to extract information from web pages.

Version 6.4.9078 (August 16, 2020):
- Added uflFileAddText2() function to provide control over encoding and emitting BOM.

Version 6.4.9077 (July 2, 2020):
- uflFileAddText() and uflFileAddTextKey functions can now accept a string array as the text input. This avoids multiple calls when writing very long lines.

Version 6.4.9076 (July 1, 2020):
- uflExecuteSQLNoReturn(), uflSQLNoReturn(), uflSQLReturnValue(), uflExecuteSQLReturnFile(), uflExecuteSQLReturnDelimited(), and uflExecuteSQLReturnDelimitedSegment() functions can now accept a string array for each of their sql1, sql2, sql3, sql4, sql5 arguments. This accommodates extremely long sql statements.

Version 6.4.9074 (May15, 2020):
- Packaged with a newer component.

Version 6.4.9072 (April 6, 2020):
- Added uflFileAge2() to allow more options for type of age (CreationTime, LastAccessTime, or LastWriteTime).
- Packaged with several newer components.

Version 6.4.9069 (12/20/2019):
- Added uflFile2Image() to allow insertion of PDF files as resized images into reports.

Version 6.4.9068 (11/23/2019):
- Added uflEncode() to provide barcode128 encoding functionality

Version 6.4.9067 (5/16/2019):
- To allow dynamic setting of the Google API key, added a new required argument to the **GoogleDrivingTimeDistance()** and **uFLGoogleGeoAddress2LatLong()** functions. If you upgrade to this version, and you wish to keep using the ini file entry to set the API key, simply add a blank argument like this:
uFLGoogleGeoAddress2LatLong("5275 Rome Ct, Erie, PA 16509, USA", "")

uFLGoogleDrivingTimeDistance(Origin, Destination, "imperial", "");

Version 6.4.9066 (3/18/2019):

- Added uflXLSGetValue() and uflXLSSetValue() as enhanced versions of the older uflGetXlsValue and uflSetXLSValue(). These functions allow getting and setting multiple cell values in a single pass. They also allow getting values longer than 254 characters.

Version 6.4.9064 (1/24/2019):

- Added uflGetINIValueSegment() to allow retrieving results longer than 254 characters. It also allows specifying the text to return if the ini entry is not found.
- Fixed uflGetINIValue() so it returns "" when entry is found but it has a blank value.

Version 6.4.9061 (1/18/2019):

- The **Number2Words()** function now has an option to convert a number to Arabic words while expressing the decimal portion as N/100. See sample image.

Version 6.4.9059 (9/23/2018):

- Added uflINISectionDelete() function to delete ini file section.
- Added **uflBulletChart()**, **uflGaugeRadial()**, and **uflSparkline()** functions to generate Radial Gauge, bullet charts. And Sparklines. See image.
- Added uflImageCrop() function. Allows cropping and tight placement of images. See image.

Version 6.4.9043 (7/3/2018):

- Added ufl**KeySetNewItem**() and ufl**KeySetClear**() functions to provide better performance and more control compared to the uflNewKey() function

Version 6.4.9042 (6/25/2018):

- Added uflGoogleSentiment**()** function for natural language sentiment analysis. See image sample.

Version 6.4.9041 (6/2/2018):

- Added **Number2Words()** function to convert a number to Arabic words for a specified currency. See image sample.

Version 6.4.9040 (5/16/2018):

- Added **SQLReturnValue()** and **SQLNoReturn()** functions with no need for ADODB COM and with support for specifying a long sql statement via a single string argument.

Version 6.4.9038 (4/3/2018):

- Added **CultureInfoName()** function to return the computer's locale name (e.g. 'en-US').
- **ImageResize**() now saves to image types based on file extension.

Version 6.4.9037 (2/24/2018):
Internal changes for optimized performance.

Version 6.4.9036 (10/21/2017):
- Fixed handling of Lat/Long pairs as inputs to **GoogleDrivingTimeDistance()** function.

Version 6.4.9035 (10/21/2017):
- added ufl**BB**arcode...() versions to all 16 barcode functions. Two key benefits:
    - **Simpler to use:** because they return the path to the image file. So a single function call in the 'Graphic Location' property of a picture is all you need.
    - **Better Error Handling:** CUT Light automatically fits the text (font size & wrapping) of the error message in the image and returns the path to it. So instead of a barcode image, you see an image of the error message. This makes troubleshooting simpler. It also **avoids displaying the wrong image** (original image used when designing the report).

  See example of the simplified calls, including one case with an error returned here.

Version 6.4.9034 (10/18/2017):
- Added **Barcode39**(), **Barcode93**(), **Barcode128**(), **BarcodeCodeBar**(), **BarcodeITF**(), **BarcodeMSI**(), **BarcodePlessey**(), **BarcodeUpcA**(), **BarcodeUpcE**(), **BarcodeUpcEanExtension**(), **BarcodeEAN8**(), **BarcodeEAN13**() function to generate barcodes of these types on the fly without dependency on font files. See example.
- Added ***BatchFileRun()*** function allowing **triggering BAT or CMD files, with an option to hide the batch file window**.
- **CmdRun()** as well as **BatchFileRun()** now require a 1-time user permission to trigger such functions. The permission is stored as an entry in the ***CUT_Light_Options.ini*** file.

Version 6.4.9031 (9/15/2017):
- Packaged with newer component
- Fixed sporadic Google API handshake issue.

Version 6.4.9029 (7/27/2017):
- Added **XLSLookUp()** function for fast lookups without needing excel to be installed.
- Added **GoogleAddress2LatLong()** function to get the Latitude and Longitude of an address.
- Added **GoogleDrivingTimeDistance()** function to get driving time (taking into account current traffic conditions) and distance between two locations specified as addresses or Latitude|Longitude.
- Fixed 2 Input Box functions.

Version 6.4.9025 (6/21/2017):
- Added support for removing the quiet zone around QR Barcode by setting the AddedMargin argument to -1.

Version 6.4.9024 (5/30/2017):

- Changed **httpToimage()** function to
  a) also support png and bmp images,
  b) support an optional argument of *ImageWidth* (useful when the web page is responsive and you wish to target a certain width), and
  c) support an optional *Wait4Load* argument specifying waiting time in milliseconds, allowing the web page to fully load before being captured as image.
- Optimized **httpToimage()** and **html2image()** to reduce memory consumption.
- Added **httpExists()** function (treats redirect as failure)
- Added **FileUnzip()** function.

Version 6.4.9019 (4/8/2017):

- Added **IpDot2Long()** and **IpLong2Dot()** functions to convert IP addresses between dot and number representations.

Version 6.4.9018 (2/17/2017):

- Implemented code changes to support new deployment to non-standard application location.

Version 6.4.9017 (10/26/2016):

- Added **GoogleTranslate**() function, allowing you to translate any amount of text using the paid ($20 for 1 million characters) Google Translate service. This service supports more than 100 languages.

Version 6.4.9014 (10/17/2016):

- Added 4 new functions to render 2D barcodes: **BarcodeQR**(), **BarcodePDF417**(), **BarcodeDataMatrix**(), and **BarcodeAztec**(). You can use these functions to generate barcode images on the fly and load the image into a picture object on the Crystal report using a dynamic Graphic Location expression (see example).

Version 6.4.9012 (10/11/2016):

- Modified setup properties so you can now install both the 32-bit version as well as the 64-bit version on the same computer. This allows you to develop a report using Crystal Designer (32-bit) and test on the same machine using the 64-bit version of DataLink Viewer.

Version 6.4.9011 (9/11/2016):

- Added **HTML2Image**() function to overcome limitations with how Crystal interprets HTML. Instead, it converts the HTML to an image so you can then load the image into a picture object using a dynamic Graphic Location expression (see example).
- Added **ConvertRTF2Text**() function to convert Rich Text to Plain Text (see example).
- Functions that accept sql1,sql2,sql3,sql4,sql5 arguments no longer insert spaces while combining them into a single SQL statement. This avoids problems when a space should be avoiding. If you need a space, simply include it at the start or end of a segment.

Version 6.4.9009 (8/28/2016):

- Added **httpToImage**() function to allow capturing web page as image and bringing it into the report via the Graphic Location expression.
- Added **FileCompare**() function to check if the content of 2 files is the same.

Version 6.4.9008 (8/3/2016):

- Packaged with newer Chilkat component.

Version 6.4.9007 (4/26/2016):

- Added **SecondsToTimeString**() function to convert seconds to a formatted time string.
- Added **TimeStringtoSeconds**() function to convert a time string to seconds.

Version 6.4.9006 (4/13/2016):

- Added **LookupResetEntries**() function to reset all entries previously set via **LookupAddEntry**() calls. This is useful in cases where a user runs multiple reports without closing the reporting software (Crystal, DataLink Viewer, …) between reports.

Version 6.4.9005 (3/19/2016):

- Added **LookupAddEntry()** and **LookupGetEntry()** functions. This allows storing and retrieving Kay-Value pairs in memory (for example, across report/subreport boundaries) without the complexity and 1K rows limitations of array variables.

Version 6.4.9004 (2/24/2016):

- Added **ImageResize()**function. This allows creating a resized version of a given image file. The resized version can then be loaded into the report using the Picture location expression.

Version 6.4.9001 (10/12/2015):

- Added **GetTextHeight()** and **GetTextNumberOfLine()** function. These functions provide information about the height and number of lines of wrapped text given the allowed width and the font type, font size, and style.

Version 6.4.8002 (9/26/2015):

- Added **httpCallServiceGetTokens()** function to call a web service and return one or more tokens. For example, this can be used to send an SMS message.

Version 6.4.6005 (9/19/2015):

- Added a 64-bit version of the UFL.

Version 6.4.6001 (8/13/2015):

- Added **SetXLSValue()** function to set the value of a spreadsheet cell.
- Added **FileDelete()**, **FileRename()**, and **FileCopy()** functions.
- Added **CmdRun()** function to pass a command line to Cmd.exe

Version 6.4.4001 (8/5/2015):

- Fixed a problem causing DistanceByZip(), DistanceByZip5(), and DistanceByUK() to not be available in Crystal formula editor.
- Optimized DistanceByZip(), DistanceByZip5(), and DistanceByUK() to be much faster.
- Added information for missing USA zip codes.

Version 6.4.2001 (6/4/2015):

- Packaged with a newer Chilkat dll to match the version used by Visual CUT
- Added **httpFileDownloadRename()** function to return the original file name downloaded from the server and to elect to keep or restore that name to the downloaded file.

Version 6.3.1006 (4/1/2015)

- Added 3 Regular Expression functions:
  - **RegExpReplace**() for text replace functionality
  - **RegExpMatch**() for finding and extracting text that matches a pattern
  - **RegExpIsMatch**() for validating that a string matches a pattern

- Added **ConvertRTFFileToText**() for converting RTF file to plain text file.

- Added **ExecuteSQLReturnFile**() for retrieving the content of a database column in a specific record, optionally converting the content:
  RTF to Plain Text or
  Any image type to bmp, png, jpeg, or gif (including an option to convert transparent background to Bitmap with white background), and saving the result to a file.

Version 6.3.1 (1/2/2015): Released .NET version.

Known Issues and Limitation

1. Function arguments passed to a UFL should not exceed 254 characters. That is why some of the functions provided by CUT Light allow you to specify array arguments or use several arguments that are internally combined. In other cases, you simply need to chop the string into segments and loop. For example, here is how you can write a large string to a text:

```
StringVar MyText := {@LargeText};
StringVar TargetFile := "c:\temp\test.txt";
// chop and write the text in 254 character segments
While Len(MyText) > 254 Do
   (
   FileAddText (TargetFile, Left(MyText, 254), False, False);
   MyText := Mid(MyText, 255);
   );
// write the remaining small segment
FileAddText (TargetFile, MyText, False, False);
```

2. In cases where a function accepts array arguments (for example, EmailSend(), you can convert a long string to multiple array elements (each at a maximum of 254 characters. The user manual section about EmailSend() provides an example.

3. A simple preview of a Crystal report may trigger evaluation/formatting of only the 1st page. If you want the Master report to be fully processed upon initial preview (without manually scrolling to the last page), you may need to insert a Special Field such as "Page N of M" to force immediate processing for the whole report.

4. Crystal XI (later versions are fine) suffers from an issue (ADAPT00755322) leading to a termination of Crystal when running a report that uses dynamic parameters on a machine that also has a UFL installed. If you are using Crystal XI and reports with dynamic parameters, you should not install UFLs until Business Objects resolves this issue.