← Click on left for Bookmarks

CUT Light: Crystal UTilities

User Function Library for email, Text, ini Files, Time Zones, SQL, etc.

www.MilletSoftware.com

Version 6.4.9
(August 2016)

By

Ido Millet

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

Disclaimer: These component and accompanying files are provided "as-is" by Ido Millet without assuming any responsibility for harm to computer systems, software, or data with which these files are used.

Dave Clutter (Reed Manufacturing), Ken Hamady (www.kenhamady.com), Jason Lord (Trintech), Ross C. Ryding (Southern Store Fixtures), and Adam Peter Butt (apbreports.com) were instrumental in testing this utility and providing suggestions leading to changes and enhancements.

Introduction

CUT Light allows you to use functions within Crystal Report formulas to:

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. **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.

3. **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.

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

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

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

7. **Lookup Values in the Registry**

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

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

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

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

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

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

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

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

16. **Convert HEX strings to Values**

17. **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.

18. **Encrypt/Decrypt Text**

Install / Uninstall

The zip file you received should contain a **CRUFLido_Light.msi** (*Microsoft Installer*) file containing all the files to be installed and this document. Double-Clicking the **CRUFLido_Light.msi** file will start the installation process, which takes care of registering the dll's and, unless you specify another location, defaults to installing some files to the ***C:\Program Files\CRUFLido_Light*** directory.

Double-Clicking the **CRUFLido_Light.msi** file again after the installation, allows you to Uninstall the software.

If prompted during the install process, you should elect to ignore cases where the file is already being used on your PC (skipping the install of that file) and not overwrite files on your PC with older versions from the Install.

After the installation, the formula editor within Crystal should show the following new functions (under Additional Functions):

Windows & Miscelleneous

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).
False, if this is a new unique key, in which case it gets added to an internal set of keys that is reset only when a new report preview is triggered.

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 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.

ClipboardSetText()

Arguments: (TextToSet)

Returns: The text specified as an argument.

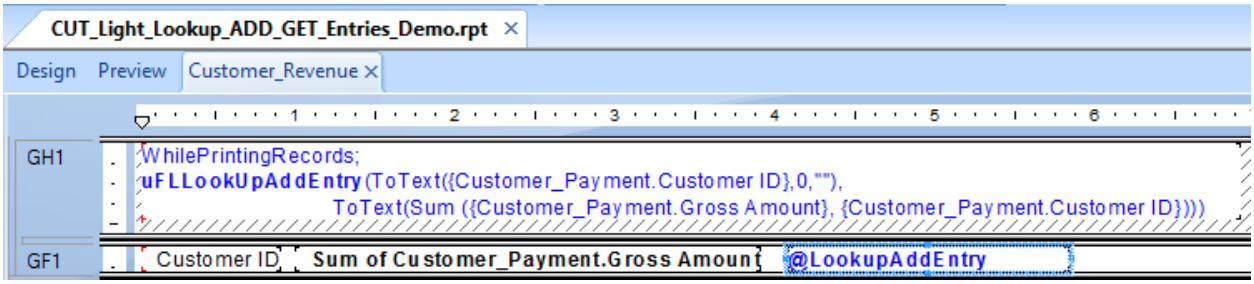
For example: **ClipboardSetText** (ToText({Invoice.Invoice_N},0,'')

LookupAddEntry()

Arguments (sKey, sValue)
Returns: "OK" if the Kay-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.

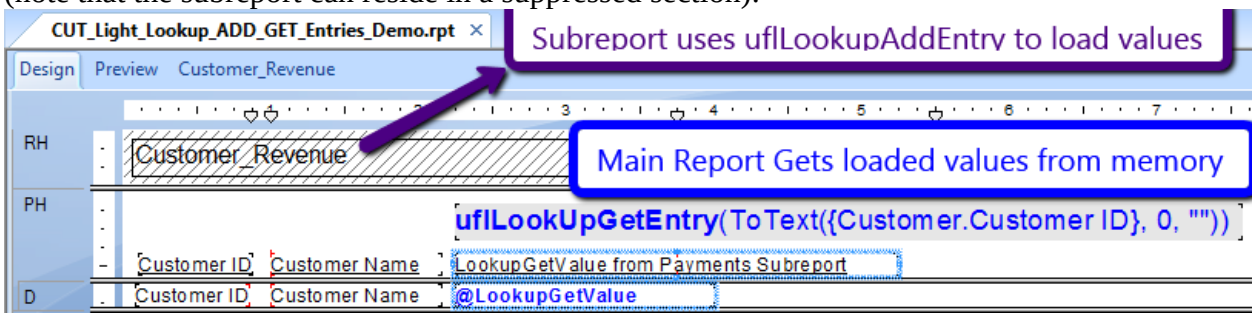
Example of subreport **loading** total payments by Customer into Key-Value pairs in memory:



LookupGetEntry()

Arguments (sKey)
Returns: The Value from matching Kay-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 Sp	$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

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.

GetUser()

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.

GetMachineName()

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

GetRegisteredCompanyName()

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

GetDiskSerialNumber()

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**)

GetMachineIPAddress()

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

GetEnvironmentVar()

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

CreateGUID()

Arguments: none

Returns: a GUID as string.

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

GetImageProperties()

Arguments: (image file path & name)

Returns: Width/Height 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"

BitWiseAnd()

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**

NormDist ()

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

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.

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.

CmdRun()

Arguments: (CommandLineArguments)

This function triggers Cmd.exe passing to it a command line.

Returns: 'Done' or error message.

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.

String/Text Manipulations

PopulateTemplate()

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, "** **")

ExpandStringwithEnvironmentVar()

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

ReplaceAccentedChars()

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.

Hex2Ascii()

Arguments: (String)

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

Returns: the ascii string

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.

File/ini/registry

FileAge()

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

FileCopy()

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.

FileDelete ()

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

FileExists()

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

FileRename()

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.

FileJustName ()

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

FileJustPath ()

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

FileAddText()

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.

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…).

If the text you are writing is longer than 254 characters, break it to multiple calls like this:

```
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);
```

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()**.

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.

FileGetTextUTF8()

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

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 recourses 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

GetTempFolder()

Arguments: None

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

GetINIValue()

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.

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.

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

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.

GMT/Local and Time-Related Functions

GMTtoLocalMinutes()

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.

GMTtoLocal()

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)));
```

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.

SecondsToTimeString()

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"

TimeStringToSeconds()

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

Returns: time string converted to seconds.

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

Message/Input Boxes

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.

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.

InputBox()

Arguments: (Prompt, Title, DefaultText)

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

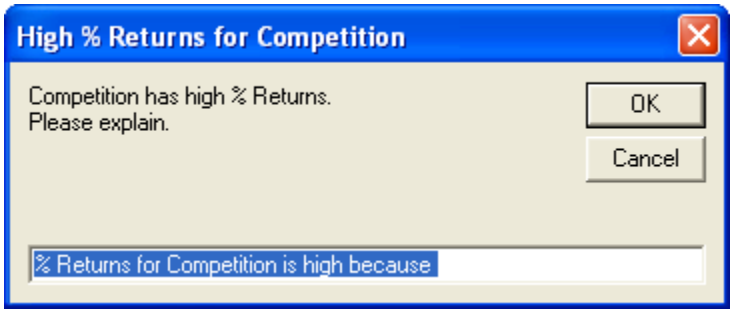
Returns:
Blank text if the user clicks Cancel
The user-entered text (up to 254 characters) if the user clicks OK

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:



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).

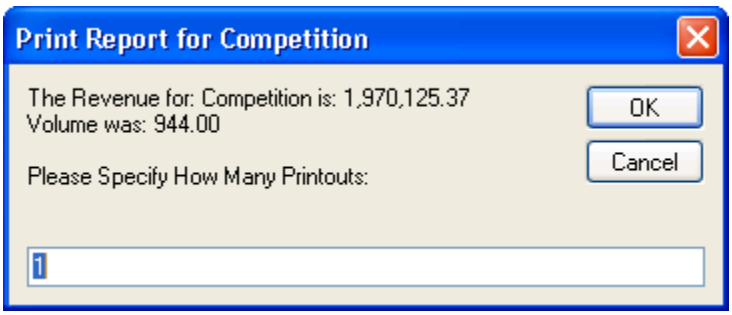
Returns:
Error message if the ExePath doesn't find an exe file in the specified location.
"No Input - Processing Skipped" if the user clicks Cancel or input was blank.
The user-entered text (up to 254 characters) if the user clicks OK

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

HTMLfile2RTFfile()

Arguments: (HTMLfile, RTFfile)

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

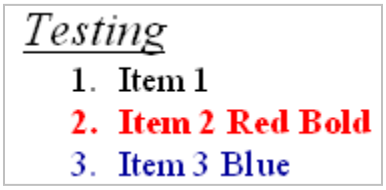
Returns:
"OK" or "Failed"

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.

```
HTMLfile2RTFFile ("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 FileGetTextUTF8( "c:\temp\CUT_Light.rtf", i) <>"" Do
   (sFile := sFile + FileGetTextUTF8( "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:
SAP Note 1214798 - What RTF tags are supported in Crystal Reports?

For a description of HTML rendering limitations in Crystal Reports, see:
SAP Note 1217084 - What are the supported HTML tags and attributes with HTML Text Interpretation?

HTMLstring2RTFFile()

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.

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.

```
HTMLstring2RTFFile ({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 FileGetText( "c:\temp\CUT_Light.rtf", i) <>"" Do
   (sFile := sFile + FileGetText( "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:
http://support.businessobjects.com/library/kbase/articles/c2011504.asp

For a description of HTML RTF rendering limitations in Crystal Reports, see:
http://support.businessobjects.com/library/kbase/articles/c2014842.asp

HTMLString2Txtfile()

Arguments: (HTMLString, Txtfile)

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

Returns:
"OK" or "Failed"

To support cases where the HTML string is longer than 254 characters, if the TxtFile argument is left out, the call simply appends the specified content to an internal variable. If the call specified the TxtFile argument, the resulting variable is also written to the specified text file (and the internal variable is then reset).

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};

//append the content to a global internal variable but don't yet write to a text File
While Len(HTMLString) > 254 Do
   ( HTMLstring2TxtFile(Left(HTMLString, 254), "");
   HTMLString := Mid(HTMLString, 255);
   i := i + 1 );
// Append to the string variable and write to the Text file
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;
---------------------------------------
```

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

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

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.

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.

SQL

ExecuteSQLCanConnect()

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

ExecuteSQLNoReturn()

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.

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.**

ExecuteSQLReturnValue()

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 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","","","","",""))
```

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".

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.

ExecuteSQLReturnDelimitedSegment()

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".

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.

Geo

Distance()

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**

DistanceByZip5()

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().

DistanceByZipUK()

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().

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.

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.

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().

Excel

GetXLSValue()

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.

SetXLSValue()

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.

GetXLSOutput()

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.

Font

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")

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.

Encryption

BlowFishEncrypt ()

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

BlowFishDecrypt ()

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 **BlowFishDecrypt**Segment**()** (see next page).

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 Functions

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.

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

EmailSet()

Arguments: (FromEmail, FromName, ToEmail, ToName,
CcEmail, CcName, BccEmail, ReplyToEmail,
Subject, Message, Attachment,
EnableLog, SMTPHost)

Returns: ignore the return value

Used to set up (but not yet trigger) an e-mail message.
A full example is provided later in this manual.

EmailSet2()

Arguments: (AsHTML, LinksRoot, Priority, ReturnReceipt, UseAuthentication, UsePopAuthentication, POP3Host, UserName, Password, SMTPPort, NoMessageBoxes)

Returns: ignore the return value

Used to follow EmailSet() with more advanced options such allowing **HTML** message bodies, setting message **priority**, requesting **Return Receipt**, providing user id and password for mail host and POP3 **authentication**, and specifying **SMTP Port** in cases where users must override the default port (25).

AsHTML: set this *Boolean* argument to TRUE if your message body should be interpreted and displayed by the receiving email client as HTML instead of plain text. Your message body (specified via EmailSet() and EmailAdd() function calls should includes valid HTML tags such as:

```
"<html><body>" +
"<h1>Hi Ido,</h1>" +
"Here's the sales performance for " +
"<B>" + {Product_Type.Product Type Name} + "</B>" +
"</body></html>"
```

NOTE: if AsHTML is set to TRUE, files (for example **<img src="myfile.gif">**) referenced inside the HTML should be included as file attachments to the e-mail message.

LinksRoot: Applies to messages sent as html. If you set this *String* argument to "**http://www.mysite.com**", you can reference images as
<IMG SRC = "**/images/happyface.gif**"> instead of
<IMG SRC = "**http://www.mysite.com/images/happyface.gif**">.
Note: not supported by all email clients.

Priority: Set this *String* argument to 'High', 'Medium' or 'Low'
If you leave this argument as **""** it will default to 'Normal' priority.
Note: not supported by all email clients.

ReturnReceipt: Set this *Boolean* argument to TRUE to request that a confirmation be emailed back to once the recipient has opened the message.
Note: not supported by all email clients.

UseAuthentication: Set this *Boolean* argument to TRUE if your SMTP mail server requires LOGIN Authentication. In such cases, provide also UserName and Password arguments.

UsePOPAuthentication: Set this *Boolean* argument to TRUE if your SMTP mail server requires 'POP First' authentication. Some servers (e.g., yahoo.com) require POP3

authentication before allowing SMTP transactions. In such cases, provide also POP3Host, UserName, Password arguments.

POP3Host: this *String* argument is required by servers configured for 'POP First' authentication. The POP3Host address can be provided in 123.45.678.9 format or as a domain name (e.g., mymailhost.com).

UserName & Password: Set these *String* arguments if SMTP or POP3 authentication are required.

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

SMTPPort: Set this *String* argument if your SMTP server is not using port 25.

NoMessageBoxes: Set this *Boolean* argument to TRUE if you want to suppress messages due to failed email operations. This feature was requested by one user who needed unattended operation.

Note: call ***EmailSet2()*** after **EmailSet().**

EmailAdd()

Arguments: (ToEmail, ToName, CcEmail, CcName, BccEmail, Message, Attachment)

Returns: ignore the return value

Used to add more recipients, append more text to the e-mail message, or add file attachments to the e-mail message that was set up by a previous ***EmailSet()*** function call.
A full example is provided later in this manual.

Note: By calling EmailAdd() several times you can create long message bodies (Crystal imposes a **limit of 254 characters each time you pass an argument to a function**).

Note: if you set as TRUE the ***AsHTML*** argument of the ***EmailSet2()*** function, make sure you are using proper HTML tags in content you add via the ***Message*** argument. For example, you can call ***EmailAdd()*** for each Product within a Product Type group and specify this as the Message argument:

```
"<ul> <li><p>" + GroupName ({Product.Product Name}) + ": " +
{@Product_Value} +
"</p></li> </ul>"
```

This would append bullets with information about each Product to the e-mail message:

EmailAddText()

Arguments: (MoreText)

Returns: ignore the return value

This function offers a simple way (simpler than ***EmailAdd()*** with its long list of arguments) to add text to the email message body.

Unlike ***EmailAdd(),*** which automatically starts added text on a new line, ***EmailAddText()*** simply continues the previous paragraph unless you specifically request a new line by including CHR(13) + CHR(10) + … within the ***MoreText*** argument.

This provides more freedom in constructing long paragraphs without being forced to start new lines in the middle.

Note: By calling EmailAdd() several times you can create long message bodies (Crystal imposes a **limit of 254 characters each time you pass an argument to a function**).

EmailSend

No arguments. Ignore the Return value.

Used to actually trigger the email message constructed using the ***EmailSet()*** and ***EmailAdd()*** functions above. A full example is provided later in this manual.

Using the Functions

I believe a concrete example is the best way to learn how these functions work. Use Crystal to open and view the **Master_with_multiple_formulas.rpt** report located in the folder where you installed *CUT Light*. This report shows sale information by Product Type for a given year (a parameter).

Let's assume you need to e-mail information from this report so that the manager of each product type would receive the information just for that product type.

Study the following formulas (don't preview the report yet since this would generate e-mails to me):

1. The **@Set_Up_Email** formula is located in a Product Type group header.

   This formula uses the ***EmailSet()*** function (provided by the CUT utility) to set up, but not yet send, the e-mail message to the Product Type manager.

   Important: please be sure to change all e-mail addresses specified in this formula to your own e-mail address. Otherwise, a preview of this report would generate e-mail messages to me.

2. The **@Append_to_Email** formula is located in the Product Name group (level 2) footer.

   This formula uses the ***EmailAdd()*** function (provided by the CUT utility) to append to the text of the e-mail message summary information for each product included in the report export for the current Product Type.

3. Finally, the **@Send_Email** formula is located in the Product Type group (level 1) footer.

   This formula uses the ***EmailSend*** function (provided by the CUT utility) to send the e-mail message constructed by the previous function calls.

Preview

Assuming you changed the e-mail addresses in the **@Set_Up_Email** formula to your own e-mail address, you can preview the report. Select TRUE for the ***?Enable_Email*** parameter.

Monday, April 14, 2003 — Ido Millet, ixm7@psu.edu

Product Sales by Type in 1996

Gloves [$1,614.74]		**Email Setup Done**
Active Outdoors Lycra Glove	$674.85	**Append Done**
Active Outdoors Crochet Glove	$516.20	**Append Done**
InFlux Lycra Glove	$369.69	**Append Done**
Others	$54.00	**Append Done**
		Send Done
Saddles [$750.84]		**Email Setup Done**
Vesper Gelflex ATB Saddle	$180.00	**Append Done**
Xtreme Gellite Ladies Saddle	$117.50	**Append Done**
Xtreme Gellite Mens Saddle	$117.50	**Append Done**
Others	$335.84	**Append Done**
		Send Done
Locks [$577.60]		**Email Setup Done**
Guardian Mini Lock	$325.23	**Append Done**
Xtreme Mtn Lock	$91.63	**Append Done**
Guardian ATB Lock	$39.42	**Append Done**
Others	$121.32	**Append Done**
		Send Done

E-mail Content Sample

You should then get three e-mail messages. The first e-mail message should look like this:

X-PH: V4.1@f04n07
From: "Ido Millet (sender)" <ixm7@psu.edu>
To: "Millet Ido (recipient)" <ixm7@psu.edu>
Subject: Sales Report for Gloves
Reply-to: <ixm7@psu.edu>
Date: Mon, 14 Apr 2003 13:18:24 -0400

Hi Ido,

Here's your report summarizing the sales performance
for Gloves:

Active Outdoors Lycra Glove: $674.85

Active Outdoors Crochet Glove: $516.20

InFlux Lycra Glove: $369.69

Others: $54.00

Note: obviously, in real use, you would use the e-mail address of each Product Type manager. This would require having that information in your database.

Below is a more detailed description of the functionality.

Email Setup using EmailSet(), EmailAdd(), and EmailSend

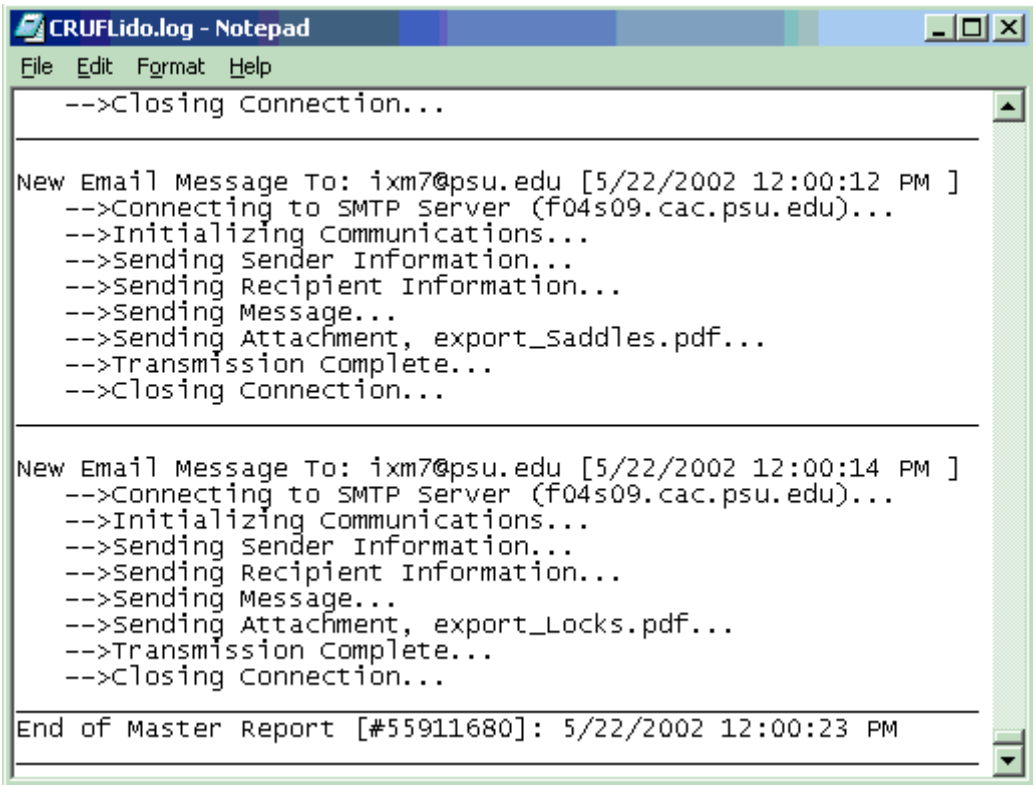
EmailSet() sets up the mail message properties.
EmailAdd() appends more elements or more text.
EmailSend triggers the actual e-mailing.

```
// *** Email Setup using EmailSet()  ***

// Note: obviously, the e-mail functionality can be used on its own,
// but in this case we use it to e-mail the exported report after each bursting cycle.

EmailSet
(
// FromEmail   Note: doesn't have to be your e-mail (e.g., use Sales Manager's e-mail)
"ixm7@psu.edu",
// FromName
"Ido Millet (sender)",
// ToEmail   (e.g., The Product Type Manager's e-mail grabbed from the database)
"ixm7@psu.edu",
// ToName
"Millet Ido (recipient)",
// CcEmail
"",
// CcName
"",
// BccEmail    Blind Copy
"",
// ReplyToEmail  Note: doesn't have to be your e-mail
"ixm7@psu.edu",
// Subject Line
"Sales Report for " + {Product_Type.Product Type Name},
// e-mail message body
CHR(13) + CHR(10) + "Hi, " +
CHR(13) + CHR(10) + "Here's your report summarizing the sales performance" +
CHR(13) + CHR(10) + "for " + {Product_Type.Product Type Name},
// File attachment (in our case we are not attaching any file)
"",
// Enable log file (CRUFLido.log) to track e-mailing activity
TRUE,
// Optional argument: SMTP server for your outgoing e-mails. For example: "smtp.psu.edu"
// In many cases can be left blank ("") to automatically be detected by the utility
"");
```

Effect of the EnableLog option

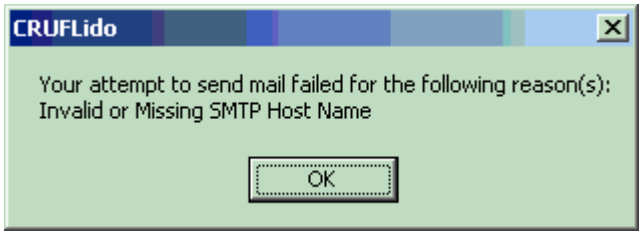
If the ***EnableLog*** option in the ***EmailSet()*** function is set to **TRUE**, all e-mail activity, including failure and success outcomes are logged to **CRUFido.log**

This text file can be opened by *Notepad* or any word processor and is automatically created by enabling this option. It is located at the user's temp folder.
Here is what this log file looks like:



Effect of the *SMTPHost* option

If the ***SMTPHost*** option in the ***EmailSet()*** function is left blank ("") the utility will attempt to automatically detect the SMTP server (the remote computer responsible for processing your outgoing email messages). If you get a timeout message, you should specify the SMTP server yourself using its domain name (for example, "smtp.psu.edu") or its IP address.
Wrong SMTP server option would result in the following message:



Add Email information using EmailAdd()

```
// *** Add Email information using EmailAdd()  ****
//  you can add more Recipients, more Message Lines, and more file attachments
//  by using the EmailAdd() function as demonstrated below

// In this example we append message information about total sales

EmailAdd(
"",                             // Another ToEmail
"",                             // Another ToName
"",                             // Another CcEmail
"",                             // Another CcName
"",                             // Another BccEmail
CHR(13) + CHR(10) +             // Append content to the e-mail message body
"Total Sales Value for Product Type: " + {Product_Type.Product Type Name} +
CHR(13) + CHR(10) +
"in " + {?Year} +
"was: $" +
Cstr(Sum ({@value}, {Product_Type.Product Type Name}), 0),
"");                            // Another File Attachments
```

Note: The ***EmailAdd()*** function does three distinct things in one function call: add **Recipients**, add **Text** to the message body, and add **Attachments**. If all you want to do is add text to the message body, the other arguments would remain blank ("") as shown above.

Note: e-mails sent via this utility will not appear in your OUT box since the operation bypasses your e-mail client software. If you want to see copies of the messages you send, add yourself as a ***To***, ***CC***, or ***BCC*** recipient.

Record Information to a Customized Log File using FileAddText()

```
// *** Write to a Custom Log File using FileAddText()  ****

// The FileAddText() can be used to create any customized text logging/export

// In real life you may want to embed this in an IF THEN 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,
// FALSE = Don't Add an Extra carriage return at the end of the added text
FALSE);
```

Update History

Version **6.4.9** (8/3/2016):

- Packaged with a newer Chilkat dll to match the version used by Visual CUT

Version **6.4.7** (4/26/2016):

- Added **SecondsToTimeString**() function to convert seconds to a formatted time string.
- Added **TimeStringtoSeconds**() function to convert a time string to seconds.
- Added **LookupAddEntry()**, **LookupGetEntry()**, and **LookupResetEntries**() 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 **6.4.6002** (9/13/2015):

- Fixed an issue with httpFileDownloadRename()
- Packaged with a newer Chilkat dll to match the version used by Visual CUT

Version **6.4.6** (8/14/2015):

- 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 **6.4.4** (8/6/2015):

- Added information for missing USA zip codes needed for **DistanceByZip5()**.

Version **6.4.3** (7/6/2015):

- Packaged with a newer Chilkat dll to match the version used by Visual CUT

Version **6.4.2** (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 **6.3.1** (4/1/2015):

- Added **PopulateTemplate()** function to facilitate constructing strings mixing static and dynamic content. The functionality includes special handling for missing or Null dynamic values. This is useful, for example, for constructing HTML table syntax where empty cells should contain a non-breaking space ( )
- Added **FileJustPath()** function to return the path to a file.
- Added **FileJustName()** function to return the name of a file.

Version **6.2.9** (10/14/2014):

- Added **GMTtoLocalMinutes()** function to return the number of minutes between local and universal (GMT) time, taking into account DayLight Savings periods.

- Added **ExpandStringwithEnvironmentVar()** function to return an input string after expanding any references to environment variables within it to their dynamic values.

Version **6.2.8** (8/6/2014):
- Added **GetTempFolder()** function to return the path to the user's temporary folder.
- Changed email logging to use the user's temp folder instead of the application folder of the process running the Crystal report.

Version **6.2.7** (7/28/2014):
- Added **FileAge()** function to return file age in minutes.

Version **6.2.6** (6/25/2014):
- Added **ExecuteSQLCanConnect()** function to test connection to a database before calling other ExecuteSQL… functions.
- Improved error handling for ExecuteSQL… functions to avoid memory full error when connection to database fails.
- If you also own Visual CUT, you can **refer to saved encrypted passwords by their name** instead of specifying the password itself. You need to copy the encrypted password name entry from the ini file in Visual CUT to the CUT_Light.ini file:
  [Options]
  Encrypted_Password_DB=64D26BF23E2C830AE2BE0A8EAC689353159D5618ED8D5D45

Version **6.2.5** (5/12/2014):
- Added **GetImageProperties()** function
  This function allows getting image width and height for a specified image file.
- Added **CreateGUID()** function

Version **6.2.4** (1/1/2014):
- Added **GetXLSOutput()** function
  This function allows specifying an input value for a specified cell, and getting the output value in another cell. One typical use scenario is for **VLOOKUP** functionality.

Version **6.2.3** (12/06/2013):
- Added **LookupText()** function for looking up a value in a text file containing Key-Value pairs.

Version **6.2.2** (11/13/2013):
- Added **GetEnvironmentVar()** function (get Windows environment variable values).

Version **6.2.1** (10/17/2013):
- Added **IsValidEmails()** function (for checking multiple email addresses).

Version **6.1.9** (8/14/2013):

- ExecuteSQLReturn… functions now return an empty string if the SQL statement resulted in a NULL.

- Updated the description of the 4 SQL functions function:
  ExecuteSQLNoReturn()
  ExecuteSQLReturnValue()
  ExecuteSQLReturnDelimited()
  ExecuteSQLReturnDelimitedSegment()
  to reflect the fact that they **support specifying either an ODBC DSN or an OLEDB Connection String as the data source**

Version **6.1.8** (6/19/2013):

- Added **NewKey()** function. This allows avoiding duplicate processing for any function call and to compute Distinct Sums.

- Added **FileAddTextKey()** function. This allows avoiding duplicate processing when writing to a text file.

Version **6.1.3** (5/18/2013):

- Added **GetFontSizeToFitText()** function.

Version **6.1.2** (5/7/2013):

- Added **httpFileDownload()** function.

Version **6.1.1** (4/8/2013):

- Added **HTMLString2Txtfile()** function.

Version **5.9.9** (3/31/2013):

- Added **FileGetTextUTF8()** function.

Version **5.9.8** (3/20/2013):

- Added **IsValidEmail()** function

Version **5.9.7** (1/23/2013):

- Added **BitWiseAnd()** function

Version **5.9.6** (10/19/2012):

- **ExeRun ()** can now also trigger batch (*.bat) and command (*.cmd) files

Version **5.9.5** (9/22/2012):

- **ExecuteSQLNoReturn()** can now embed the content of referenced text files inside the specified SQL. See '***Embedding File(s) Content in the SQL Statement***'

Version **5.9.0** (7/31/2012):

- Fixed a problem with ExecuteSQL functions with multiple segments, if a segment ended in the middle of a word.

Version **5.8.0** (7/7/2012):
- Added **BlowFishDecryptSegment()** function

Version **5.7.0** (6/21/2012):
- **DistanceByZip5() can now handle US Zip codes as well as Canadian Zip Codes**
- Added **GetLatLongFromZip5() function.** This function uses a local data file (CUT_Light.ini) so it is **more reliable than the GetLatLongFromZip() function** (which require a web connection and is subject to a daily quota of hits).
- Added **BlowFishEncrypt ()** function
- Added **BlowFishDecrypt()** function

Version **5.6.0** (4/16/2012):
- Added **DistanceByZipUK() function.**
  Similar to DistanceByZip5() except that it is used for UK Outer Zip codes.

Version **5.5.0** (12/13/2011):
- Fixed ZIP codes data set for cases with missing leading zeros.

Version **5.4.0** (10/25/2011):
- Added **DistanceByZip5() function.**
  This function uses a local data file (CUT_Light.ini) so it is **more reliable than the DistanceByZip() function** (which require a web connection and is subject to a daily quota of hits). On the down side, it is **limited to 5-digit zip codes**.

Version **5.3.0** (06/21/2011):
- Added **FileListFromWildCards()** function

Version **5.2.0** (05/23/2011):
- Added **ClipboardSetText()** function
- Added **GetRegisteredCompanyName()** function
- Added **GetDriveSerialNumber()** function
- Updated the **DistanceByZip()** and **GetLatLongFromZip()** functions

Version **5.1.0** (11/03/2010):
- Added **NormDist()** function

Version **4.9.0** (2/4/2010):
- Added **XLSGetValue()** function

Version **4.8.0** (11/2/2009):
- Fixed handling of New Lines embedded in RTF Files retrieved via **FileGetText()**

Version **4.7.0** (3/7/2009):
- Added **Distance()** function
- Added **DistanceByZip()** function
- Added **GetLatLongFromZip()** function

Version **4.6.0** (1/27/2009):
- Added **httpFileExists()** function

Version **4.5.0** (10/10/2008):
- Added **Hex2Number()** function

Version **4.4.0** (7/13/2008):
- Added **GetMachineIPAddress()** function.

Version **4.3.0** (4/17/2008):
- Added **ExecuteSQLReturnDelimitedSegment()** function.

Version **4.2.0** (3/10/2007):
- Fixed incompatibility with Crystal XI dynamic parameters.

Version **4.1.0** (9/30/2006):
- Added **ExecuteSQLReturnDelimited()** function.
- Fixed a problem with **HTMLstring2RTFFile()** function.
- Added **ReplaceAccentedChars()** function.
- Added **Hex2Ascii()** function.

Version **4.0.0** (6/2/2006):
- Added **InputBox2Command()** function.
- Added **ExeRun()** function.

Version **3.9.0** (4/27/2006):
- Added **ExecuteSQLNoReturn()** function.
- Added **ExecuteSQLReturnValue()** function.

Version **3.8.0** (1/03/2006):
- Added **InputBox()** function.
- Added **HTMLfile2RTFfile()** function.
- Added **HTMLstring2RTFFile()** function.

Version **3.7.0** (11/29/2005):
- Fixed importing of RTF files in **FileGetText()**.

Version **3.6.0** (7/12/2005):
- Added **MessageBoxOK()** function.
- Added **MessageBoxYesNo()** function.

Version **3.5.0** (11/26/2004):

- Added VisualCutRun**()** function.

Version **3.3.0** (09/15/2004):

- Added **SetIniValue()** function.

Version **3.2.0** (08/25/2004):

- **FileAddText()** now creates the target folder if it doesn't exist even when it is specified in **UNC format**.

Version **3.1.0** (08/08/2004):

- Enhanced **FileGetText()** function so it can load large text files (segment by segment) into a String variable in Crystal. It can then be displayed within Crystal as text, RTF, or HTML.

Version **3.0.0** (08/02/2004):

- Added **FileGetText()** function.

Version **2.9.0** (06/07/2004):

- Added **GetRegistryString()** function.

Version **2.8.0** (04/15/2004):

- FileAddText() now automatically creates the specified file directory if it doesn't exist.

Version **2.7.0** (03/12/2004):

- FileAddText() can now create and add content to any text file; not just files with ".TXT" name extensions.

Version **2.6.0** (12/12/2003):

- Fixed a problem leading to SMTP server error messages in special situations.

Version **2.5.0** (09/20/2003):

- Added **GetUser()** function to return the User Name who is logged into 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.

- Added **GetMachineName()** function to return the Name of the PC running the report.

- Incorporated an **updated e-mail component** (vbSendMail version 3.65 instead of 3.54) providing the following relevant improvements:
  - **Removed extra blank line from the beginning of the message body**
  - Fixed time/date formatting error on certain language settings
  - Changed login authentication code to improve compatibility
  - Other minor bug fixes

Version **2.0** (10/10/2002):

- Released with a version number parallel to that of CUT

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 inputs using 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);
```

1. 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.

2. Crystal XI 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.