

CUT Light: Crystal Utilities

User Function Library for e-mail, Text, ini Files, Time Zones, etc.

www.MilletSoftware.com

Version 4.9.0

(February 2010)

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.

Credits:

The SMTP e-mail functionality was adapted from code in *vbSendMail*.

vbSendMail (co-authored by Dean Dusenbery) is available from FreeVBCode.com

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

INTRODUCTION	4
INSTALL / UNINSTALL	5
WHAT FUNCTIONS ARE INCLUDED?.....	6
[1] FileExists()	6
[2] FileAddText()	6
[3] FileGetText()	7
[4] GetINIValue().....	7
[5] SetINIValue()	7
[6] GMTToLocal()	8
[7] GMTtoZone()	9
[8] EmailSet().....	10
[9] EmailSet2().....	11
[10] EmailAdd()	13
[11] EmailAddText()	14
[12] EmailSend	14
[13] GetUser()	15
[14] GetMachineName()	15
[15] GetMachineIPAddress()	15
[16] GetRegistryString().....	16
[17] VisualCutRun().....	16
[18] EXERun()	17
[19] ReplaceAccentedChars()	17
[20] Hex2Ascii().....	17
[21] Hex2Number()	17
[22] MessageBoxOK()	18
[23] MessageBoxYesNo()	18
[24] InputBox().....	19
[25] InputBox2Command()	20
[26] HTMLfile2RTFfile()	21
[27] HTMLstring2RTFFile().....	22
[28] httpFileExists()	23
[29] ExecuteSQLNoReturn().....	24
Avoiding Duplicate Processing	25
[30] ExecuteSQLReturnValue()	26
[31] ExecuteSQLReturnDelimited().....	27
[32] ExecuteSQLReturnDelimitedSegment().....	28
[33] Distance().....	29
[34] DistanceByZip()	29
[35] GetLatLongFromZip ().....	29
USING THE FUNCTIONS.....	31
PREVIEW.....	32
E-MAIL CONTENT SAMPLE	33
EMAIL SETUP USING EMAILSET(), EMAILADD(), AND EMAILSEND	34
EFFECT OF THE ENABLELOG OPTION	35
EFFECT OF THE <i>SMTPHOST</i> OPTION	35
ADD EMAIL INFORMATION USING EMAILADD()	36
RECORD INFORMATION TO A CUSTOMIZED LOG FILE USING FILEADDTEXT()	37
UPDATE HISTORY	38

Introduction

CUT Light provides the same functionality as *CUT* except for the functions dealing with exporting and bursting slave reports. This allows for a much smaller install file and a simpler User Manual. *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.

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.

What Functions are Included?

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

[1] FileExists()

Arguments: (FileName)

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

Can be used, for example, in a Crystal formula to check if the report you are about to export exists. If it doesn't, the formula can skip the steps of emailing the result of the export.

[2] 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...).

[3] 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.

[4] 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.

[5] 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.

[6] 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)));
```

[7] 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.

[8] 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.

[9] 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 ``) 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 `` instead of ``.

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.

UsePOPAAuthentication: 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.

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()*.

[10] EmailAdd()

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

Returns: ignore the return value

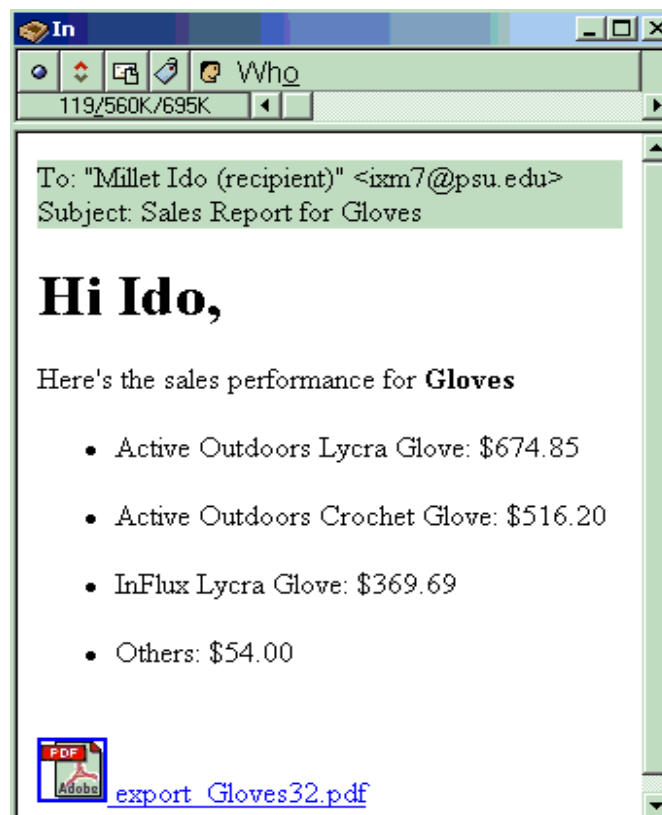
Used to add more recipients, append more text to the e-mail message, or add file attachments to the e-mail message that was set up by a previous *EmailSet()* function call. A full example is provided later in this manual.

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:



[11] 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**).

[12] 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.

[13] 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.

[14] GetMachineName()

No arguments.

Returns: a String with the PC name where the Crystal report is running.

[15] GetMachineIPAddress()

No arguments.

Returns: a String with the IP Address of the machine where the Crystal report is running.

[16] 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%\Local Settings\Temp

[17] 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.

[18] EXERun()

Arguments: (ExePath, CommandLineArguments)

This function triggers another application (EXE 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.

[19] 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.

[20] Hex2Ascii()

Arguments: (String)

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

Returns: the ascii string

[21] 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.

[22] **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.

[23] **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.

[24] 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

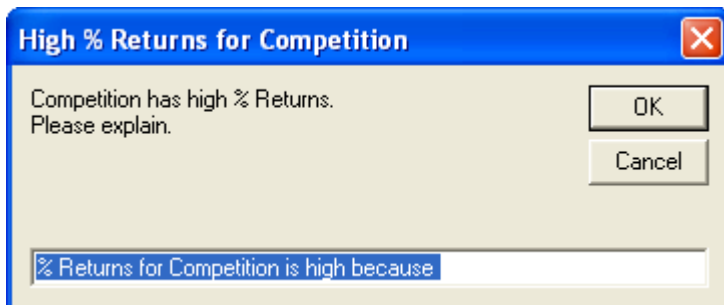
A typical scenario for using this functionality is to add comments to areas in the report that show exceptional (good/bad) performance. This is something that standard report parameters cannot do because:

- 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:



[25] 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.

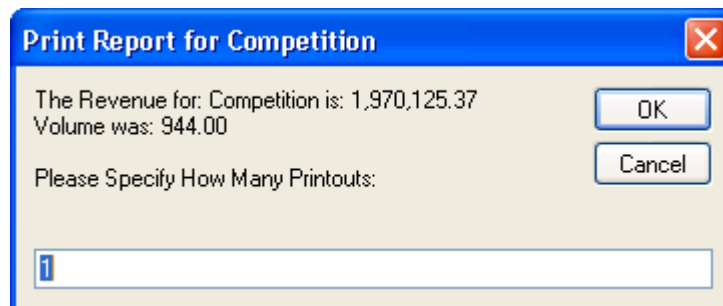
The user-entered text (up to 254 characters) if the user clicks OK

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.

[26] 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 FileGetText( "c:\temp\CUT_Light.rtf", i) <> "" Do
```

```
    (sFile := sFile + FileGetText( "c:\temp\CUT_Light.rtf", i);
```

```
    i := i + 1);
```

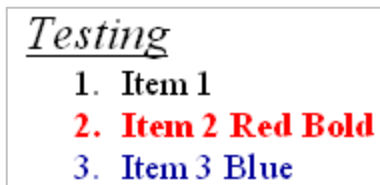
```
// Return the resulting string
```

```
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:



Testing
1. Item 1
2. **Item 2 Red Bold**
3. **Item 3 Blue**



Testing
Item 1
Item 2 Red Bold
Item 3 Blue

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?](#)

[27] 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);
```

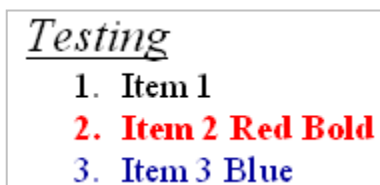
```
// Return the resulting string
```

```
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>

[28] **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")

[29] ExecuteSQLNoReturn()

Arguments: (ODBC DSN, 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). 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 by product of viewing a Crystal report. A typical use is to update a status column to "eMailed" when a Visual CUT process runs a Crystal report and emails the information (Invoice, Order Confirmation, PO, ...).

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

Avoiding Duplicate Processing

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 statement and record the execution time for that particular ticket in the ini file.

[30] ExecuteSQLReturnValue()

Arguments: (ODBC DSN, 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). 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,'') , "" , "" , "" , "" )
```

[31] ExecuteSQLReturnDelimited()

Arguments: (ODBC DSN, 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). 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
```

[32] ExecuteSQLReturnDelimitedSegment()

Arguments: (ODBC DSN, 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). 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.

[33] 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** (distance in miles between the two points)

[34] 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. Please contact Millet Software if the function stops performing. It may require adjustment if the web site providing the information has changed.

[35] GetLatLongFromZip ()

Arguments: (Zip)

Zip is the zip code (e.g., “16563” or “16563-1400”)

Returns: Latitude/Longitude string.

Example: **GetLatLongFromZip("16563-1400")**
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. Please contact Millet Software if the function stops performing. It may require adjustment if the web site providing the information has changed.

[36] 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.

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.

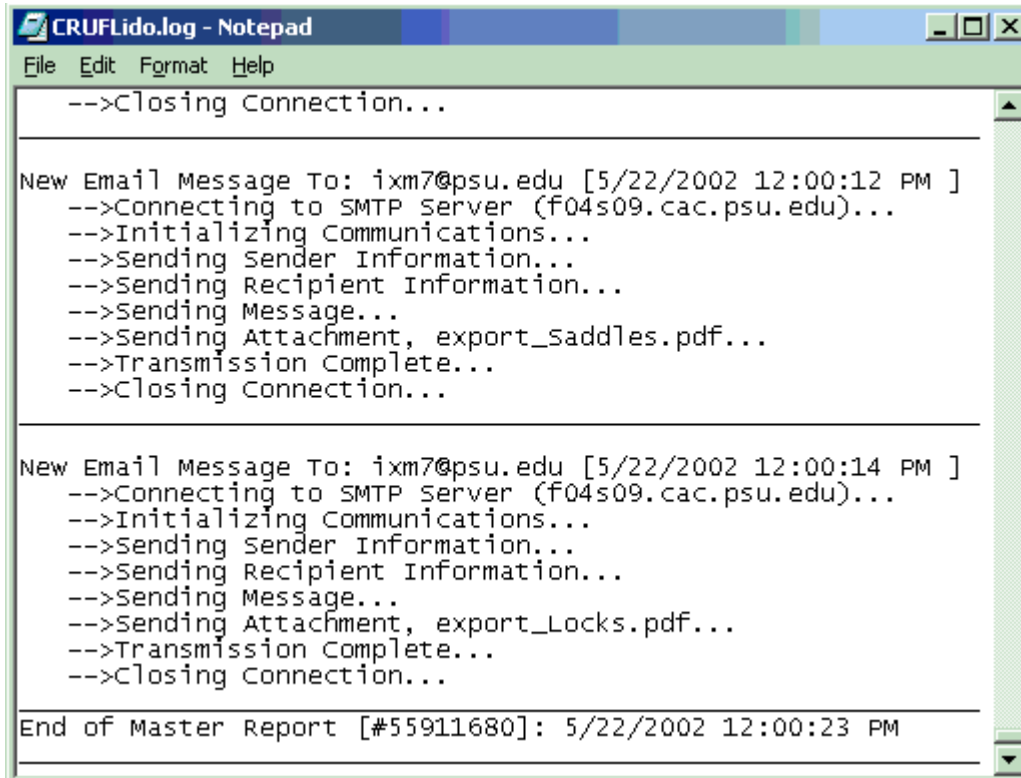
EmailSet
(
// FromEmail Note: doesn't have to be your e-mail (e.g., use Sales Manager's e-mail)
"ixm7@psu.edu",
// FromName
"Ido Millet (sender)",
// ToEmail (e.g., The Product Type Manager's e-mail grabbed from the database)
"ixm7@psu.edu",
// ToName
"Millet Ido (recipient)",
// CcEmail
"",
// CcName
"",
// 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: C:\Program Files\Seagate Software\Crystal Reports (or where the Crystal Reports software is located).

Here is what this log file looks like:



```
CRUFido.log - Notepad
File Edit Format Help
-->Closing Connection...

New Email Message To: ixm7@psu.edu [5/22/2002 12:00:12 PM ]
-->Connecting to SMTP server (f04s09.cac.psu.edu)...
-->Initializing Communications...
-->Sending Sender Information...
-->Sending Recipient Information...
-->Sending Message...
-->Sending Attachment, export_saddles.pdf...
-->Transmission Complete...
-->Closing Connection...

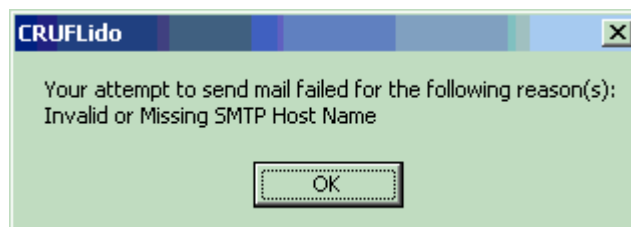
New Email Message To: ixm7@psu.edu [5/22/2002 12:00:14 PM ]
-->Connecting to SMTP server (f04s09.cac.psu.edu)...
-->Initializing Communications...
-->Sending Sender Information...
-->Sending Recipient Information...
-->Sending Message...
-->Sending Attachment, export_Locks.pdf...
-->Transmission Complete...
-->Closing Connection...

End of Master Report [#55911680]: 5/22/2002 12:00:23 PM
```

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 **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** (08/08/200):

- ◆ 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. A simple preview of a Crystal report triggers 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 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 a 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.