ORACLE
INSURANCE

Oracle® Documaker

DAL Reference

version 11.5

Part number: E16256-01
March 2012

ORACLE

THIRD PARTY SOFTWARE NOTICES

This product includes software developed by Apache Software Foundation (http://www.apache.org/).

This product includes software distributed via the Berkeley Software Distribution (BSD) and licensed for binary distribution under the Generic BSD license.

This product includes software developed by the JDOM Project (http://www.jdom.org/).

This product includes software developed by the Massachusetts Institute of Technology (MIT).

This product includes software developed by Jean-loup Gailly and Mark Adler. This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Contents

Chapter 3, Keyword Reference

Chapter 1

Using DAL

This guide provides the information you need to write calculations for variable fields. Field calculations simplify data entry.

For example, entry personnel may be required to enter amounts in three different variable fields. The sum of these amounts determines a total amount which is placed in a fourth field.

You can write a field calculation to automatically enter the amount in the fourth field. Entry personnel do not have to add the amounts and enter the total.

This chapter discusses:

Introduction to DAL

The language you use for field calculations is called the Document Automation Language (DAL). The calculation itself is called a *script*. By using the proper script, you can make sure the data is processed in the manner you intend. This chapter explains calculation language and how to write scripts.

To assign a calculation to a field:

- Enter your calculation directly on the field's Properties window by selecting the Calculation tab.

After you assign a calculation to a variable field, you have these additional options. Choose one of the following:

- DAL calc, if you want the system to recalculate the value of the field as soon as you highlight or enter any field. The system recalculates all Calc scripts for all fields when you highlight a new field.
- DAL script, if you want the system to recalculate the value in the field when you exit the field. The script is executed only when you exit the field containing the script reference and not during any other field actions.
- Disabled, if you do not want to run calculations during Section Check or during entry. This is a convenient way to disable the script without deleting it from the Properties window.

NOTE: The SAMPCO sample resources contain a great number of DAL examples and explanations. Be sure to check out this resource as you create DAL scripts for your company.

Using the Field's Properties Window

You enter calculations for variable fields on the Calculation tab of the field's Properties window. Here is a sample calculation:

Calculation	Return (@("Prem Basis1") * @("Prem/Ops Rate1")/100)
Result	Takes the value of a variable field named PremBasis1 multiplies it by the value of a variable field named Prem/Ops Rate1, divides the product by 100 and places the result in the current variable field.

The calculation language in the Properties window has a particular format. Keep the following formatting points in mind as you enter your calculation in the Properties window:

- You can enter up to 512 bytes (or characters) of information for a calculation. For larger scripts, create them as external files (*.DAL).
- The calculation language is not case sensitive.
- Place comments only on the last line of a calculation. Begin each comment line with asterisks.
- Place a semicolon (;) at the end of each calculation.
- If you have multiple calculations, separate the calculations with semicolons, as shown here:

  ```
  If flag = "y" then return (sum("field")); else return ("exclude");
  end;
  ```

- Extra space and tab characters within script statements are considered white space. White space may appear anywhere in the script to improve readability, but is ignored during the evaluation of the script. Blank lines within external script files are also considered white space.

NOTE: All space and tab characters inside a string constant are not considered white space, but rather part of the string.

Entering Calculations in External Files

You can save a calculation script in an external file. External files containing script calculations are standard ASCII text files. You create and maintain your script files with any standard text file editor. If you use a word processor, remember to save the script file as an ASCII text file. The calculation language that you use within an external file is exactly the same as the language you use in the Properties window.

You may want to use calculations from external script files if your calculations are long or if you want to use identical calculations for various variable fields in multiple sections. You must maintain your external script files in the DEFLIB directory of your master resource library.

To reference an external script file, you must use the CALL or CHAIN functions. The extension of the external script file is usually specified in your FSISYS.INI file as *DAL*. If it is defined in your INI file, you do not have to specify an extension for the file name.

Calculation	Result
Return(Call ("TestCalc"));	Calls a calculation from an external file named TestCalc. Once completed, control returns to the script that initiated the function.
Chain("TestCalc");	Chain executes an external script file but, unlike CALL, does not return to the script that initiated the procedure. Instead, it proceeds to the next calculation.

Formatting the Script

The calculation language in external files has a particular format. Keep the following formatting points in mind as you enter your calculation in an external file:

- The external script file can contain any number of lines. Each line can be up to 255 characters in length. Each line must end in a carriage return/line feed pair (\r\n). You can end the file with a CTRL+Z; however, it is not required that you end the file with CTRL+Z. Most ASCII text editors will handle this automatically.
- Calculation language is not case sensitive. The calculation can be written in either upper- or lowercase.
- Blank lines can occur anywhere in the file. Blank lines are always ignored as the calculation is processed. Use spaces, tabs, and blank lines to improve readability.
- You create comment lines in a calculation by placing an asterisk (*) at the beginning of the line. The system ignores any line which begins with an asterisk during processing. You can place comments anywhere in the file and use them for any reason you choose. Comments are typically used to provide explanations of sections in the file.

  Please note that it is not recommended to include comments in the scripts entered directly onto the Calculation tab of the Properties window. If, however, you do need to include comments, place them at the end of the calculation.

Creating a DAL Script Library

You can also create libraries of DAL scripts as structured named subroutines. The libraries which contain these named subroutines are standard ASCII files.

You can create and maintain the libraries with any standard text file editor. If you use a word processor, just remember to save the file as an ASCII text file.

NOTE: The calculation language you use within a library is exactly the same as the language you use in the Field Properties window.

The layout of the library is shown here. Each script in the file must begin with *BeginSub* and end with *EndSub*.

```
BeginSub SCRIPT1
*   This script returns #x set to 2 if #x was equal to 1 on enter.
IF (#x = 1) THEN #x = 2;
END;
RETURN (#x);
EndSub

BeginSub Script2
*   This script returns a negative one if #y was equal to 5.
if(#y = 5) then Return (-1);
end;
EndSub

BeginSub Parse
*   Parse a word from the string "parse_it"
#position = FIND (parse_it, " ");
word  = SUB (parse_it, 1, (#position - 1) )
parse_it  = CUT (parse_it, 1, #position );
return;
EndSub
```

In this example, SCRIPT1 is the name of the first script, Script2 is the name of the second script, and so on.

SCRIPT1, Script2, and Parse are only names, you can use any name you want as long as it is not the name of a DAL reserved function, statement, or key word such CALL, FIND, IF, and so on. You can use upper- and lowercase letters in script names.

BeginSub and EndSub must be paired per script. You must have a space between BeginSub and the script name. For more information on these functions see BeginSub and EndSub on page 28.

NOTE: If you plan to use the XDB to update (separate) DDT file information, keep in mind that DAL scripts stored in the data section should follow the requirements specified for DDT data entry.

This means that if you continue to use separate FAP and DDT files in version 11.0 and higher, the DAL statement separator should be two colons (::) rather than the normal semicolon (;).

If you use Documaker Studio with the new merged FAP files, you can use a single semicolon (;) as the statement separator in your rule data. The use of two colons (::) is no longer required. Note however, that the system will process the two colon (::) statement separators correctly.

Also keep in mind that when you are entering a script into the AFGJOB.JDT file – as a PreTransDAL or a PostTransDAL – you must use two colons (::) as the statement separator. For instance if you write multiple DAL statements into the data area, you must use two colons (::) as your statement separator.

Loading a DAL library

Once a DAL library is loaded, you can reference the scripts in the library by name. You do not have to use CALL or CHAIN.

For example, assume the DAL library file, EXAMPLE.DAL contains the sub-routine functions on the previous page and the file has been loaded into cache memory using the following INI control group and option:

```
< DALLibraries >
    Lib  = example.dal
```

In this example, you reference the sub-routine function name directly: Script1( ) or Script2( ).

```
If ( @("multiphy_value") = " " Then

Return( Script1( ) )
Else
Return( Script2( ) )
End
```

You can execute SCRIPT1 or SCRIPT2 or neither after using the LoadLib function. For more information, see LoadLib on page 301.

NOTE: You should only execute the LoadLib function *once*. You can execute the scripts in the library as many times as you wish.

For more information, see Using INI Options on page 8 and LoadLib on page 301.

Executing a DAL Script from a Menu

You can use the AFEBatchDalProcess MEN.RES option to execute any DAL script from a menu option. For instance, you can use this option to run a script which batch processes all of the current WIP for the current user.

To use this option, include a line similar to the one shown here in your MEN.RES file:

```
MENUITEM  "Batch DAL..." 294 "AFEW32->AFEBatchDalProcess""Process
DAL in Batch"
```

This line tells the system that when a user selects the Batch DAL option, it should execute the script identified in the following INI option. Make sure your FSIUSER.INI or FSISYS.INI file includes this control group and option:

```
< Batch_DAL >
    ScriptFile = xxx.DAL
```

Where *xxx* is the name of the DAL script you want the system to execute. You must use the extension *DAL*.

Here are some examples:

Script name	Content	Results
COMPLETE.DAL	Complete ();	Completes each entry in WIP. This is the same result as if you chose the File, Complete option.
ASSIGN.DAL	AssignWIP (Fanelli);	Assigns each entry in WIP to the user ID ***Fanelli***. This is the same result as if you chose the Formset, Assign option.
ASSIGN1.DAL	If WIPKey1()="Account" then AssignWIP (Brown);end;	For each entry in WIP whose WIPKey1 equals ***Account,*** the script assigns the documents to the user ID ***Brown***. This is the same result as if you chose the Formset, Assign Document option.

Using INI Options

You can use several FSISYS.INI file control groups and options to control the way the system processes DAL functions and scripts. These options let you:

- Purge or retain target variables between form sets.
- Specify the file extension for external DAL scripts.
- Determine which DLL-based DAL functions are automatically registered and available to your DAL scripts at runtime.
- Specify the name of the DAL script you want to execute.
- Set the title for the DAL runtime tool. For more information about the DAL runtime tool, see Testing DAL Scripts on page 31.

This table shows the various control groups and options, along with a description of what you should enter for each option.

Option	Explanation
Control control group	
FlushSymbols	Enter No to maintain the defined target variables and their contents from the previous form set. The default is Yes, which tells the system to delete DAL target variables between form set processing.
DateFmt2To4Year	Enter the cutoff year for determining the century. For instance, if you enter 50 for this option, the system assumes a two-digit year greater than or equal to 50 should be prefaced by *19*. If you omit this option, the system assumes the current century when it encounters a two-digit year. All internal date manipulation is performed using four-digit years.
DAL control group	
Ext	Enter a period and an extension. The default is *DAL*. Use this option to define the file extension used for external DAL scripts and file names.
DALFunctions control group	
Keyword	Enter **DLLMOD->FunctionName**. This option defines the DLL-based DAL functions that are automatically registered and made available to the scripts executed in the session. This option is used by the DAL runtime tool (DALRUN).
DALLibraries control group	
CompileWhenLoaded	Enter Yes to compile each DAL library file when loaded. In situations where you are processing a lot of transactions and you have a lot of DAL functions which are used during processing, this can speed performance. The default is No.
Lib	Use this option to specify the DAL library file to be loaded. You can specify multiple files. There is no default for this option.

Option	Explanation
DALRun control group	
Script	Enter a file name. Use this option to specify the file name of the script to execute. You can use any file extension. If you omit the extension, the system assumes if is *DAL*.
Title	Enter a title.The default is *DALRUN - Document Automation Language Runtime*
RunMode control group	
FlushDALSymbols	Enter Yes to clear DAL internal variables set by the previous transaction before the subsequent transaction is processed. Use the Retain function to identify DAL variables you do not want cleared. The default is No.
Debug_Switches control group	
DALLib	Enter Yes to have the system create debug information related to the execution of library subroutines. The default is No.
Debug_DAL_Rules	Enter Yes to create debug date related to the execution of each DAL function or procedure that is executed. The default is No.
DumpDAL	Enter the name of the DAL script for which you want to generate debug data. You can also enter **All**, which tells the system to generate data for all DAL scripts. Be sure to set the DALLib option to Yes if you use the DumpDAL option. The system sends the output to the file you specified with the TraceFile option in the Data control group or your default trace file.
VerifyKeyID control group	
Script	Enter the name of the DAL script you want the system to use. Store this script in the DefLib directory or in MASTER.LBY if you are using Library Manager.
MasterResource control group	
DALTriggers	Enter the name of the DAL library file that contains your section trigger scripts (DAL triggers). The default is the name stored in the FormsetTriggers option in the MasterResource control group. If this option is omitted, the system looks for *SetRcpTb*.

The system also provides a number of specialized INI functions. For more information, see INI Functions on page 70.

Using Built-In Functions

Use the DALRUN and DALVAR built-in functions to execute DAL scripts or get DAL variable information you can use to complete INI options. For instance, you can use this to map unique recipient information into batch records.

These functions are automatically registered when DAL is initialized. Several programs can initialize DAL, such as the GenData and GenPrint programs, the AFEMAIN program (including RACLIB/RACCO), Documaker Studio, Image Editor, and various utilities such as ARCRET, ARCSPLIT, and DALRUN.

NOTE: If you try to use these functions in systems that do not initialize DAL, an incorrect INI value is returned.

Here is an example:

```
< INIGroup >
    Option1 = ~DALRUN MY.DAL
    Option2 = ~DALVAR XYZ_VAL
```

If the program requests Option1, the script MY.DAL is executed and the resulting option is assigned.

If the program requests Option2, the DAL variable XYZ_VAL is located and its contents are assigned to the INI option.

Using this function with the GenPrint program to initialize INI options can produce errors. At the point in the GenPrint program that INI files are loaded, the system may not have processed enough information to use some DAL functions in the script executed by this function. Here is an example:

```
< PDFNames >
    Archive = c:..\Output\~GetEnv ExtrFileName ~DALRUN Archive_Name
< Printer2 >
    Port = <PDFNames> Archive =
```

Here is the problem statement from the DAL script (ARCHIVE_NAME.DAL):

```
f_name = "_"  & GVM("RunDate") & "_A" & newcount & "_" &
GVM("PolicyNumber")
```

Instead you will receive an error message similar to the following.

```
DM12041:  Error : FAP library error: Transaction:<>,
    area:<..\C\genbannr.c,Jun 23 2004
20:14:14,400.110.002,GENDALErrorNotify>
        code1:<0>, code2:<0>
        msg:<Script: c:..\Deflib\Archive_Name.dal
        Line: 6  Col: 33  Err: 15  Token: )
        Msg: No result value returned>.
```

In this example, the GVM values, RunDate and PolicyNumber have not been loaded.

Checking KeyID Entries

In addition to the following restrictions on KeyID values, you can use DAL to make sure that data entered conforms to a specific alpha and numeric format. For instance, KeyIDs can be:

- Limited by the use of the AutoKeyID table (only accepts KeyIDs listed in the table)
- Limited as to whether there can be duplicates in WIP or archive or both
- Converted to uppercase (if the CaseSensitiveKeys option is set to No)
- Limited to the length defined in the database. (A standard WIP file allows 20 characters for the KeyID.)

NOTE: KeyIDs are typically used as the policy, document, or form set number.

In version 10.2 and higher, you can use the VerifyKeyID hook to call a DAL script. Within the DAL script, the verification can be constant, or provide exceptions based on the Key1 (Company), Key2 (Line of Business), or the transaction code currently selected.

All the relevant WIP record information taken from the Form Selection window is available to the DAL script for examination. Simply use the available DAL functions like WIPKeyID on page 433, WIPKey1 on page 431, or WIPFld on page 430.

NOTE: The script can retrieve WIP values, but not change them.

You must handle any error messages using the MSG function. See MSG on page 323 for more information.

To install the KeyID validation hook, include these INI options.

```
< AFEProcedures >
   AutoKeyID = TRNW32->TRNVerifyKeyID
< VerifyKeyID >
   Script = KeyID.DAL
   OnCreate = Yes
   OnUpdate = No
```

Option	Description
AFEProcedures control group	
AutoKeyID	Enter **TRNW32–>TRNVerifyKeyID** as shown above to install the KeyID validation hook.
VerifyKeyID control group	
Script	Enter the name of the script you want the system to use. Store this script in the DefLib directory specified for your master resource library (MRL). If you omit this option, a message appears on the Form Selection window. You will have to exit and correct the INI file by either defining the script or removing the hook declaration.
OnCreate	This option defaults to Yes to indicate you want to call the script when creating a new form set via the Form Selection window. To exclude newly-created form sets, set this option to No.
OnUpdate	This option defaults to No to indicate you do not want to call the script to verify the KeyID on transactions that have already been saved to WIP. To verify WIP transactions as well, set this option to Yes.

The script can do whatever evaluation is necessary for validation purposes. Here is an example DAL script that validates a KeyID using a format token string.

```
------------------------------------------------------------------
* Define the format requirement in the fmt variable below.
* 9 - means numeric
* A - means alphabetic
* X - means alphanumeric
* * - means any character - not limited to alphabetic or numeric
* For example, if you need 4 numeric, followed by 2 alpha, followed
* by 2 numeric, followed by 2 alphanum, you would define:
* fmt = "9999AA99XA"
* The length of the overall format string is assumed to also define
* the required length of the key value.
* Note DAL does not support case sensitive string comparisons.
* Therefore, it assumes either case is sufficient and that if the
* key is required to be in uppercase, you have set the
* CaseSensitiveKeys option to No.

fmt="9999AA99XA"

* This next statement is used to get the KeyID prompt
name = GETINISTRING(,"DlgTitles", "KeyIDTitle", "Policy #");
```

```
val = WIPKeyID();
if (val = "")
* This is returned successfully because a blank key is going to
* be handled by the Form Selection window anyway.
    return("Yes");
End
#l = len(fmt);
if (#l != len(val))
    msg(name, "Length must be " & #l & '.');
    return("No");
End
* Now example each character from right to left because we
* already have the length from the earlier check.
top:
if (#l = 0)
    goto done:
end
f = sub(fmt,#l,1);
g = sub(val,#l,1);
if (f = '9')
    if (NUMERIC(g) = 0)
        msg(name, "Position "& #l & " must be numeric.");
        return("No");
    end
elseif (f = 'A')
    if (g < 'A' OR g > 'Z')
        msg(name, "Position "& #l & " must be alphabetic.");
        return("No");
    end
elseif (f = 'X')
    if (NUMERIC(g) = 0)
        if (g < 'A' OR g > 'Z')
            msg(name, "Position "& #l & " must be alphanumeric.");
            return("No");
        end
    end
elseif (f != '*')
    msg("Invalid format found at position " & #l & ".");
    return("No");
end
#l -= 1;
goto top:
done:
return("Yes");
-------------------------------------------------------------------
```

Grammar and Syntax

Document Automation Language controls every aspect of the calculation. You control what type of calculation takes place, the sequence of the calculation, and where the calculation result is placed in the form set. It is important that you understand the calculation language as you write scripts. The calculation language consists of:

- Assignment Statements

  Assignment statements are used to place a value from the right side of an equation into a target variable on the left side of an equation.

- Flow Control Statements

  Flow control statements manage the sequence of the calculation. These language statements direct the order in which the calculation is executed and the placement of the calculation result within the form set.

- Data Storage Statements

  These statements return target variable data to the section variable fields.

NOTE: You can also get information about the various DAL keywords in the Keyword Reference on page 451.

Assignment Statements

Assignment statements give values to target variables. Assignment statements have two parts: a target variable and a source expression. The source expression determines what is used to obtain a result. The target is assigned the result of the calculation. The assignment statement format is:

```
Target = Source expression
```

Target variables can be one of these types: string, integer, or decimal. Targets always receive a value that matches their assigned type. Target variables retain data until it is placed in the form set or used in another calculation or expression.

The *source expression* specifies what calculation is performed. Source expressions can be simple or complex. Simple expressions assign the value of a section variable field to the target, or they assign a constant value to the target variable. Complex expressions calculate results from multiple sources.

NOTE: The result of the source expression is *always* converted to the assigned type of the target variable, unless the result of the source expression is a decimal.

Target variable

The target variable contains the result of the source expression calculation. Data is placed in the target after the calculation is performed. The data is maintained in the target until you replace it via another statement. Any script that uses a target value always uses the last value received by that target. This lets you reuse target values.

Target variable names are not case sensitive. Mixed case has no affect on how the name is processed or read during a calculation. Mixed case can be used for clarity. A target name cannot be a reserved keyword. A target's type is designated by the first character of its assigned name. Target variables are one of these types:

- string
- decimal
- integer

Each type is explained below.

- String Target Variables

  String target variable names start with a letter (a- z). The name can be up to 20 characters in length. The remaining characters in the name can be any upper- or lowercase letter, number, the underscore (_), or percent sign (%). Here are some examples:

  ```
  First_Name = "John"
  LASTNAME = "Graham"
  LAST_NAME = "Graham"
  CompanyName = "Oracle"
  ```

  The value received by a string target variable can be from zero to 255 ASCII characters in length.

- Decimal Target Variables

  Decimal target variable names start with a dollar sign ($). The name can be up to 20 characters in length. The remaining characters in the name can be any upper- or lowercase letter, number, the underscore (_), or percent sign (%). Here are some examples:

  ```
  $BEGIN_BAL = 100.00
  $Final_Balance =   00.00
  ```

  Decimal target variables receive numeric values with decimals. The values in these fields can contain up to 14 digits and a decimal.

- Integer Target Variables

  Integer target variable names start with a pound sign (#). The name can be up to 20 characters in length. The remaining characters in the name can be any upper- or lowercase letter, number, the underscore (_), or percent sign (%). Here are some examples:

  ```
  #Employees = 3000
  #Number_of_Insured = 2300
  #%Insured = (#Number_of_Insured / #Employees * 100)
  ```

  Integer target variables receive numeric values as whole numbers — no decimals. The values in these fields can range from plus or minus two billion.

Declaring Variables

In most cases, you do not have to worry about specific variable types when using DAL. Unqualified names are considered string variables and DAL automatically converts the type, depending upon the use. You can, however, force a variable to be something other than a string type by using a specific name qualifier.

Qualifier	Description
$myFloat	The $ denotes that this is a floating point number.
#myInteger	The # denotes that this is an numeric integer.
%myHandle	The % denotes that this is a numeric handle. No conversions should be done on this when used in DAL.

Handle type variables are the exception when it comes to conversions. DAL cannot convert the other variable types into a handle type and a handle type cannot be converted into the other types. For any function that requires a *%variable* as a parameter, you must specify that type of parameter.

Source expression

Source expressions specify what calculation is performed. The result of the source expression is placed in the target variable. Source expressions can contain form set variable field names, target variable results, numeric constants, string constants, keywords, operators, punctuation, and labels. Each of these source expression language categories is explained in the following topics.

Form set variable fields

Variable fields which exist in the form set can be used in the source expression. Variable field names which are used in the source expression must be written in a particular format. The name must be enclosed in quotes. Here is an example:

```
$SubTotal = sum ("Amount")
```

In this example, the sum of the all section fields that have names starting with *Amount* are subtotaled. The result is stored in the decimal target variable named *$SubTotal*. Form set field names are not case sensitive.

NOTE: If you want to use a particular field name, the name must appear in this format:

```
@("ThisField1")
```

Be sure to include the parentheses and the quotation marks.

Target variables

A target variable which results from one source expression can be used in a subsequent source expression. All three target variable types (string, decimal, and integer) can be used in a source expression. Here is an example:

```
$FinalTotal = $SubTotal + 15.00
```

In this example, the value of the decimal target variable $SubTotal (which was previously calculated) is added to the constant value of 15.00. The result is stored in a new decimal target variable named $FinalTotal.

Numeric constants

You can use numeric constants anywhere in a source expression. There are two types of numeric constants: integer and decimal. Do not include commas in either type.

- Integer Constants contain whole numbers. Negative integer constants are preceded by a minus sign. Here is an example of a source expression which contains an integer constant:

  ```
  $FinalTotal = $SubTotal + 15
  ```

  In this example, the integer constant 15 is added to the value of the decimal target variable $SubTotal (which was previously calculated). The result is stored in a new decimal target variable named $FinalTotal.

- Decimal Constants contain fractional numbers with a decimal point. They can contain a fractional portion, represented by the digits to the right of the decimal point. Negative decimal constants are preceded by a minus sign. Here is an example of a source expression containing decimal constants:

  ```
  $My_Dec_Constant = 3.14810
  $Answer = $My_Dec_Constant * 10.80
  ```

  In this example, the decimal constant 3.14810 is stored in the decimal target variable $My_Dec_Constant. The value in the decimal target variable $My_Dec_Constant is then multiplied by the decimal constant 10.80. The result is stored in a new decimal target variable named $Answer.

String constants

You can use string constants anywhere in the source expression. String constants are any group of consecutive characters. String constants can consist of 1 to 253 characters. The characters are delimited either by apostrophes (' ') or by quotation marks (“ “). Use quotation marks if you need apostrophes inside the constant. The string constant consists of everything between the delimiters, including spaces. Here is an example of a source expression containing string constants:

```
My_String_Constant = ' Congratulations on your purchase. '
Greeting = My_String_Constant & ”Thank you for choosing us.”
```

In this example, the string constant *' Congratulations on your purchase. '* is stored in the string target variable My_String_Constant. The value in My_String_Constant is then added to the string constant *Thank you for choosing us*. The result is stored in the string target variable named Greeting.

When Greeting is returned to a field, it appears as:

Congratulations on your purchase. Thank you for choosing us.

Operators

Operators are used in the source expression. Operators control what calculation is performed using the other components in the source expression.

Operator	Function
=	Assignment operator or logical test for equality.
+	Addition.
+ =	Value on the right is added to then assigned to the target variable on the left.
-	Subtraction. Unary minus (negative)
- =	Value on the right is subtracted from then assigned to the target variable on the left.

Operator	Function
*	Multiplication
* =	Value on the right is multiplied with then assigned to the target variable on the left.
/	Division
/ =	Value on the right is divided into then assigned to the target variable on the left.
&	String concatenation.
& =	Value on the right is concatenated to then assigned to the target variable on the left.
>	Logical greater than.
<	Logical less than.
!	Logical not. Returns the opposite of the tested value. (For example: !(10=9) = true)
!=	Logical not equal. Tests if the value at the left is not equal to the value at the right.
>=	Logical greater than or equal.
<=	Logical less than or equal.
!>	Logical not greater than.
<!	Logical not less than.
!>=	Logical not greater than or equal.
!<=	Logical not less than or equal.
AND	Connects two values. Both values must evaluate true to produce a true result.
OR	Connects two values. Either value can evaluate true to produce a true result.

Punctuation

Four types of punctuation can be used within the source expression. Punctuation is used to enclose subexpressions within the main source expression or to establish parameters. Each punctuation mark performs a particular function.

Punctuation	Function
( )	Encloses subexpressions or parameter lists. Indicates precedence of execution within calculations. Parentheses can override the normal execution order.
,	Separates parameters of built-in functions. See Function Reference on page 39 for an explanation of built-in functions.
;	Separates statements.
\	Continues a statement on the next source line.

Execution order

Operators, in combination with punctuation, are executed in a particular order. Normally, operators are executed from highest to lowest priority. When two operators are of equal priority, left to right execution applies.

The normal order of execution is overridden by the use of parentheses. Expressions in parentheses are executed first. In a set of parentheses, operators are executed from highest to lowest priority. Operators of equal priority within parentheses are executed from left to right. Operators are ranked and executed in this order:

Operator	Order of Execution
( )	Highest priority—executed first
- (Unary minus (negative))	Second highest priority—executed after operations in parentheses
* / (Multiplication and division)	Third highest priority.
+ - & (Addition, subtraction, string concatenation)	Fourth priority
! != (Logical not and logical not equal)	Fifth priority
AND OR	Sixth priority
= (Assignment)	Lowest priority

Here are two example assignment statements. The components and execution order of each statement is fully explained.

```
$AMOUNT = @("BEG_BAL") + 100.00
```

Target variable	$AMOUNT
Source expression	@("BEG_BAL") + 100.00
Calculation	Takes the value in the section variable field named BEG_BAL adds 100.00 and places the result in the target decimal variable named $AMOUNT
Order of execution	Reads the expression from left to right

```
$AMOUNT = (@("PremBasis1") + @("PremBasis2")) * @("Prem/OpsRate1")/
100
```

Target variable	$AMOUNT
Source expression	(@("PremBasis1") + @("PremBasis2")) * @("Prem/OpsRate1")/100
Calculation	Takes the value in the section variable field named PremBasis1 adds the value in the section variable field named PremBasis2; multiples the total of these two fields by the value in the section variable field Prem/OpsRate1; then divides the total by 100 and places the result in the target decimal variable named AMOUNT.
Order of execution	Reads the expression from left to right applying the priority of operators (multiplication and division prior to addition). However, the first set of parenthesis overrides the normal priority, so the addition operation is performed first.

Implicit conversion

Implicit conversion occurs when operands of differing types are acted upon by an operator. During assignment, the result of the operand on the right will always be implicitly converted to the type of operand on the left of the assignment operator. This table outlines the conversion rules that occur in operations other than assignments:

Expression operands	Implicit conversion of operands	Internal result type
STRING op INTEGER	STRING op STRING	STRING
STRING op DECIMAL	STRING op STRING	STRING
STRING op STRING	STRING op STRING	STRING
INTEGER op INTEGER	INTEGER op INTEGER	*INTEGER DECIMAL
INTEGER op DECIMAL	DECIMAL op DECIMAL	DECIMAL
INTEGER op STRING	INTEGER op INTEGER or **DECIMAL op DECIMAL	INTEGER DECIMAL
DECIMAL op INTEGER	DECIMAL op DECIMAL	DECIMAL
DECIMAL op DECIMAL	DECIMAL op DECIMAL	DECIMAL
DECIMAL op STRING	DECIMAL op DECIMAL	DECIMAL

* The result of division between INTEGER data types is always a DECIMAL.

** When a string requires conversion to a numeric value it is converted to a DECIMAL data type if it contains a valid decimal value otherwise, it is converted to an INTEGER data type. The resulting type then determines which implicit conversion rules apply.

Here is an example:

```
#val=$temp
```

The value of $temp is converted (internally) to an integer because the assignment is to an integer. During this implicit conversion, the actual value contained in $temp is not changed. If $temp has a value of 10.25 before executing this statement, #val would now have a value of 10.25, and $temp would still be 10.25.

NOTE: Operands of differing types can be assigned to each other, but this does not mean that the two operands will be equal after such assignment.

In this example...

```
#val="January"
```

the string constant would be converted to an INTEGER before assignment. Since the string constant does not contain a valid number, the value of #val will be zero (0) after execution of this statement.

In this example...

```
$temp= 10/6
```

the constants 10 and 6 are of type INTEGER because they have no decimal value indicated. The resulting internal calculation will be a DECIMAL because the act of division always results in a DECIMAL value. Therefore, the value of $temp after the evaluation will be 1.66667. To assign the integer result of division into a DECIMAL data type, it will be necessary to first assign the result into an INTEGER data type, or to use the expression as the parameter to the INT built-in function.

Here is an example of implicit conversion differences:

```
TEXT="001";
IF (TEXT=1);
    TEMP1="YES";
ELSE;
    TEMP1="NO";
END;
IF(1=TEXT);
    TEMP2="YES";
ELSE;
    TEMP2="NO";
END
```

After executing these statements, TEMP1 will contain *NO* and TEMP2 will contain *YES*.

In the first IF statement, the expression (TEXT=1) compares a string with an integer. According to the rules of implicit conversion, the integer is first converted into a string and then the two objects are evaluated according to the operator. When comparing strings, 001 does not equal 1.

In the second IF statement, the expression (1=TEXT) compares an integer to a string. Implicit conversion will change the string into an integer before performing the operation. The converted expression can be represented as (1=1), which are equal.

Labels

Labels are a name for a location within a script. Labels must end with a colon (:). The label can be up to 20 characters in length (including the colon). Labels must appear on a line by themselves. Labels are not case sensitive. Here is an example:

```
TOP:
#Num = #Num +1
If #Num < 22
$Temp = $Temp + @ ("Prem/OpsPrem" & #Num)
GOTO TOP:
END
```

Labels are frequently used as the destination of a GOTO flow statement. For more information about flow statements, see Flow Control Statements on page 22.

Flow Control Statements

Flow control statements dictate how the calculation is executed. They control how the components of the source expression are used. Flow control statements are embedded in the source expression. Flow control directs the use of the source expression components.

Keywords

Keywords are used for flow control statements. These words define the statement operations. These keywords are reserved for use in calculation language. The keywords cannot be used as variable field names. Keywords are not case sensitive.

Keyword	Flow Control
IF	Begins a conditional statement (Optional)
AND	Used within an IF statement (Optional)
OR	Used within an IF statement (Optional)
ELSE	Used within an IF statement (Optional)
ELSEIF	Used within an IF statement (Optional)
THEN	Used within an IF statement (Optional)
END	Ends an IF statement
WHILE...WEND	Executes a series of statements, as long as a given condition is true
BREAK	Used to exit a While...Wend statement block
CONTINUE	Restarts a While...Wend statement loop
GOTO	Jumps to a label within a calculation
RETURN	Tells the calculation to return a result
CALL	Temporarily calls another calculation file
CHAIN	Permanently calls another calculation file

These statements are explained in the following topics.

RETURN statements

A RETURN statement directs the calculation to return with or without a value. A RETURN statement must begin with the keyword RETURN. A RETURN statement may return the result of the calculation to be placed in the field that initiated the script.

A RETURN statement is also used to return results to one calculation script from another. Using a CALL statement temporarily suspends the current script calculation and sends control to another script file. A RETURN statement sends control back to the original script which may then continue processing. See CALL statements on page 25 for more information. Here are some sample RETURN statements:

```
RETURN(@("LAST_NAME") & ', ' & @("FIRST_NAME") & " " &
@("MIDDLE_INIT"))
```

RESULT: Takes the data in the section variable field LAST_NAME adds a comma; adds the data in the section variable field FIRST_NAME; adds the data in the section variable field MIDDLE_INIT and places this data in another section variable field.

```
RETURN (CALL('FirstFile'))
```

RESULT: Returns the result of the calculation generated by calling the script FirstFile.

IF statements

An IF statement is executed based on the occurrence of a certain condition. IF statements must begin with the keyword IF and terminate with the keyword END.

Components within IF statements can be connected with the keywords AND or OR. IF statements can have three forms: a simple IF statement, an IF statement with an ELSE condition, or an IF statement with an ELSEIF condition.

- Simple IF Statement

  A simple IF Statement contains a single statement block. The calculation is performed only if the logical expression is true. If the logical expression is false, control passes to the next statement after the END keyword. Here is an example:

  ```
  IF (@("FirstAmount") < 1000.00)  THEN
       $FinalAmount = @("FirstAmount") * .05;
  END;
  RETURN ($FinalAmount)
  ```

  CALCULATION: If the value of the section variable field FirstAmount is less than 1000.00 then the value is multiplied by .05 and entered in the target variable $FinalAmount. The value of the $FinalAmount target variable is then returned to the section variable field.

- Use of the keyword connector THEN is optional.
- IF Statement with ELSE Condition

  An IF Statement with an ELSE condition contains an alternative calculation. If the logical expression is false, control passes to the statement after the ELSE keyword.

Here is an example:

```
IF (@("FirstAmount") < 1000.00) THEN
$FinalAmount = @("FirstAmount") * .05;
ELSE
$FinalAmount = @("FirstAmount") + 10.00;
END;
RETURN ($FinalAmount)
```

CALCULATION: If the value of the section variable field FirstAmount is less than 1000.00 then the amount is multiplied by .05 and entered in the target variable $FinalAmount.

However, if the value of the section variable field FirstAmount is greater than or equal to 1000.00 then 10.00 is added to the amount and entered in the target variable $FinalAmount.

The value of the $FinalAmount field is then returned to the caller or section variable field.

Use of the keyword connector THEN is optional.

- IF Statement with ELSEIF Condition

  An IF statement with an ELSEIF condition is the most complicated type of IF statement. If the first logical expression is true, the statement block after IF is executed until the first ELSEIF statement is reached. If the first logical expression is false, the first ELSEIF logical expression is evaluated. If the ELSEIF logical expression is true, the statement block from the ELSEIF to the next ELSEIF (or ELSE) is executed. If the ELSEIF statement is false, the next ELSEIF is evaluated. If all logical expressions are false, control passes to the ELSE block. If there is no ELSE block, control passes to the statement following the END keyword.

  An ELSEIF statement is considered part of the same IF statement. Only one END keyword is needed to end an IF, ELSEIF, ELSE statement. IF statements can be nested inside other IF statements. A nested IF statement requires its own END keyword. A missing or mismatched keyword results in a runtime syntax error. Here is a sample IF statement with ELSEIF condition:

```
IF (@("FirstAmount") < 1000.00)
$FinalAmount = @("FirstAmount") * .05;
ELSEIF @("FirstAmount") < 5000.00
$FinalAmount = @("FirstAmount") * .03;
ELSEIF @("FirstAmount") < 10000.00
$FinalAmount = @("FirstAmount") * .02;
ELSE
$FinalAmount = @("FirstAmount") + 10.00;
END;
RETURN ($FinalAmount)
```

CALCULATION: If the value of the section variable field FirstAmount is less than 1000.00 then the amount is multiplied by .05 and entered in the target variable $FinalAmount.

If the value of the section variable field FirstAmount is greater than or equal to 1000.00 but less than 5000.00 then the amount is multiplied by .03 and entered in the target variable $FinalAmount.

If the value of the section variable field FirstAmount is greater than or equal to 5000.00 but less than 10000.00 then the amount is multiplied by .02 and entered in the target variable $FinalAmount.

If the value of the section variable field FirstAmount is greater than or equal to 10000.00 then 10.00 is added to the amount and entered in the target variable $FinalAmount.

The value of the $FinalAmount field is then returned to the caller or section variable field.

GOTO statements

A GOTO statement moves to a specific location within a calculation. The location has been named with a label. (See Labels on page 22 for more information.) A GOTO statement must begin with the keyword GOTO. Here is an example:

```
GOTO SECTION_ONE:
```

RESULT: The control jumps to SECTION_ONE in a calculation.

The destination label can occur anywhere in the script containing the GOTO statement. If the label cannot be located in the script, a syntax error will be generated.

GOTO will support retrieving the label from a target variable. Here is an example:

```
SECTION = "MY_LABEL:"
GOTO SECTION
```

RESULT: Since the word following the GOTO statement does not contain a colon, the program will assume the label is contained in the target variable named. In this case, control will jump to the location of MY_LABEL in the current script.

CALL statements

A CALL statement temporarily suspends one calculation and calls another calculation file. A CALL statement must begin with the keyword CALL. The calculation file that is called must contain a RETURN statement if the original calculation expects a returned value. Here is an example:

```
CALL( 'TestCalc' )
```

RESULT: Temporarily calls the calculation file TestCalc. After the calculations in TestCalc are completed, processing returns to the current script. In this example, TestCalc is not expected to return a value.

CHAIN statements

A CHAIN statement permanently calls another calculation language file. A CHAIN statement must begin with the keyword CHAIN. There is no limit to the number of CHAIN statements that can be used. Here is an example:

```
CHAIN 'LastCalc'

or
CHAIN( 'LastCalc' )
```

RESULT: Permanently calls the calculation file LastCalc. Processing does not return to the current script. No statements from the original script will be evaluated after the CHAIN statement.

Using While…Wend Statements

Use While...Wend statements to execute a series of statements, as long as a given condition is true.

```
While condition
[statements]
Wend
```

Parameter	Description
Condition	Required. The condition is any expression that evaluates to true or false. False is assumed to be a zero value. Any non-zero value is assumed to be true.
Statements	One or more statements executed while the condition is true.

If *condition* is true, the statements within the While block are executed. When the Wend statement is encountered, control returns to the While statement and *condition* is again evaluated. If *condition* is still true, the process repeats. If it is false, execution resumes with the statement which follows the Wend statement.

You can nest While...Wend loops to any level. Each Wend matches the most recent While.

NOTE: Keep in mind that you can start an endless loop if you specify a condition that can never be satisfied. The system cannot syntactically detect an endless loop, so if you create one, the program will lock up and you will have to kill the program.

(Ellipses in the following examples represent additional statements, not shown.)

```
While(10 > #value)
    ...
    While (#new = 1)
        ...
    Wend
    ...
Wend
```

You do not have to use tabs to indent nested While...Wend statements. Tabs are used in these examples, to help identify statement blocks. You may want to also use tabs in your code to make the source easier to read.

Break statements

Break statements provide a way to exit a While...Wend statement block.

```
Break
or
Break(levels)
```

Parameter	Description
Levels	The value you enter defines how many nested While...Wend statement blocks you want to terminate.If you omit this parameter, control passes to the statement following the next Wend statement encountered.

You can only include Break statements inside While...Wend statement blocks. Break statements transfer control to the statement following the Wend statement.

When used within nested While...Wend statements, you can include the Levels parameter to transfer control to the statement following the Wend level you specify.

Here are some examples. (Ellipses in the following examples represent additional statements, not shown.)

```
While(1)
    ...
    While (2)
        ...
        Break
    Wend
    ...
Wend
```

In this example, the Break statement only terminates the While...Wend which contains the statement. Control passes to the first (outside) While...Wend statement block.

Here is another example:

```
While(1)
    ...
    While (2)
        ...
        While(3)
            ...
            Break(3)
        Wend
        ...
    Wend
    ...
Wend
```

In this example, the Break(3) statement terminates all three While...Wend blocks that are active.

Continue statements

Use Continue statements to restart a While...Wend statement loop.

```
Continue
```

Executing the Continue statement stops the current sequence of statement execution and restarts program flow at the beginning of the loop. This causes the While statement to retest the condition and, if true, execute the loop again.

Statements after the Continue keyword are not executed. Continue is often, but not always, activated by an IF test. Here is an example:

(Ellipses in the following examples represent additional statements, not shown.)

```
While(#x < 10)
    ...
    If (value)
        Continue
    End
    ...
Wend
```

GOTO statements

GOTO statements have not changed with the implementation of the While loops, but note that you can use GOTO statements to jump into or out of a While loop.

When jumping into a While loop, you bypass the check of the While condition. The condition is not checked until a Continue or Wend statement is encountered. If the While condition is true, you stay in the loop. Otherwise, control moves to the next statement following the Wend for that loop.

If a GoTo statement is encountered within a While...Wend loop, control passes to the location of the destination label named. This label may be in or outside the control of the While statement.

BeginSub and EndSub

BeginSub and EndSub are keywords, but not Flow Control statements. You will only see these keywords when loading a DAL script library (a library of DAL subroutines). They designate the start and end of a subroutine. You will not see them in the normal *flow* of script execution.

BeginSub

Use BeginSub to begin each subroutine in a DAL subroutine library.

Syntax

```
BeginSub (Name)
```

Once a DAL library is loaded, you can reference the scripts contained in the library by name. You do not have to CALL or CHAIN to the script.

Parameter	Description	Required
Name	Name associated with the subroutine	Yes

BeginSub and EndSub must be paired per script. You must have a space between BeginSub and the script name.

Example

```
BeginSub SCRIPT1
*       This script returns #x set to 2 if #x was equal to 1 on enter.
IF (#x = 1) THEN #x = 2;
END;
RETURN (#x);
EndSub

BeginSub Script2
*       This script returns a negative one if #y was equal to 5.
if(#y = 5) then Return (-1);
end;
EndSub
```

SCRIPT1 is the name of the first script and *Script2* is the name of the second script.

NOTE: *SCRIPT1* and *Script2* are only names, you can use any name you want as long as the name *is not* a DAL reserved function, statement, or key word such as CALL, FIND, IF, and so on. You can mix case in script names.

EndSub

Use this function to end each subroutine in a DAL subroutine library.

Syntax

```
EndSub ( )
```

Parameter	Description
None	No parameters are necessary for this function.

BeginSub and EndSub must be paired per script.

Example

Here is an example:

```
BeginSub SCRIPT1
*       This script returns #x set to 2 if #x was equal to 1 on enter.
IF (#x = 1) THEN #x = 2;
END;
RETURN (#x);
EndSub

BeginSub Script2
*       This script returns a negative one if #y was equal to 5.
if(#y = 5) then Return (-1);
end;
EndSub
```

Script1 is the name of the first script. *Script2* is the name of the second one.

DATA STORAGE STATEMENTS

Data storage statements return the results of the calculation to the variable field that initiated the script or stores the results in the variable field you specify.

You use *keywords* for storing data. Keywords define the statement operations and are reserved for use in the calculation language. You cannot use these keywords in variable field names. Keywords are not case sensitive.

NOTE: Keywords are the only way to return or store data results in a variable field.

Keyword	Action
Return	Directs a calculation to return with or without a value to the variable field that initiated the script. Returns target variable results to a DAL script from another DAL script (see CALL statements on page 25); sends control back to the original script.
SetFld	Assigns a value or the results of a calculation (target variable) to a variable field on a section. The variable field maybe on any section or form in the form set
AppendText	Attaches text to the end of a multi-line text variable field from an external ASCII text field.
AppendTxm	Attaches text to the end of a multi-line text variable field from the first text area field found on a section you specify.
AppendTxmUnique	Attaches text to the end of a multi-line text variable field from the first text area field found on a section you specify. Also renames any embedded variable field imported from the external text area. Embedded variable fields will then have a unique name.

Testing DAL Scripts

You can use the DALRUN utility to test scripts and trigger the interactive DAL Debugger. Debug messages, certain errors, and a dump of the symbol table at the end of the run are examples of output this utility will generate.

Syntax

```
DALRW32 /X /INI /D  /T
```

Parameter	Description

/X	This optional parameter supplies the name of a script to run. If you omit this option, you can use this INI option to provide the name of the script:
	`< DALRun >`
	`    Script = file name`
	You can use any extension. The default is *DAL*.

/INI	This optional parameter supplies the name of an INI file to load. This INI file supplies additional parameters and options. If the DALRUN.INI file is present, the utility loads it by default.
	Here are the INI options you can include in the INI file:
	`< DALRun >`
	`    Title = title string(an override to the window title)`
	`    Script = file name (the script to run)`
	`< DALFunctions >`
	`    Keyword = DLLMOD->FunctionName`
	`    Keyword2 = DLLMOD->FunctionName2`
	`    (and so on)`

/D	The debug switch starts the DAL Debugger. When on, the script executes in single step mode and registers this DAL function: *DEBUG("message")*. The DEBUG function breaks execution, displays a message, and invokes the debugger in single step mode.
/T	This parameter sends certain text messages to the standard output device. These messages are not visible at runtime, but may be redirected when you run this utility.

Here is an example:

```
DALRW32 /ini=test  /d  /t  > test.txt
```

This example tells the system to run the DALRUN utility using the TEST.INI file. The /D parameter tells the system to start the DAL debugger. The /T parameter tells the system to send messages to a file named TEST.TXT.

USING THE DAL DEBUGGER IN DOCUMAKER WORKSTATION

You can enable the DAL Debugger in Documaker Workstation by adding the following lines to the MEN.RES file in your master resource library (MRL). You can edit this file using any ASCII text editor. Before you edit the file, make a backup copy. Here is an example of what you need to add to the MEN.RES file:

```
POPUP        "&Tools" 255 "Utility Programs"
    BEGIN
    MENUITEM "Enab&le Debugger..." 502 "DBGW32->DBGEnableDebugger"
"Enable DAL debugger." 0
    SEPARATOR
```

Runtime Error Messages

Use the following table to resolve any error messages you may receive.

Message	Number	Description
Out of memory	1	The calculation needs more memory than is available. Make more memory available to the program and try again.
Open failure on script file	2	The file containing the calculation cannot be opened. This may mean the file does not exist; is protected from reading; or that the file is not located in the default directory established by your INI file option. The default directory is usually DefLib.
Syntax error	3	A calculation contains invalid information or does not use proper statement syntax.
Wrong number of parameters	4	A built-in function or procedure requires more parameters than are provided.
Wrong type of parameter	5	A built-in function or procedure expects a particular type of parameter. This may mean that the variable type used is not automatically converted to the type required by the routine.
Invalid or unknown symbol	6	A character (or set of characters) does not correspond to a known operator or keyword. Can also indicate that you need to add a Return statement.
Invalid assignment statement	7	The assignment statement fails to provide a valid source expression or destination variable.
Cannotmodify target	8	A statement attempted to change the value of an identifier that cannot be changed.
Unexpected internal error	9	A calculation caused an unexpected error or event that cannot be corrected.
Missing/ mismatched parenthesis	10	The number of open parentheses does not match the number of close parentheses.
Invalid IF statement	11	An IF statement contains or fails to contain a keyword.
Unexpected end of script	12	The end of the script occurred before the current statement could be fully evaluated. This may be due to the script being incomplete or an inability to read the entire script.
Invalid expression syntax	13	Generates due to a number of problems, such as: an expression fails to yield a result or encounters an unknown variable type.
Attempt to divide by zero	14	An attempt to divide a value by zero was found. Division by zero is undefined and must be avoided.

Message	Number	Description
No result value returned	15	An expression expects a return value when calling a procedure. Only functions can return values. This error may also result if a RETURN statement is missing from a file that has been invoked with a CALL statement.
Statement label already used	16	Another label with the same name has been found within the script.
Unknown statement label	17	A GOTO statement names a label that does not occur within the script.
Invalid statement label	18	An invalid label was found.
Illegal label location	19	A GOTO statement attempted to locate a label within an IF statement. A GOTO statement can jump from an IF statement, but not into an IF statement.
Function out of place	20	A function was called but the statement does not expect a return value. Since a function must return a value, the call must be an error.
Illegal parameter value	21	A built-in function or procedure passed a parameter value that is not valid.

DAL Script Examples

Here are some DAL script examples you can refer to as you create your own DAL scripts.

Preparing AFP or Metacode print streams for Docusave

This example shows DAL scripting which you could use to format and configure an AFP or Metacode print stream for storage using Docusave.

The FSISYS.INI or FSIUSER.INI files must contain these options:

```
< PRTType:xxx >
    OutMode                 = MRG4 or JES2
    DocuSaveScript          = DOCUSAVE.DAL
```

Where *XXX* is either AFP or XER. For the OutMode option, enter *MRG4* or *JES2*. Enter the name of the script in the DocusaveScript option.

The DOCUSAVE.DAL script file should contain this information:

```
* Add Docusave Comment - use default: APPIDX record!
comment = AppIdxRec(  )
class   = PAD("bio",8)
cabinet = PAD("rpex7",8)
title   = PAD("TITLE",22)
indextag= comment & class & cabinet & title
Print_It (indextag)
AddDocuSaveComment (indextag)
Return ('FINISHED!')
```

Preparing PCL print streams for Docusave

To add Docusave comments to an PCL print stream, add the DocusaveScript option and the name of a DAL script to execute. The DAL script should call the AddDocusaveComment function to add a string as a Docusave comment record. Here is an example:

```
< PrtType:PCL >
    DocusaveScript = DOCUSAVE.DAL
```

Here is an example of what the DOCUSAVE.DAL file might look like:

```
* Add Docusave Comment - use default: APPIDX record!
COMMENT = AppIdxRec()
PRINT_IT(COMMENT)
ADDDOCUSAVECOMMENT(COMMENT)
RETURN('FINISHED!')
```

Preparing AFP print streams for IBM's OnDemand

This example shows DAL scripting which you could use to format and configure an AFP print stream for storage using OnDemand. Keep in mind...

- The AFP Conversion and Indexing Facility (ACIF), which is an IBM product, writes some AFP structures such as Tag Logical Element (TLEs) in an AFP print stream.
- Oracle Insurance's comment support for AFP does not use TLEs. It was designed for OnDemand.
- The system uses the D3EEEE AFP structure, also known as a NOP (No-Operation) structure.

The FSISYS.INI or FSIUSER.INI files must specify the name of the DAL script in the OnDemandScript option:

```
< PrtType:AFP >
```

```
OnDemandScript =  ONDEMAND.DAL
```

The ONDEMAND.DAL script file should contain this information:

```
*    Make sure #loadlib is initialized
#loadlib =  #loadlib
* Load script into cache memory!
If (#loadlib = 0) Then
        LoadLib('OnDmdLib')
End
#loadlib+= 1
* Execute script!
OnDemand( )
Return('FINISHED!')

OnDmdLib.DAL script library file

BeginSub OnDemand

* OnDemand Script is only valid for AFP print streams!

If (PrinterClass() != 'AFP') Then
        Return
End

* Example of reading GVM variables
* If (HaveGVM('Company') Then
*             company = GVM('Company')
* End

* Make sure #docnum is initialized

#docnum = #docnum
If (#docnum = 0) Then
        semi= ';'
        colon = ':'
        acifinfo = 'ACIFINFO'
        docnum = 'DOCUMENT_NO'
        mvsfile= 'MVS_FILENAME'
        expbprep = 'EXPBPREP'
        procdate = 'PROCESS_DATE'
        proctime = 'PROCESS_TIME'
        idxname = 'ACIF_INDEX_NAME'
        idxdata = 'ACIF_INDEX_DATA'
        recid = 'RECID=470'
        grpname = GroupName( )
        dapver = MajorVersion( ) & '.' & MinorVersion( )
        Print_It ('DAP Version is ' & dapver)
End

*   Add comment, ' ACIFINFO;DOCUMENT_NO:0000001'

#docnum += 1
AddComment (acifinfo & semi& docnum & colon &
Format(#docnum,'n',9999999))

*   Add comment, 'MVS_FILENAME:PROD.EX.P.DCS.AFP.PREPOUT'
```

```
AddComment (mvsfile & colon & 'PROD.EX.P.DCS.AFP.PREPOUT')

*   Add comment, 'EXPBPREP;PROCESS_DATE:mm-dd-yyyy'

AddComment (expbprep & semi & procdate & colon & Date('1-4'))

*   Add comment, 'EXPBPREP;PROCESS_TIME:hh:mm:ss'

AddComment (expbprep & semi& proctime & colon & TIME())

* Add comment, 'RECID=470;ACIF_INDEX_NAME01;026;Correspondence Copy
Number'
* Add comment, 'RECID=470;ACIF_INDEX_DATA01;009;840127920'

#idxnum = 1
fldname = 'Correspondance Copy Number'
flddata = '840127920'
AddComment (recid & semi& idxname & Format (#idxnum,'n',99) & semi & \
    Format (Len (fldname),'n',999) & semi & fldname)
AddComment (recid & semi& idxdata & Format (#idxnum,'n',99) & semi & \
    Format (Len (flddata),'n',999) & semi& flddata)

* Add Comment, 'RECID=470;ACIF_INDEX_NAME02;019;Correspondance Type'
* Add Comment, 'RECID=470;ACIF_INDEX_DATA02;025;Notice of Initial
Reserve'

#idxnum += 1
fldname = 'Correspondance Type'
flddata = 'Notice of Initial Reserve'
AddComment (recid & semi& idxname & Format (#idxnum,'n',99) & semi&\
    Format (Len (fldname),'n',999) & semi& fldname)
AddComment (recid & semi& idxdata & Format (#idxnum,'n',99) & semi&\
    Format (Len (flddata),'n',999) & semi& flddata)

* Get DAP Field - 'INSURED NAME'
* Add Comment, 'recid=470;ACIF_INDEX_NAME03;012;INSURED NAME'
* Add Comment, 'recid=470;ACIF_INDEX_DATA03;008;John Doe'

If (HaveField('INSURED NAME',,,grpname)) Then
        #idxnum += 1
        fldname = 'INSURED NAME'
        flddata   = @(fldname,,,grpname)
        AddComment (recid & semi& idxname & Format (#idxnum,'n',99)
& semi&\
                Format (Len (fldname),'n',999) & semi & fldname)
       AddComment(recid & semi& idxdata & Format (#idxnum,'n',99) &
semi&\
                Format (Len (flddata),'n',999) & semi & flddata)
End

Return
EndSub
```

Chapter 2

Function Reference

Numerous functions are built into the DAL calculation language. These functions let you apply operations to form set objects, to previously calculated target variables, to constants, or to any combination of the three. The functions fall into these categories:

- Locating Objects on page 94
- Where DAL Functions are Used on page 97

Some functions may be applicable to more than one category. Each function, however, will only be discussed once in the category that best describes it.

Each category has a table listing the functions. The table lists and briefly describes each function. Use the table to quickly scan the available functions. Each function is discussed in detail in alphabetical order at the end of this chapter.

OVERVIEW

Functions and procedures and their and associated parameters must be written in this syntax:

```
FUNCTION( parameters )
```

Many functions return a value the script may use in some fashion. For instance, the following statements each use the value returned from a function:

Statement	This statement...
IF (FUNCTION( )) then ... END	Shows the returned value used in the logical evaluation of the IF statement. If the returned value is non-zero, the IF statement is TRUE. If the value is zero, the IF will evaluate FALSE.
Y = FUNCTION( );	Demonstrates assigning another variable the result returned from a function.
Y = FUNCTION(FUNCTION2( ));	Is similar to the last, except it also demonstrates the use of a function's return value as a parameter to another function.
$VAL = 17.00 / FUNCTION( );	Demonstrates the use of a returned value as an operand in a mathematical expression.

Some functions do not return a value and simply perform some operation and return. These types of functions are often referred to as *procedures* to distinguish them from those functions that do return values. If a function does not return a value, using it in one of the above described manners causes a syntax error.

Sometimes a function may behave as either a function or procedure. For these functions, if they are used in one of the manners shown, a result will be returned. If called in a manner that does not expect a result, none will be returned.

Please note however, for those functions that must return a value, you are required to use the result in one of the above described manners or a syntax error will be generated.

Each function description identifies any required or optional return value.

NOTE: The SAMPCO sample resources contain a great number of DAL examples and explanations. Be sure to check out this resource as you create DAL scripts for your company.

Bit/Binary Functions

The Bit/Binary functions are summarized in the table below. These functions allow bit manipulation within integers. Click on the function name to jump to a discussion of that function.

Function	Result
BitAnd	Returns the result of a bitwise AND operation performed on two numeric values.
BitClear	Returns the result after clearing the specified bit in a value.
BitNot	Returns the result of a bitwise logical NOT operation performed on a numeric value.
BitOr	Returns the result of a bitwise inclusive OR operation performed on two numeric values.
BitRotate	Returns the result of a bit shift-and-rotate operation performed on a numeric value.
BitSet	Returns the result after setting the specified bit on in a value.
BitShift	Returns the result of a bit logical shift operation performed on a numeric value.
BitTest	Returns TRUE (1) if the specified bit in a value is a 1; otherwise FALSE (0) is returned.
BitXor	Returns the result of a bitwise exclusive OR operation performed on two numeric values.
DashCode	Creates a value to assign to a series of fields from the binary value of an integer.
Dec2Hex	Returns the hexadecimal equivalent of an integer value.
Hex2Dec	Returns the integer equivalent of a hexadecimal string.

DATABASE FUNCTIONS

Database functions perform tasks using databases. By default, all database styles recognized by the system are supported. A typical use of these functions is to reference tables created for ODBC in Windows and DB2 (DB2/2). The functions you can use are listed below. Click on the function name to jump to a discussion of that function.

Function	Result
DBAdd	Adds a record to an open database table. Optionally returns one (1) on success or zero (0) on failure.
DBClose	Closes an open database table. Optionally returns one (1) on success or zero (0) on failure.
DBDelete	Deletes a record from a database table. Optionally returns one (1) on success or zero (0) on failure.
DBFind	Retrieves a record by key value from an open database table. Optionally returns one (1) on success or zero (0) on failure.
DBFirstRec	Retrieves the first record from an open database table. Optionally returns one (1) on success or zero (0) on failure.
DBNextRec	Retrieves the next record from an open database table. Optionally returns one (1) on success or zero (0) on failure.
DBOpen	Opens a database table. Optionally returns one (1) on success or zero (0) on failure.
DBPrepVars	Creates the DAL variables associated with a table record.
DBUnloadDFD	Streamlines the use of DAL with ODBC and memory tables by creating DFD files and using only memory tables
DBUpdate	Updates a record retrieved from a database table. Optionally returns one (1) on success or zero (0) on failure.

The functions are generic for any supported database including ODBC (Open Data Base Connectivity) compliant databases and DB2/2 compliant databases.

NOTE: The customer is responsible for licensing and installing the desired database product and any required operating system driver.

All database access is routed through the system's database library DLL. This DLL handles interfacing with the supported types of databases. Each database type has an associated database handler. Database handlers can be described in an INI control group which begins with *DBHANDLER*: followed by the database handler name, such as:

```
< DBHandler:ODBC >
```

ODBC Handler

The standard handler name for ODBC is *ODBC*. Here is an example:

```
< DBHandler:ODBC >
    Install =   SQW32->SQInstallHandler
or
    InstallMod= SQW32
    InstallFunc= SQInstallHandler
```

The Install option specifies the DLL module name and handler function name. This function is linked dynamically when the handler is initialized. Actually, the above definitions are not necessary for ODBC support. The database library will default the module and function name to the values shown.

Additional values can be optionally set in the INI file.

```
Server =  Server name (default is "MS SQL Server")
```

The Server option relates to an ODBC term which is specified on the control panel which essentially provides the name of a driver. *MS SQL Server* is the default if the option is omitted.

```
Qualifier  =  Qualifier(no default)
```

The Qualifier option provides data source specific information, for example, the database name for an Access database.

```
User=  User ID (no default)
PassWd= User password(no default)
```

The User and PassWd (password) options provide a way to automatically log on to the database. Not all drivers support this usage. When unspecified, some ODBC drivers may display a logon window and prompt for the information. Some drivers will ignore the options if the connected database manager does not require or support logging in.

```
CreateIndex=Yes / No(default is Yes)
CreateTable=Yes / No(default is Yes)
```

The CreateTable and CreateIndex options can be used to prevent time delay while a table is checked for existence. In this way, the normal capabilities of the connected driver may be overridden. When set to No, any attempt to open the file with a mode of *CREATE_IF_NEW* will automatically be rejected. Some drivers may not support creating a table or index, and may require these options to be set to No.

DB2/2 Handler

The database handler for DB2 is defined in a similar manner to that described for ODBC. The following INI options are valid for installing the DB2 handler.

```
< DBHandler:DB2 >
    Install = DB2W32->DB2InstallHandler
or
    InstallMod = DB2W32
    InstallFunc = DB2InstallHandler
```

The Install option specifies the DLL module name and handler function name. This function is linked dynamically when the handler is initialized. These INI options are not necessary for tables specifying DB2 as the database type. DB2 is also supported via static linking under z/OS, and currently only version 3.1 has been tested in that environment.

Here are other INI options that can be specified for DB2.

```
Database = Database name(no default)
```

The Database option specifies the name of the database and is required.

```
Bindfile = Bind file name(no default)
```

The Bindfile option specifies the name of a *bind* file which provides the bound access plan for the database. The DB2LIB.BND file is provided with the system's DB2LIB and can be used as a bind file.

CREATING A DATABASE HANDLER FOR AN EXCEL DATABASE

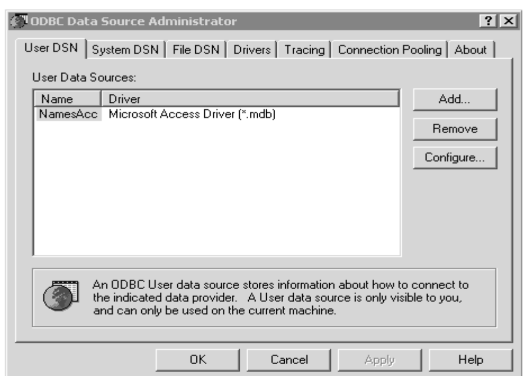
You define a database handler for a Microsoft Excel database in a similar manner to that described for an ODBC database. The following INI options are used to install the Excel handler to access a database defined as part of an Excel spreadsheet. The handler name in this example is *NamesExcel*.

```
< DBHandler:NamesExcel >
   Class = ODBC
   Server = NamesExc
```

The Class option tells the DAL database handler what type of driver to use. Enter **ODBC**. This option is required.

The Server option specifies the user data sources name as shown on the ODBC Data Source Administrator window. This name specifies the ODBC driver to be used as the data source. The default drive is MS SQL Server.

This example shows how to add a user data source which is an Excel database named *NamesExc*. NamesExc is defined in an Excel spreadsheet entitled *Names*. The user data source name, NamesExc, is assigned to use the Microsoft Excel (ODBC) Driver (*.xls).



To add a new data source name, follow these steps:

1. Click the Add button and select the ODBC driver to use. Then click Finish.
2. Enter the desired Data Source Name and description. You can enter up to 22 characters for the data source name.
3. Click the Workbook Selection button and select the path for the database. Then click Ok.

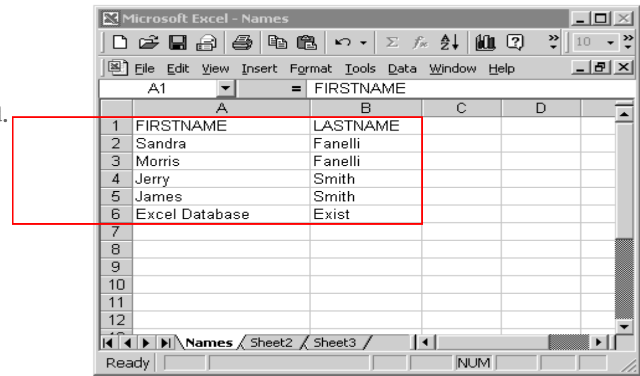

Create New Data Source
Select a driver for which you want to set up a data source.
Name
Microsoft Excel Driver (*.xls)
Microsoft Excel-Treiber (*.xls)
Microsoft FoxPro VFP Driver (*.dbf)
Microsoft ODBC for Oracle
Microsoft Paradox Driver (*.db)
Microsoft Paradox-Treiber (*.db)
Microsoft Text Driver (*.txt; *.csv)
Microsoft Text-Treiber (*.txt; *.csv)
Microsoft Visual FoxPro Driver
ODBC Microsoft Excel Setup
Data Source Name: NamesExc
Description: Names Tabel In Excel Spreadsheet - Names.xls
Database
Version: Excel 97-2000
Workbook: C:\...\DALDB\Names.xls
Select Workbook...
Use Current Directory
OK
Cancel
Help
ODBC Data Source Administrator
User DSN
System DSN
File DSN
Drivers
Tracing
Connection Pooling
About
User Data Sources:
Name
Driver
NamesAcc
Microsoft Access Driver (*.mdb)
NamesExc
Microsoft Excel Driver (*.xls)
Add...
Remove
Configure...
An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you, and can only be used on the current machine.
OK
Cancel
Apply
Help

Here is an example of the steps you would follow to define a database in an Excel spreadsheet.

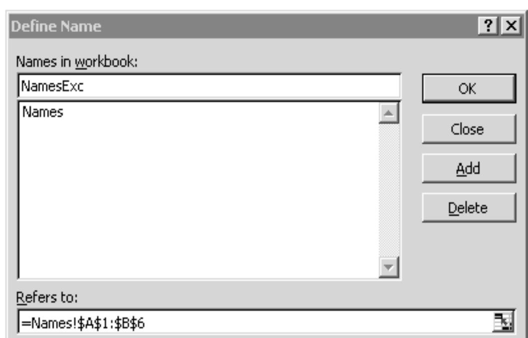
1 Enter the field names in the first row of each column that make up the table. Then enter the data in each column.

2 Select the columns and rows that comprise the table.

These rows and columns are selected or highlighted.



3 Choose the Insert, Name, Define option. Then enter the name of the table on the Define Name window and click Add.



4 Define the name of the worksheet and save it.

Associating Tables with Handlers

You can describe database tables in an INI control group which begins with *DBTable:* followed by the database table name. The database table section associates attributes specific to the table. Here is an example:

```
< DBTable:AppIdx >
    DBHandler = ODBC
```

The DBHANDLER option allows a database table to be mapped by name to the appropriate database handler. No other table-level options are defined at this time.

The system supports multiple simultaneous ODBC connections via different ODBC drivers. This lets you, for instance, connect at the same time to multiple:

- Databases on an SQL server

- Databases on an SQL server and Excel spreadsheet databases
- Access databases and Excel spreadsheet databases
- Access databases
- Excel spreadsheet databases
- Databases for which you have an ODBC-compliant driver

The system does not support multiple different DB2 databases using native DB2 drivers. Support is limited to ODBC-compliant data bases.

Accessing Database Fields

Usually the information in a database table is logically divided into records. These records typically contain one or more components called fields. In DAL, record fields will be associated together via a common DAL variable prefix name. Ability to access individual data elements is supported by using a dot (".") operator.

Here is an example:

Assume a table contains records with three fields:

- LOANTYPE
- PAYMENT
- DUEDATE

In the script you will designate a prefix name for these variables when using the database functions. So you could end up with something like:

```
RECORD.LOANTYPE
RECORD.PAYMENT
RECORD.DUEDATE
```

Each field from the same record will have the same prefix name (which you can assign) concatenated with the dot operator.

Setting Up Memory Tables

Memory tables are useful when a program needs to create a temporary database table for a fast search, sort, or sequential access, such as with DAL scripts with DALDB. For instance, you create a few database tables from the input extract XML file for easier mapping and searching if those tasks were taking too long.

To tell the system to open a memory table in a DAL script, include the MEM or MEMORY parameter as the database type. This is the second parameter of DBOpen function. Here is an example:

```
rc=DBOpen("table1","MEM","d:\deflib\appidx.dfd","READ & WRITE");
```

Keep in mind that since the tables are in memory, they go away once the program terminates and the data is lost. DFD files are required to use memory tables since those tables are not self-describing.

When you use a memory table with either a DAL script that did not specify the MEM parameter or with some other kind of table, include one of these INI options to tell the system the table will be using memory:

```
< DBTable:XXX >
    DBHandler = MEM
```

or

```
< DBTable:XXX >
    DBHandler = MEMORY
```

To keep the table in memory after the DBClose call, include this INI option:

```
< DBTable:XXX >
    Persistent = Yes
```

Keep in mind, in this case table memory is released only when the program terminates. Use carefully to make sure you do not run out of memory.

Date Functions

Date functions perform specific operations regarding date information. These functions enter or alter a date in a particular manner. The date functions are summarized in the table below. Click on the function name to jump to a discussion of that function.

Function	Result
Date	Returns a date string or the current date.
Date2Date	Converts one date format to a new format and returns the result.
DateAdd	Adds days, months, and years to the date and returns the result.
DateCnv	Converts a date specified with a two-digit year into a date containing a four-digit year value.
Day	Returns the day of the month number from a date and returns the result.
DayName	Returns the specified day name.
DaysInMonth	Returns the number of days in the specified month and year.
DaysInYear	Returns the number of days in the specified year.
DiffDate	Calculates the difference between two dates and returns a positive or negative value based on which date is earlier.
DiffDays	Returns the difference in days between two dates.
DiffMonths	Returns the difference in months between two dates.
DiffYears	Returns the difference in years between two dates.
LeapYear	Returns one (1) if the specified year is a leap year and zero (0) if it is not a leap year.
Month	Returns the month number from a date.
MonthName	Returns the specified month name.
WeekDay	Returns the week day number from a date.
Year	Returns the year from a date.
YearDay	Returns the number of the day of the year from a date.

Before we examine each date function individually you must understand the available date formats. Date formats are usually one of the parameters you enter for a date function. The date format determines how your date information appears when it is returned to the section assigned to a target variable.

DATE FORMATS

Date formats consist of these components, placed inside quotation marks, in this order:

```
(Format type)(Separator)(Year size)(Case)(Locale)
```

Parameter	Description
Format type	1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, or X. You must include a format type if you want to specify a separator, a year size, or a locale. The default is 1 (one). See Date format types on page 53 for a list of the various format types.
Separator	For the separator character, you can enter a backslash (/), a dash (-), a period (.), a comma (,), or B (or b), which indicates a blank space. You should only enter separator characters for format types which include separators (see the table of format types below). If the format type does not include separators, such as format type C, the system ignores any separator character you enter. The default separator is a backslash (/).
Year size	For the year size, you can specify either 2 (09) or 4 (2097) to indicate a two- or four-digit year. Use four-digit years. DAL functions use a four-digit year unless the format or the input data specifies otherwise. For example, if you enter 1/2, you specify date format 1 and a two-digit year, such as 02/17/09.
Case	(Optional) To return an uppercase date, such as FEBRUARY 17, 2009, include this character before the Locale: > To return a lowercase date, such as february 17, 2009, include this character before the Locale: < For a mixed case date, such as February 17, 2009, omit this parameter.
Locale	For DAL functions, you can enter an additional component to specify the locale. This is done with @xxx, where xxx indicates the locale. You must include the @, or the system ignores the locale code (xxx). US English (USD) is the default. See Locales on page 55, for a list of locale codes.

NOTE: Date formats are also used in the variable field properties. If you try to use DAL to place a formatted date value into a variable field with a different date format, the system will try to convert the date to the proper format. This can result in an incorrect value and may cause an error message if it cannot be converted.

Date format types

Format	Date order	Description
1	MM/DD/YY	Month-Day-Year with leading zeros (02/17/2009)
2	DD/MM/YY	Day-Month-Year with leading zeros (17/02/2009)
3	YY/MM/DD	Year-Month-Day with leading zeros (2009/02/17)
4	Month D, Yr	Month name-Day-Year with no leading zeros (February 17, 2009)
5	M/D/YY	Month-Day-Year with no leading zeros (2/17/2009)
6	D/M/YY	Day-Month-Year with no leading zeros (17/2/2009)
7	YY/M/D	Year-Month-Day with no leading zeros (2009/2/17)
8	bM/bD/YY	Month-Day-Year with spaces instead of leading zeros (2/17/2009)
9	bD/bM/YY	Day-Month-Year with spaces instead of leading zeros (17/ 2/2009)
A	YY/bM/bD	Year-Month-Day with spaces instead of leading zeros (2009/ 2/17)
B	MMDDYY	Month-Day-Year with no separators (02172009)
C	DDMMYY	Day-Month-Year with no separators (17022009)
D	YYMMDD	Year-Month-Day with no separators (20090217)
E	MonDDYY	Month abbreviation-Day-Year with leading zeros (Feb172009)
F	DDMonYY	Day-Month abbreviation-Year with leading zeros (17Feb2009)
G	YYMonDD	Year-Month abbreviation-Day with leading zeros (2009Feb17)
H	day/YY	Day of year (counting consecutively from January 1)-Year (48/2009)
I	YY/day	Year-Day of Year (counting consecutively from January 1—often called the Julian date format) (2009/48)
J	D Month, Yr	Day-Month name-Year (17 February, 2009)
K	Yr, Month D	Year-Month name-Day (2009, February 17)
L *	Mon-DD-YYYY	Month abbreviation, Day with leading zeros, Year (Feb 17, 2009)
M *	DD-Mon-YYYY	Day with leading zeros, Month abbreviation, Year 17 Feb, 2009.

*** This format defaults to a two-digit year, but can be overridden to have four digits.**

Format	Date order	Description
N	YYYYY-Mon-DD	Year, Month abbreviation, Day with leading zeros (2009, Feb 17) This format defaults to a two-digit year, but can be overridden to have four digits.
O	Mon DD, YYYY	Month abbreviation, Day with leading zeros, Year (Feb 17, 2013)
P	DD Mon, YYYY	Day with leading zeros, Month abbreviation, Year (17 Feb, 2013)
Q	YYYY, Mon DD	Year, Month abbreviation, Day with leading zeros (2013, Feb 17)
X	(hexadecimal)	Eight-character hexadecimal representation of the system date. Valid dates range from 12/31/1969 to 01/18/2038. Valid dates may differ depending on the type of machine (PC or host) and the type of CPU chip.

*** This format defaults to a two-digit year, but can be overridden to have four digits.**

Month abbreviations consist of the first three characters of the month's name. Months with four-character names, such as June, are not abbreviated.

NOTE: The century cut-off date is used to determine the century for 2-digit years. This date defaults to 50, but you can change it using this INI option:

```
< Control >
   DateFMT2To4Year =
```

Anything less than or equal to the cut-off year is considered to fall in the current century. For instance using the default of 50, *13* would be interpreted as *2013*. Anything greater than the cut-off year is considered to fall in the previous century. For instance, again using the default of 50, *88* would be interpreted as *1988*. This is important when you have to determine the years or days between two dates.

There is a scenario where the system overrides a 2-digit year output. This only happens when the input has 4-digits and the output has 2-digits and the resulting 2-digit output does not yield the same results when read in again.

For instance, suppose your input is 01/01/1927 and the cutoff year is 50. Normally any 2-digit year with a value less than 50 is considered part of the current century. So if the system outputs the data as 01/01/27 and then tries to read this date back in, you would get 01/01/2025 and not 01/01/1927.

The system changes its normal behavior because it is designed to be able to read its own output and come up with the result originally provided in the original input. If, however, you specifically tell the system you only want two digits, you will get that output, but the system may not be able to read it back in and get the same results.

Locales

Here is a list of the currently supported localities:

For this country	And this language	Use this code
Argentina	Spanish	ARS
Australia	English	AUD
Austria	German	ATS
Belgium	Dutch	BED
Belgium	French	BEF
Bolivia	Spanish	BOB
Brazil	Portuguese	BRC
Canada	English	CAN
Canada	French	CAD
Chile	Spanish	CLP
Columbia	Spanish	COP
Denmark	Danish	DKK
Ecuador	Spanish	ECS
European Union	English	EUR
France	French	FRF
Finland	Finnish	FIM
Finland	Swedish	FMK
Germany	German	DEM
Guatemala	Spanish	GTQ
Iceland	Icelandic	ISK
Indonesia	Indonesian	IDR
Italy	Italian	ITL
Ireland	English	IEP
Liechtenstein	German	CHL
Luxembourg	French	FLX
Luxembourg	German	LUF
Mexico	Spanish	MXN

For this country	And this language	Use this code
The Netherlands	Dutch	NLG
New Zealand	English	NZD
Norway	Norwegian	NOK
Panama	Spanish	PAB
Paraguay	Spanish	PYG
Peru	Spanish	PES
Portugal	Portuguese	PTE
South Africa	English	ZAR
South Africa	Afrikaans	ZAA
Spain	Spanish	ESP
Sweden	Swedish	SEK
Switzerland	German	CHF
Switzerland	French	CHH
Switzerland	Italian	CHI
United Kingdom	English	GBP
United States	English	USD
Uruguay	Spanish	UYU
Venezuela	Spanish	VEB

Here are some examples, using *December 18, 2010*:

Example	Description	Result
1	Format type 1	12/18/10
1-	Format type 1 with dashes (-) as the separator characters	12-18-10
1/2	Format type 1 with backslashes (/) as the separator characters and a two-digit year	12/18/10
14	Format type 1 with a four-digit year (no separator specified but the format type includes separators so the default separator (/) will be used	12/18/10
B4	Format type B with a four-digit year (no separator specified and the format type does not include separators, so none will be included)	12182010

Example	Description	Result
4@CAD	Format type 4, with French Canadian as the locality. If you use “4@CAD” in a DAL function, the system returns the French Canadian translation of date format type 4 (Month D, YYYY with month spelled out). If you specify a locale, it must be the last component of the date format	décembre 18, 2010

Documaker Server Functions

The Documaker Server functions are summarized in the table below. Click on the function name to jump to a discussion of that function.

Function	Result
?	Returns data from an extract file.
AddOvFlwSym	Creates an overflow symbol.
AppIdxRec	Get an archive record based on the APPIDX.DFD file and Trigger2Archive INI settings.
CountRec	Counts the number of records in an extract file transaction that match a search mask parameter.
DDTSourceName	Returns the contents of the Source Name field in the DDT file you are currently processing. Applicable to batch processing only.
FieldRule	Executes a field-level rule from within a DAL script.
GetData	Retrieves data from a flat file extract file.
GetOvFlwSym	Retrieves the value stored in an overflow symbol.
GVM	Retrieves the contents of a GVM variable.
HaveGVM	Determines if a GVM variable exists.
IncOvFlwSym	Increments an overflow symbol.
KickToWIP	Sends a transaction to WIP from the GenData program.
ResetOvFlwSym	Resets the value in an overflow symbol to zero.
RPErrorMsg	Writes an error message into Documaker Server's error file.
RPLogMsg	Writes a message into Documaker Server's log file
RPWarningMsg	Writes a warning message into Documaker Server's error file.
SrchData	Retrieves data from an XML or flat extract file
SetGVM	Updates the contents of a GVM variable.
TriggerFormName	Returns the form name of the current SetRecipTb entry being processed.
TriggerImageName	Returns the section (FAP file) name of the current SetRecipTb entry being processed.
TriggerRecsPerOvFlw	Retrieves the number of records per overflow section value which is stored in the SETRCPTBL.DAT entry being processed.

Documaker Workstation Functions

The Documaker Workstation functions are summarized in the table below. Click on the function name to jump to a discussion of that function.

Function	Result
Ask	Creates a message box which requires a Yes or No answer from the user.
Beep	Creates a beep, which signals an event to the user.
Input	Creates a message which asks the user to enter information.
MLEInput	Creates a window with a title, prompt message, and a place for a user to enter multiple lines of text.
MLETranslate	Translates the *n* characters in a data string created by the MLEInput function.
MSG	Creates a message with an Ok button.
Refresh	Refreshes or repaints the screen.
SetEdit	Specifies which section field is the next field that should be used.
Table	Locate and return a value from a table.
TotalPages	Returns the number of pages that will print for a given recipient or for all recipients.
TotalSheets	Returns the total number of sheets of paper that will print for a recipient.

Docupresentment Functions

The Docupresentment functions are summarized in the table below. Click the function name to jump to a discussion of that function.

Function	Result
AddAttachVAR	Adds a string value as an attachment variable
GetAttachVAR	Returns the string value of an attachment variable
RemoveAttachVAR	Removes an attachment variable

Field Functions

Field functions retrieve or change data associated with variable fields defined on sections. The variable field functions are summarized in the table below. Click on the function name to jump to a discussion of that function.

Function	Result
@	Returns the value contained in a field.
AppendText	Append text into a multi-line field from an external text file.
AppendTxm	Append text into a multi-line field from an external multi-line text area.
AppendTxmUnique	Append text into a multi-line field from an external multi-line text area and rename the fields imported from the external text area so they have unique names.
CompressFlds	Compresses blank space by moving field data.
ConnectFlds	Repositions and aligns field text along a common horizontal coordinate so the field's data appears concatenated.
DelField	Deletes a field from a section.
FieldFormat	Returns the format string associated with the field format type.
FieldPrompt	Returns the text of the prompt for a field.
FieldType	Returns the field format type assigned to a field.
FieldX	Returns the X coordinate of a field object.
FieldY	Returns the Y coordinate of a field object.
JustField	Justifies a variable field content by modifying its field coordinates.
MAX	Returns the maximum value found in a set of fields that share a naming method.
MIN	Returns the minimum value found in a set of fields that share a naming method.
NUM	Return the numeric value from a field regardless of the field's format.
ResetFld	Clears a field of data.
SetFld	Assigns a value to a section field.
SetFont	Change the font on a field.
SetLink	Updates a hyperlink setting in a variable field, a graphic, or a text label.
SetProtect	Prevents a specified field from being altered.
SetRequiredFld	Changes the required option of a field to Required or Not Required.

Function	Result
Size	Returns the integer size of the data area of a section field.
SpanField	Moves a field horizontally and then resizes it to span the distance between two other specified fields.
STR	Return the contents of a field as a string without conversion.

Before you examine each field function individually, you should understand the available field formats and how to locate a specific field.

Field Formats

You can specify the field format for a specific section field. This restricts the type of data the field can accept. When you include field formats in DAL statements, place them in quotation marks. The following table lists the available field formats:

Format	Definition	Description
a	Alphabetic	Accepts only alphabetic characters (case sensitive)
A	Uppercase Alphabetic	Accepts only alphabetic characters and displays uppercase
B	Bar code	Accepts characters according to a bar code format string
C	Custom**	A custom formatted string
d	Date	Accepts date information according to a date format string
i	International Alphabetic	Accepts all alphabetic characters, including international characters, and is case sensitive
I	International Uppercase Alphabetic	Accepts all alphabetic characters, including international characters, and converts to uppercase
k	International Alphanumeric	Accepts all characters, including international characters, and is case sensitive
K	International Uppercase Alphanumeric	Accepts all characters, including international characters, and displays uppercase
m	X or space	Accepts an X or a space (used for a check box)
M	Multi-line text	No format
n	Numeric	Accepts numbers and uses a numeric format string
t	Table only	Accepts only information selected from a table
T	Time	Accepts only time

Format	Definition	Description
x	Alphanumeric	Accepts all non-international characters (case sensitive)
X	Uppercase Alphanumeric	Accepts all non-international characters and displays uppercase
y	Y or N	Accepts a Y or N (Yes or No)

** Custom formats are unique formats you create. You specify text to be inserted in an input string and where the text is to be inserted. For example, assume the input string is "123456789" and the custom format string is "3,-,2,-". This format takes the first three characters of the input string and inserts a hyphen(-), then takes the next two characters of the input string and inserts a hyphen (-), then appends the remainder of the input string. The result is: 123-45-6789.

Insertion text can be longer than a single character. Look at these examples:

Input Text	Format String	Output
B105	1,97	B97105
First Street	6, (not 1st)	First (not 1st) Street

Numeric Formats

The following table describes some common components that make up numeric formats.

Component	Description
","	Tells the system to automatically insert a comma in the specified position(s) of the field at data entry time.
"9"	Tells the system to place a number zero through nine (0-9) in that space. If there is no number to fill a digit preceding the number, the system uses zeros as placeholders.
"."	Tells the system to accept only a decimal point in the specified position at data entry time.

Component	Description
"Z"	Tells the system to automatically suppress leading zeros in the specified positions of the field at data entry time. Before version 10.0, system would suppress zeros and insert blanks. In version 10.0 and in subsequent versions, the system will not print a blank character. For example, if the field format was ($zzzzz9.99 and you entered $255.98, the system would display ($258.98). In version 10.0 and in subsequent versions, it shows ($258.98).
"$"	Tells the system to automatically insert a dollar sign in the specified position of the field at data entry time. The dollar sign may be used in a drifting manner or dollar fill. A single dollar sign in a field specifies that a currency system will always appear in the right most position before the first non-zero number. A dollar fill is specified by two dollar signs in the field format. A dollar fill specifies that leading zeros will be suppressed and replaced by the $symbol.
"*"	Works much the same way as a dollar fill, but suppresses zeros with asterisks instead of dollar signs. An asterisk (*) must follow a dollar sign to a valid field format.

The following lists provides examples of various numeric formats:

```
-ZZZZZZ9.99%
+ZZZZZZ9.99%
ZZZZZZ9.99-
ZZZZZZ9.99+
ZZZZZZ9.99DB
ZZZZZZ9.99CR
ZZZZZZ9.99
$ZZZZZZ9.99
99999999999
ZZZZZZZZZZZZZ
```

Locating Fields

The field functions can be used to get or change information on any field within a form set. By default, these field functions will assume that you are referencing a field located on the current section. To locate specific fields, elsewhere in the document, requires additional information. Any field's location can be precisely determined by the following hierarchy:

```
Field -> Section -> Form -> Group
```

Fields occur on sections. Sections occur on forms. Forms are defined within a form group (called a *Line of Business* in the insurance market). The form groups are specified by the user during form set selection.

Typically you will not have to specify all four components of the hierarchy to locate a given field for the DAL fields functions. By default, all field functions will search the current section which is the section that contains the script being executed. If the field you wish to reference occurs on the current section, then you do not have to specify any other information.

NOTE: You can also use the asterisk (*) as a wildcard, however, for optimal performance, avoid using wildcards (*) when searching for field, section, or form names.

To locate a field on a section other than the current one requires additional information. Each field function accepts optional parameters to identify a specific field, section, form, and/or group to search. In addition, each of these parameters will support an optional occurrence count to further identify the precise location of the field being requested.

A given field name is usually unique to a section. However, that same field name might also be used on any number of other sections. Further, there may be any number of occurrences of a section on a given form. Likewise, there may be additional copies of a form included in the form set. And finally, any two forms might share one or more sections in common.

Since it is possible to have any number of a similar named objects within a form set, the occurrence count, used with the object's name, is sometimes necessary to identify a specific object. The following table explains the method that DAL field functions will use to locate fields:

Field Name	Section Name	Form Name	Group Name	Description
omitted	*omitted*	*omitted*	*omitted*	In the absence of any of these parameters, the function will assume that you wish to use the current field.
"FLD"	*omitted*	*omitted*	*omitted*	Find FLD on the current section.
"FLD"	"IMG"	*omitted*	*omitted*	Find the first occurrence of IMG (a section) on the current form. If located, find FLD on that section.
"FLD"	*omitted*	"FRM"	*omitted*	Find the first occurrence of FRM (a form) in the current group. If located, find the first occurrence of FLD on that form. FLD may occur on any section on FRM since that parameter was omitted.
"FLD"	*omitted*	*omitted*	"GRP"	Find the first occurrence of FLD within the group, GRP. This field may be on any section on any form within that group.

Field Name	Section Name	Form Name	Group Name	Description
"FLD"	"IMG"	"FRM"	*omitted*	Find the first occurrence of FRM in the current group. Find the first occurrence of IMG on that form. Find FLD on that section.
"FLD"	"IMG"	*omitted"	"GRP"	Find the first occurrence of IMG within the group, GRP. This section may occur on any form since that parameter was not specified. Then find FLD on that section.
"FLD"	"IMG"	"FRM"	"GRP"	Find the first occurrence of FRM within the group, GRP. Then find the first occurrence of IMG. Finally, locate FLD on that section.

Notice that many of these descriptions referred to the first occurrence of a particular object. This is the default search method unless an occurrence count is specified on the object name. For instance, if there are three occurrences of the field "MYFIELD" on a particular form, you would distinguish them as "MYFIELD\1", "MYFIELD\2", and "MYFIELD\3". (In practice you do not have to specify "\1" to identify the first occurrence except on those field functions that match on partial names.)

The backslash is not a valid character in any object name. When found, the field functions will assume that the number following the backslash identifies the particular occurrence of that named object you are requesting.

Field, section, and form names may specify occurrence numbers. Group does not require an occurrence number because form groups are unique within the form set. The following table demonstrates several uses of occurrence indicators.

Field Name	Section Name	Form Name	Group Name	Description
"FLD"	"IMG\2"	*omitted*	*omitted*	Find the second occurrence of IMG (a section) on the current form. If located, find FLD on that section.
"FLD\3 "	*omitted*	"FRM\2"	*omitted*	Find the second occurrence of FRM (a form) in the current group. If located, find the third occurrence of FLD on that form. The third occurrence of FLD may occur on any section on FRM since that parameter was omitted.
"FLD\8 "	*omitted*	*omitted*	"GRP"	Find the eighth occurrence of FLD within the group, GRP. This field may occur on any section or form within that group.
"FLD"	"IMG\5"	*omitted*	"GRP"	Find the fifth occurrence of IMG (a section) within the group, GRP. If located, find FLD on that section.

Finally, it should be noted that if a named object, or occurrence of that object, cannot be located then the search will end in failure. For instance, if in the last example there are not 5 occurrences of IMG within the named group, then the function will cease looking for FLD and return without success.

File and Path Functions

The File and Path functions are summarized in the table below. Click on the function name to jump to a discussion of that function.

Function	Result
FileDrive	Gets the drive component of a file name.
FileExt	Gets the extension component of a file name.
FileName	Gets the name component of a file name.
FilePath	Gets the path component of a file name.
FullFileName	Makes a full file name from a string containing the file name components.
PathCreate	Creates the subdirectory path you specify if it does not exist.
PathExist	Checks the path you specify to make sure it exists.

Have Functions

The Have functions are summarized in the table below. Click on the function name to jump to a discussion of that function.

Function	Result
AddresseeCount	Determines the number of addressees for a named recipient parameter.
GetAddresseeValues	Gets the mapped values of addressee record members, as defined in the Extract Dictionary (XDD) Addressee record.
GetFormAttrib	Returns the content of the named user attribute (metadata) for the form you specify.
HaveField	Determines whether a named field exists.
HaveForm	Determines whether a named form exists.
HaveGroup	Determines whether a named group exists.
HaveImage	Determines whether a named section exists.
HaveLogo	Determines whether a named graphic (LOG) exists.
HaveRecip	Determines if a recipient name is defined in the FORM.DAT file.
PutFormAttrib	Saves the named attribute and information to a form within your document set
RecipCopyCount	Counts the number of recipient copies for specified sections and returns that number.

INI Functions

INI functions let your retrieve or set certain INI control group and option values. The INI functions you can use are listed below. Click on the function name to jump to a discussion of that function.

Function	Result
GetINIBool	Retrieves from memory the Boolean value of an INI control group and option string.
GetINIString	Retrieves from memory an INI control group and option string.
INI	Retrieves and INI control group and option string.
LoadINIFile	Loads an INI file into cache memory.
PutINIBool	Store a Boolean value in an INI control group and option Boolean variable.
PutINIString	Store a string value in an INI control group and option string variable.
SaveINIFile	Saves the values from an INI control group and option into a file.

NOTE: These functions retrieve values from any INI files loaded in memory. The system typically loads the FSIUSER.INI file first, which tells it to then load the FSISYS.INI file. If the same control group and option appear in more than one location in the files, these functions retrieve the value first defined.

See Using INI Options on page 8 also for a list of the DAL-related INI control groups and options.

Graphics Functions

The graphics functions are summarized in the table below. These functions affect LOG files. Click on the function name to jump to a discussion of that function.

Function	Result
ChangeLogo	Replaces an existing graphic on the section with a new graphic (LOG).
DelLogo	Deletes a graphic from a form.
InlineLogo	In-lines a graphic (LOG) into the print stream
Logo	Places a new graphic (LOG) at a specified position on the section.

Mathematical Functions

Mathematical functions perform certain mathematical operations and return the resulting value. The mathematical functions you can use are listed below. Click on the function name to jump to a discussion of that function.

Function	Result
ABS	Returns the absolute value of a number.
Avg	Averages a group of fields that share a naming method and returns the result.
Count	Counts the number of fields with values, shares a naming method, and returns the result.
INT	Returns the integer portion of a number.
MOD	Returns the remainder from modular arithmetic.
Numeric	Tests if a string contains a valid numeric value and returns one (1) if it does or zero (0) if it does not.
POW	Handles calculations such as those needed to figure annuities and interests rates.
SUM	Totals all fields that share a naming method and returns the result.

NOTE: DAL has a limit of 14 significant numbers. If you have a number with greater than 14 significant numbers and apply a DAL mathematical function to it, DAL will return a value of zero (0) for that number.

Miscellaneous Functions

Miscellaneous functions perform a variety of operations and return specific information or values. The miscellaneous functions are summarized in the table below. Click the function name to jump to a discussion of that function.

Function	Result
Always	Used as a placeholder or stub.
Call	Suspends one calculation and executes another calculation file.
Chain	Calls another calculation language file.
CFind	Temporarily suspends one calculation and executes another calculation file.
Exists	Determines if a DAL symbolic variable exists.
GetValue	Returns a string that contains the contents of the DAL symbolic variable specified by the parameter.
LoadLib	Loads a file that contains a library of DAL scripts.
MajorVersion	Retrieves the major version number of the system being executed.
MinorVersion	Retrieves the minor version number of the system being executed.
Print_It	Prints a string on the console.
Retain	Retains DAL variables during transaction processing.
UniqueString	Returns a 45-character globally unique string.

Name Functions

The Name functions are summarized in the table below. Click on the function name to jump to a discussion of that function.

Function	Result
FieldName	Returns the name of a field.
FormDesc	Retrieves a form description specified in a FORM.DAT file.
FormName	Returns a specified form's name.
GroupName	Returns a specified group's name.
ImageName	Returns a specified section's name.
PageImage	Returns the name of a section on a given page number within the form set or form.
RecipientName	Returns from the FORM.DAT file the recipient name related to the specified section, form, or group.
RenameLogo	Renames a graphic (LOG).
RootName	Extracts and returns the root name, or the original part of the name, of a specified string.
SetFormDesc	Change the description of a form.
WhatForm	Returns the name of the form that includes the item you searched for.
WhatGroup	Returns the name of the group that includes the item you searched for.
WhatImage	Returns the name of the section that includes the item you searched for.

Page Functions

The Page functions are summarized in the table below. Click the function name to jump to a discussion of that function.

Function	Result
AddBlankPages	Add blank or filler pages to the print stream
DelBlankPages	Removes blank or filler pages.
PageInfo	Gets information about the page of a form you specify.
PaginateForm	Applies section origins and re-paginates the form if necessary.

Printer and Recipient Functions

Print functions perform a variety of operations and return specific information or values. These functions are summarized in the table below. Click on the function name to jump to a discussion of that function.

Function	Result
AddComment	Adds a comment to the print stream.
AddDocusaveComment	Adds a comment to a Metacode or AFP print stream created specifically for Docusave.
BreakBatch	Tells Documaker Server to break the output print stream file for the current recipient batch after processing the current recipient, including post transaction banner processing.
DeviceName	Returns the current output device file name, such as the name of the current print stream output file.
IsPrintObject	Lets you know if the section (image), form, or group is printable, based on the current print recipient and the recipient copy count.
PrinterClass	Finds out the type of print stream the system is generating.
PrinterGroup	Retrieves the group name that is being used to generate the print stream.
PrinterID	Returns the printer ID assigned during a batch processing run.
PrinterOutputSize	Returns the approximate size of the current print output file during a batch print operation.
RecipBatch	Gets the name of the recipient batch file being processed. Used in banner or comment record processing.
RecipName	Gets the name of the recipient batch record for the transaction currently being printed. Used in banner or comment record processing.
SetDeviceName	Sets a new output device file name which will be used the next time the output device is opened, assuming nothing overrides the name prior to that.
SuppressBanner	Suppresses the printing of a banner page.

Section Functions

The section (image) functions are summarized in the table below. Click on the function name to jump to a discussion of that function.

Function	Result
AddImage	Adds a specified section to a form as a new page.
ApplyInserts	Force the insertion of items associated with applying logos, state stamps, and signatures to a form set
DelForm	Deletes a specified form from the current document.
DelImage	Deletes a section from a form.
EmbedLogo	Embeds a graphic (LOG) into the NAFILE.DAT file.
ImageRect	Retrieves the coordinates of a section.
SetImagePos	Repositions a section on a page.
SetRecip	Sets the recipient copy count for a form or group.

String Functions

String functions manipulate data to conform to a certain format. The string functions are summarized in the table below.

NOTE: If the destination of the data is a field with a specific format, keep in mind the system will execute any DAL processing *before* it applies the format specified in the field's format mask.

Function	Result
BankRound	Rounds numbers based on Banker's rounding. Values below 0.5 go down, values above 0.5 go up, and values of exactly 0.5 go to the nearest even number.
CFind	Finds and returns the position of a character (or string of characters) within another string of characters.
Char	Converts an integer into a single character.
CharV	Converts a single character into an integer value.
CodeInList	Searches for a string in a list of a strings.
Cut	Removes characters from a string at a specified position and returns the result.
DeFormat	Removes formatting from a string field and returns the result.
Find	Finds the position of a substring within a string and returns the result.
Format	Formats a string field and returns the result.
FrenchNumText	Converts a number into a string of words and returns the result (in French).
Insert	Inserts a substring into a string at a specified position and returns the result.
JCenter	Returns a string center justified.
JLeft	Returns a string left justified.
JRight	Returns a string right justified.
Left	Returns a specified number of left most characters.
LEN	Returns the current length of the string.
ListInList	Searches character string lists and returns the ordinal position (integer) of the first string in the second parameter that matches any of the strings in the first parameter.
Lower	Converts all characters to lowercase and returns the result.
NL	Retrieves a string that contains a new line character sequence.
NumText	Converts a number into a string of words and returns the result (in English).

Function	Result
PAD	Adds trailing spaces or characters and returns the result.
ParseListCount	Counts the indexed components within the formatted text
ParseListItem	Returns the indexed components from the formatted text.
Right	Returns a specified number of right most characters.
Round	Returns a number rounded to the nearest specified decimal point.
STRCompare	Compares two strings, considering case.
SUB	Returns a substring from a string at a specified position.
Trim	Removes end spaces and returns the result.
Upper	Converts all characters to uppercase and returns the result.

Time Functions

Time functions perform specific operations regarding time information. These functions enter or calculate a time. The time functions are summarized in the table below.

Function	Result
DiffHours	Calculates and returns the absolute time difference in hours between two times.
DiffMinutes	Calculates and returns the absolute time difference in minutes between two times.
DiffSeconds	Calculates and returns the absolute time difference in seconds between two times.
DiffTime	Calculates the difference in time between two times and returns a signed (positive or negative) value, given in seconds.
Hour	Extracts and returns the number of hours from a time.
Minute	Extracts and returns the number of minutes from a time.
Second	Extracts and returns the number of seconds from a time.
Time	Returns a time string or the current time in a specified format.
Time2Time	Converts a time from one format to another and returns the result.
TimeAdd	Adds time to a time and returns the new time.
TimeZone	Returns the system's time zone setting or makes sure a time zone is valid.
TimeZone2TimeZone	Converts date and time values from one geographic region into date and time values that are local to another geographic region.

Before examining each individual time function, take a look at the *time formats*. Time formats are usually one of the parameters you enter for a time function. The time format determines how your time information appears when it is returned to the section.

Time Formats

Times can be entered in several formats. The time formats are explained in this table:

Format	Time Segments	Description
1	HH:MM:SS	Time is based on a 24 hour system. This is frequently referred to as “military time”. The 24 hour system is the default format. Example: 14:18:23
2	HH:MM:SS XM	Time is based on a 12 hour system. AM or PM is given. Example: 02:18:23 PM
3	HH:MM	Time is based on a 24 hour system. Seconds are not given. Example: 14:18
4	HH:MM XM	Time is based on a 12 hour system. Seconds are not given. AM or PM is given. Example: 02:18 PM

The separators you can use include:

'.' = 99.99.99	',' = 99,99,99	'-' = 99-99-99
'b' = 99 99 99	':' = 99:99:99 (default)	

Using the Time Zone Functions

The TimeZone and TimeZone2TimeZone functions are not available on mainframe platforms like z/OS. They are only available on Windows and UNIX platforms.

These functions use the International Components for Unicode (ICU) library. The ICU system time zones are derived from the tz database (also known as the Olson database) available at...

ftp://elsie.nci.nih.gov/pub

This is the data used across much of the industry, including by UNIX systems.

The ICU time zone functionality supports

- Standard time zones, such as Eastern Standard Time (EST), Central Standard Time (CST), and so on.
- Time zone IDs defined in the standard Olson data used by UNIX systems. These time zone IDs use the following format:

  ```
  continent/city or ocean/city
  ```

  For example, *America/Los_Angeles* is an ID for Pacific Standard Time.
- Custom time zones based on Greenwich Mean Time (GMT), in this format:

  ```
  “GMT[+|-]hh[[:]mm]")
  ```

ICU Time Zones

Here is a list of the various International Components for Unicode (ICU) time zones:

Time Zones			
ACT	AET	Africa/Abidjan	Africa/Accra
Africa/Addis_Ababa	Africa/Algiers	Africa/Asmera	Africa/Bamako
Africa/Bangui	Africa/Banjul	Africa/Bissau	Africa/Blantyre
Africa/Brazzaville	Africa/Bujumbura	Africa/Cairo	Africa/Casablanca
Africa/Ceuta	Africa/Conakry	Africa/Dakar	Africa/Dar_es_Salaam
Africa/Djibouti	Africa/Douala	Africa/El_Aaiun	Africa/Freetown
Africa/Gaborone	Africa/Harare	Africa/Johannesburg	Africa/Kampala
Africa/Khartoum	Africa/Kigali	Africa/Kinshasa	Africa/Lagos
Africa/Libreville	Africa/Lome	Africa/Luanda	Africa/Lubumbashi
Africa/Lusaka	Africa/Malabo	Africa/Maputo	Africa/Maseru
Africa/Mbabane	Africa/Mogadishu	Africa/Monrovia	Africa/Nairobi
Africa/Ndjamena	Africa/Niamey	Africa/Nouakchott	Africa/Ouagadougou
Africa/Porto-Novo	Africa/Sao_Tome	Africa/Timbuktu	Africa/Tripoli
Africa/Tunis	Africa/Windhoek	AGT	America/Adak
America/Anchorage	America/Anguilla	America/Antigua	America/Araguaina
America/Argentina/Buenos_Aires	America/Argentina/Catamarca	America/Argentina/ComodRivadavia	America/Argentina/Cordoba
America/Argentina/Jujuy	America/Argentina/La_Rioja	America/Argentina/Mendoza	America/Argentina/Rio_Gallegos
America/Argentina/San_Juan	America/Argentina/Tucuman	America/Argentina/Ushuaia	America/Aruba
America/Asuncion	America/Atikokan	America/Atka	America/Bahia
America/Barbados	America/Belem	America/Belize	America/Blanc-Sablon
America/Boa_Vista	America/Bogota	America/Boise	America/Buenos_Aires
America/Cambridge_Bay	America/Campo_Grande	America/Cancun	America/Caracas
America/Catamarca	America/Cayenne	America/Cayman	America/Chicago
America/Chihuahua	America/Coral_Harbour	America/Cordoba	America/Costa_Rica

Time Zones			
America/Cuiaba	America/Curacao	America/Danmarkshavn	America/Dawson
America/Dawson_Creek	America/Denver	America/Detroit	America/Dominica
America/Edmonton	America/Eirunepe	America/El_Salvador	America/Ensenada
America/Fort_Wayne	America/Fortaleza	America/Glace_Bay	America/Godthab
America/Goose_Bay	America/Grand_Turk	America/Grenada	America/Guadeloupe
America/Guatemala	America/Guayaquil	America/Guyana	America/Halifax
America/Havana	America/Hermosillo	America/Indiana/ Indianapolis	America/Indiana/Knox
America/Indiana/Marengo	America/Indiana/Petersburg	America/Indiana/Vevay	America/Indiana/Vincennes
America/Indianapolis	America/Inuvik	America/Iqaluit	America/Jamaica
America/Jujuy	America/Juneau	America/Kentucky/Louisville	America/Kentucky/ Monticello
America/Knox_IN	America/La_Paz	America/Lima	America/Los_Angeles
America/Louisville	America/Maceio	America/Managua	America/Manaus
America/Martinique	America/Mazatlan	America/Mendoza	America/Menominee
America/Merida	America/Mexico_City	America/Miquelon	America/Moncton
America/Monterrey	America/Montevideo	America/Montreal	America/Montserrat
America/Nassau	America/New_York	America/Nipigon	America/Nome
America/Noronha	America/North_Dakota/ Center	America/North_Dakota/ New_Salem	America/Panama
America/Pangnirtung	America/Paramaribo	America/Phoenix	America/Port-au-Prince
America/Port_of_Spain	America/Porto_Acre	America/Porto_Velho	America/Puerto_Rico
America/Rainy_River	America/Rankin_Inlet	America/Recife	America/Regina
America/Rio_Branco	America/Rosario	America/Santiago	America/Santo_Domingo
America/Sao_Paulo	America/Scoresbysund	America/Shiprock	America/St_Johns
America/St_Kitts	America/St_Lucia	America/St_Thomas	America/St_Vincent
America/Swift_Current	America/Tegucigalpa	America/Thule	America/Thunder_Bay
America/Tijuana	America/Toronto	America/Tortola	America/Vancouver
America/Virgin	America/Whitehorse	America/Winnipeg	America/Yakutat

Time Zones			
America/Yellowknife	Antarctica/Casey	Antarctica/Davis	Antarctica/DumontDUrville
Antarctica/Mawson	Antarctica/McMurdo	Antarctica/Palmer	Antarctica/Rothera
Antarctica/South_Pole	Antarctica/Syowa	Antarctica/Vostok	Arctic/Longyearbyen
ART	Asia/Aden	Asia/Almaty	Asia/Amman
Asia/Anadyr	Asia/Aqtau	Asia/Aqtobe	Asia/Ashgabat
Asia/Ashkhabad	Asia/Baghdad	Asia/Bahrain	Asia/Baku
Asia/Bangkok	Asia/Beirut	Asia/Bishkek	Asia/Brunei
Asia/Calcutta	Asia/Choibalsan	Asia/Chongqing	Asia/Chungking
Asia/Colombo	Asia/Dacca	Asia/Damascus	Asia/Dhaka
Asia/Dili	Asia/Dubai	Asia/Dushanbe	Asia/Gaza
Asia/Harbin	Asia/Hong_Kong	Asia/Hovd	Asia/Irkutsk
Asia/Istanbul	Asia/Jakarta	Asia/Jayapura	Asia/Jerusalem
Asia/Kabul	Asia/Kamchatka	Asia/Karachi	Asia/Kashgar
Asia/Katmandu	Asia/Krasnoyarsk	Asia/Kuala_Lumpur	Asia/Kuching
Asia/Kuwait	Asia/Macao	Asia/Macau	Asia/Magadan
Asia/Makassar	Asia/Manila	Asia/Muscat	Asia/Nicosia
Asia/Novosibirsk	Asia/Omsk	Asia/Oral	Asia/Phnom_Penh
Asia/Pontianak	Asia/Pyongyang	Asia/Qatar	Asia/Qyzylorda
Asia/Rangoon	Asia/Riyadh	Asia/Riyadh87	Asia/Riyadh88
Asia/Riyadh89	Asia/Saigon	Asia/Sakhalin	Asia/Samarkand
Asia/Seoul	Asia/Shanghai	Asia/Singapore	Asia/Taipei
Asia/Tashkent	Asia/Tbilisi	Asia/Tehran	Asia/Tel_Aviv
Asia/Thimbu	Asia/Thimphu	Asia/Tokyo	Asia/Ujung_Pandang
Asia/Ulaanbaatar	Asia/Ulan_Bator	Asia/Urumqi	Asia/Vientiane
Asia/Vladivostok	Asia/Yakutsk	Asia/Yekaterinburg	Asia/Yerevan
AST	Atlantic/Azores	Atlantic/Bermuda	Atlantic/Canary
Atlantic/Cape_Verde	Atlantic/Faeroe	Atlantic/Jan_Mayen	Atlantic/Madeira
Atlantic/Reykjavik	Atlantic/South_Georgia	Atlantic/St_Helena	Atlantic/Stanley

Time Zones			
Australia/ACT	Australia/Adelaide	Australia/Brisbane	Australia/Broken_Hill
Australia/Canberra	Australia/Currie	Australia/Darwin	Australia/Hobart
Australia/LHI	Australia/Lindeman	Australia/Lord_Howe	Australia/Melbourne
Australia/North	Australia/NSW	Australia/Perth	Australia/Queensland
Australia/South	Australia/Sydney	Australia/Tasmania	Australia/Victoria
Australia/West	Australia/Yancowinna	BET	Brazil/Acre
Brazil/DeNoronha	Brazil/East	Brazil/West	BST
Canada/Atlantic	Canada/Central	Canada/East-Saskatchewan	Canada/Eastern
Canada/Mountain	Canada/Newfoundland	Canada/Pacific	Canada/Saskatchewan
Canada/Yukon	CAT	CET	Chile/Continental
Chile/EasterIsland	CNT	CST	CST6CDT
CTT	Cuba	EAT	ECT
EET	Egypt	Eire	EST
EST5EDT	Etc/GMT	Etc/GMT+0	Etc/GMT+1
Etc/GMT+10	Etc/GMT+11	Etc/GMT+12	Etc/GMT+2
Etc/GMT+3	Etc/GMT+4	Etc/GMT+5	Etc/GMT+6
Etc/GMT+7	Etc/GMT+8	Etc/GMT+9	Etc/GMT-0
Etc/GMT-1	Etc/GMT-10	Etc/GMT-11	Etc/GMT-12
Etc/GMT-13	Etc/GMT-14	Etc/GMT-2	Etc/GMT-3
Etc/GMT-4	Etc/GMT-5	Etc/GMT-6	Etc/GMT-7
Etc/GMT-8	Etc/GMT-9	Etc/GMT0	Etc/Greenwich
Etc/UCT	Etc/Universal	Etc/UTC	Etc/Zulu
Europe/Amsterdam	Europe/Andorra	Europe/Athens	Europe/Belfast
Europe/Belgrade	Europe/Berlin	Europe/Bratislava	Europe/Brussels
Europe/Bucharest	Europe/Budapest	Europe/Chisinau	Europe/Copenhagen
Europe/Dublin	Europe/Gibraltar	Europe/Guernsey	Europe/Helsinki
Europe/Isle_of_Man	Europe/Istanbul	Europe/Jersey	Europe/Kaliningrad
Europe/Kiev	Europe/Lisbon	Europe/Ljubljana	Europe/London

Time Zones			
Europe/Luxembourg	Europe/Madrid	Europe/Malta	Europe/Mariehamn
Europe/Minsk	Europe/Monaco	Europe/Moscow	Europe/Nicosia
Europe/Oslo	Europe/Paris	Europe/Prague	Europe/Riga
Europe/Rome	Europe/Samara	Europe/San_Marino	Europe/Sarajevo
Europe/Simferopol	Europe/Skopje	Europe/Sofia	Europe/Stockholm
Europe/Tallinn	Europe/Tirane	Europe/Tiraspol	Europe/Uzhgorod
Europe/Vaduz	Europe/Vatican	Europe/Vienna	Europe/Vilnius
Europe/Volgograd	Europe/Warsaw	Europe/Zagreb	Europe/Zaporozhye
Europe/Zurich	Factory	GB	GB-Eire
GMT	GMT+0	GMT-0	GMT0
Greenwich	Hongkong	HST	Iceland
IET	Indian/Antananarivo	Indian/Chagos	Indian/Christmas
Indian/Cocos	Indian/Comoro	Indian/Kerguelen	Indian/Mahe
Indian/Maldives	Indian/Mauritius	Indian/Mayotte	Indian/Reunion
Iran	Israel	IST	Jamaica
Japan	JST	Kwajalein	Libya
MET	Mexico/BajaNorte	Mexico/BajaSur	Mexico/General
Mideast/Riyadh87	Mideast/Riyadh88	Mideast/Riyadh89	MIT
MST	MST7MDT	Navajo	NET
NST	NZ	NZ-CHAT	Pacific/Apia
Pacific/Auckland	Pacific/Chatham	Pacific/Easter	Pacific/Efate
Pacific/Enderbury	Pacific/Fakaofo	Pacific/Fiji	Pacific/Funafuti
Pacific/Galapagos	Pacific/Gambier	Pacific/Guadalcanal	Pacific/Guam
Pacific/Honolulu	Pacific/Johnston	Pacific/Kiritimati	Pacific/Kosrae
Pacific/Kwajalein	Pacific/Majuro	Pacific/Marquesas	Pacific/Midway
Pacific/Nauru	Pacific/Niue	Pacific/Norfolk	Pacific/Noumea
Pacific/Pago_Pago	Pacific/Palau	Pacific/Pitcairn	Pacific/Ponape
Pacific/Port_Moresby	Pacific/Rarotonga	Pacific/Saipan	Pacific/Samoa

Time Zones			
Pacific/Tahiti	Pacific/Tarawa	Pacific/Tongatapu	Pacific/Truk
Pacific/Wake	Pacific/Wallis	Pacific/Yap	PLT
PNT	Poland	Portugal	PRC
PRT	PST	PST8PDT	ROC
ROK	Singapore	SST	Turkey
UCT	Universal	US/Alaska	US/Aleutian
US/Arizona	US/Central	US/East-Indiana	US/Eastern
US/Hawaii	US/Indiana-Starke	US/Michigan	US/Mountain
US/Pacific	US/Pacific-New	US/Samoa	UTC
VST	W-SU	WET	Zulu

When converting times

When converting times from one locale to another, keep in mind that the two locales must represent time in a similar manner. If, for instance, you have a DAL script that requests a US (the default) time value which includes AM or PM indicators, you must make sure the target Time field can interpret a US time. Otherwise, the AM and PM are interpreted as invalid characters for the specified locale.

In these situations, you either have to change the Time format parameter to include the locale your target field wants or you have to specify a format that does not include the AM/PM indicator and then allow the field editing to fill that in for you.

For instance, if the source locale was US and the target locale was ZAA (South Africa/ Afrikaans), you either need...

```
Return ( TIME( “2@ZAA”, 13, 30, 5 ) )
```

or

```
Return ( TIME( 1, 13, 30, 5 ) )
```

This is also applicable when you are handling dates. For example, suppose you chose to make a Date field use Afrikaans with a format of *Month DD, YYYY* – where the month name is expected. If you try to type (or return from a script) *October* as the month name, you would get an error because in Afrikaans that month is spelled *Oktober*.

For any field that has a locale-specific format, be sure to enter any characters or symbols required by the target language.

WIP Functions

Work-in-process (WIP) functions perform a variety of WIP-related functions and return specific information or values, such as a value from the current WIP record. The WIP functions are summarized in the table below. Click on the function name to jump to a discussion of that function.

Function	Result
AddForm	Adds a specified form to the current document.
AddForm_Propagate	Add a new form to a document and propagates global data onto that form.
AddImage_Propagate	Add a new section to a document and propagates global data onto that section.
AFELog	Writes a message to the AFELOG file.
AssignWIP	Assigns work-in-process and associated data to a different user ID.
Complete	Completes the work-in-process.
CopyForm	Copies a form and its field contents (data) into a new form.
DelWIP	Deletes the work-in-process and its associated data.
DupForm	Duplicates a form.
MailWIP	Sends the current document to a specified email address.
Print	Prints the current form set.
RouteWIP	Routes work in process to names specified via routing slip.
SaveWIP	Saves the WIP record being processing.
SetWIPFld	Sets WIP fields from DAL to the record in memory.
SlipAppend	Appends a new email address to a slip in route.
SlipInsert	Inserts a new email address into a slip in route.
TriggerForm	Adds a form to a form set and evaluates the form's section triggers.
UserID	Returns the user ID used to log into the system.
UserLvl	Returns the current user's rights level.
WIPExit	Exits entry immediately and saves or discards work in WIP.
WIPFld	Returns the value of the identified WIP field.
WIPKey1	Returns the value of the Key1 field from the current WIP record.
WIPKey2	Returns the value of the Key2 field from the current WIP record.
WIPKeyID	Returns the value of the KeyID field from the current WIP record.

XML Functions

Use DAL XML API functions to let Documaker applications access specified XML documents and retrieve XML data via a DAL script. These functions are registered in keywords, called built-in functions. An XML built-in function performs an operation on a set of parameters and returns a DAL variable in one of the three types: list, integer, or string.

The XML functions are summarized in the table below. Click the function name to jump to a discussion of that function.

Function	Result
DestroyList	Destroys the XML tree created by the LoadXMLList function.
GetListElem	Returns a text string which contains the first element that matches the search criteria
IsXMLError	Checks the list for the error status.
LoadXMLList	Loads an XML document and extracts an XML tree.
XMLAttrName	Returns the name of the current attribute pointed to by the XMLFirstAttrib and XMLNextAttrib functions.
XMLAttrValue	Returns the value of the attribute pointed to by the XMLFirstAttrib and XMLNextAttrib functions.
XMLFind	Locates the XML path from the extracted XML tree and returns a list of matched elements to a list type DAL variable or a matched text to a string type DAL variable.
XMLFirst	Sets the current pointer to the first element in the specified list.
XMLFirstAttrib	Sets the attribute pointer to the first attribute for the current element in the element list or to the first attribute element in the attribute list.
XMLFirstText	Sets the current text to be the first text element in the XML search list and then retrieve that text.
XMLGetCurName	Returns the element name from the current element.
XMLGetCurText	Returns the text from the current element.
XMLNext	Sets the current pointer to the next node or element in the specified list.
XMLNextAttrib	Sets the current attribute pointer to the next attribute for the current element in the list or to the next attribute element in the attribute list.
XMLNextText	Retrieves the next text element in the XML search list.
XMLNthAttrName	Returns the nth attribute name indicated by the index number.
XMLNthAttrValue	Returns the nth attribute value indicated by the index number.
XMLNthText	Returns the nth text value, as indicated by the index number.

USING DAL XML FUNCTIONS

There are two scenarios in which you would use DAL XML functions:

Scenario 1

A Documaker program, such as GenData, loads an XML document and extracts the XML tree at the transaction level using the XMLFileExtract rule. This rule creates a list type DAL variable with a default name of *%extract* and pushes it onto the DAL stack.

Then you can call other XML functions in a DAL script to access the XML tree and extract XML data.

Here are examples of the form set and section rules you would add and a DAL script that would call the XML functions.

- Add this in the AFGJOB.JDT file:

  ```
  ;XMLFileExtract;2;File=.\deflib\test.xml
  ```

  The rule loads the XML file and creates a list type DAL variable to pass the XML tree to the XML API function.

- Add this in your DDT file:

  ```
  ;0;0;DALXMLSCRIPT;0;9;DALXMLSCRIPT;0;9;;DAL;Call("TEST.DAL");N;N;N;
  N;4792;19444;11010;
  ```

  TEST.DAL is the name of the DAL script file.

- Here is an example of the DAL script:

  ```
  %listH=XMLFind(%extract, "Forms", "Form");
  #rc=XMLFirst(%listH);
  if #rc=0
  return("Failed to XMLFirst");
  end
  aStr=XMLGetCurText(%listH);
  return(aStr);
  ```

  %listH denotes a list type DAL variable. *#rc* denotes an integer type DAL variable. *aStr* denotes a string type DAL variable.

Scenario 2

You can also load the XML document and create the XML tree at a specific section field by calling the LoadXMLList rule from a DAL script. You must set the calling procedure in the DDT file as shown in Scenario 1.

Here is an example of DAL script file:

```
%xListH=LoadXMLList("test.xml");
%listH=XMLFind(%xListH,"Forms","Form/@*");
aStr=XMLNthAttrValue(%listH,2);
#rc=DestroyList(%xListH);
return(aStr);
```

XML PATH LOCATOR

The XMLFind function is called the DAL XML path locator or *DAL XPath*. It is a limited version of the XML path and does not cover all aspects defined in the W3C literature.

Refer to W3C recommendations for the description of XPointer and XPath syntax. You can use the XPATHW32 testing tool to verify the applicable specifications of Oracle Insurance's DAL XPath. Run the XPATHW32 program to get the syntax.

Below is a summary of XML path specifications for DAL XPath:

Axes

These axes apply:

ancestor	ancestor-or-self	attribute
child	descendant	descendant-or-self
following	following-sibling	parent
preceding	preceding-sibling	self

Function calls

You can use these function calls:

last()	position()	node()
text()	name(***node-set***)	string(***object***)
concat(***string***, ***string***, ***string***...)		

Operators or signs

You can use these operators or signs:

=	!=	<	>	+	-	/	//	*	::	[	]

Expressions

You can use abbreviated syntax, as this table shows:

For...	Use this abbreviation:
child::*	*
child::para	para
child::chapter/child::para	chapter/para
child::para[position()=1]	para[1]
/child::chapter/child::para[position()=last()]	/chapter/para[last()]
child::text()	text()
child::node()	node()
child::para[attribute::type]	para[@type]
child::para[attribute::type="warning"]	para[@type="warning"]

For...	Use this abbreviation:
child::para[attribute::type="warning"][position()=2]	para[@type="warning"][2]
child::chapter[child::title]	chapter[title]
child::chapter[child::title="Introduction"]	chapter[title="Introduction"]
child::doc/descendant-or-self::node()/child::para	doc//para
attribute::*	@*
attribute::type	@type
/descendant-or-self::node()/child::para	//para
self::node()	.
self::node/descendant-or-self::node()/child::para	.//para
parent::node()	..
parent::node()/child::chapter	../chapter
parent::node()/attribute::type	../@type

The XMLFind function locates the XML path from the extract XML tree and returns a valid DAL variable result. It requires three input parameters, a list type DAL variable and two string type variables. They in turn pass in an XML tree, a node name from which the search starts, and XML path location for searching.

If you omit the second parameter, the search starts from the root. The return DAL variable *Result* can be either list type or string type, depending on XML path.

Here are some examples that result in different return values:

Element list

```
%elemListH=XMLFind(%extract, , "descendant::Form[@ID=Agent]");
```

In this example, DAL Xpath selects the *Form* element descendants that have an attribute with name *ID* and value *Agent* from the extract XML tree (root), and returns an element list.

Attribute list

```
%attrListH=XMLFind(%extract, "Forms", "Form/@type='warning'");
```

In this example, DAL Xpath returns an attribute list that collects type attributes with value *warning* for *Form* children of current context node Forms.

Text list

```
%TextListH= XMLFind(%extract, "Forms", "Form/text()");
```

In this example, DAL Xpath returns a text list that contains all text nodes of *Form* children of current context node *Forms*.

Text string

```
aStr=XMLFind(%extract, Forms, "string(Form[2])");
```

It returns the text of second child *Form* of the current context node *Forms*.

```
aStr=XMLFind(%extract, "Forms", "concat("Get form 2 text: ",
"Form[2])");
```

It returns the concatenation of the text string *Get form 2 text:* , and the text of the second child *Form* of current context node *Forms*.

```
aStr=XMLFind(%extract, "Forms", "name()");
```

It returns the name of current context node.

Locating Objects

Many of the graphics, section (image), page, have, WIP, and name functions support parameters that let you locate an object anywhere within the form set. The object hierarchy supported is explained below. This explanation also agrees with the field parameters discussed in Locating Fields on page 64.

```
item -> Section -> Form -> Group
```

A number of different object types are supported by sections. Three objects that can be located on a section are fields, graphics, and recipients. For information about fields, see Locating Fields on page 64. Fields, graphics, and recipients are all objects that belong or are defined on a section.

Sections occur on forms. Forms are defined within a form group (commonly referred to as a *Line Of Business*). The form groups are specified by the user during form set selection.

To locate a specific object within the document often requires one or more parameter names. For instance, to locate a specific field, in addition to the field's name, might require the section name, form name, and/or group name. Similarly, a function used to locate a specific form, in addition to the form's name, lets you specify the group to which it belongs.

In addition to an object's name, most parameters will support an optional occurrence count to further identify the precise location of the object being requested.

Typically children of a section are unique to that section. However, that same object name might also be used on any number of other sections. Further, there may be any number of occurrences of a section on a given form. Likewise, there may be additional copies of a form included in the form set. And finally, any two forms might share one or more sections in common.

Since you can have any number of a similar named objects within a form set, the occurrence count, used with the object's name, is sometimes necessary to identify a specific object. The following table explains many of the variations that are valid when locating form set objects.

Item Name	Section Name	Form Name	Group Name	Description
"ITEM"	*omitted*	*omitted*	*omitted*	Find the object on the current section.
"ITEM"	"IMG"	*omitted*	*omitted*	Find the first occurrence of IMG (a section) on the current form. If located, find ITEM on that section.
"ITEM"	*omitted*	"FRM"	*omitted*	Find the first occurrence of FRM (a form) in the current group. If located, find the first occurrence of ITEM on that form. The item may occur on any section on FRM since that parameter was omitted.
"ITEM"	*omitted*	*omitted*	"GRP"	Find the first occurrence of ITEM within the group, GRP. This item may be on any section on any form within that group.

Item Name	Section Name	Form Name	Group Name	Description
"ITEM"	"IMG"	"FRM"	*omitted*	Find the first occurrence of FRM in the current group. Find the first occurrence of IMG on that form. Find ITEM on that section.
"ITEM"	"IMG"	*omitted"	"GRP"	Find the first occurrence of IMG within the group, GRP. This section may occur on any form since that parameter was not specified. Then find ITEM on that section.
"ITEM"	"IMG"	"FRM"	"GRP"	Find the first occurrence of FRM within the group, GRP. Then find the first occurrence of IMG. Finally, find ITEM on that section.
Locating Sections				
	"IMG"	*omitted*	*omitted*	Find the occurrence of the section on the current form.
	"IMG"	"FRM"	*omitted*	Find the occurrence of the section on the form named. The form is assumed to be in the current group.
	"IMG"	*omitted*	"GRP"	Locate the section within the named group.
	"IMG"	"FRM"	"GRP"	Locate the form in the specified group. Then locate the section on that form.
Locating Forms				
		"FRM"	*omitted*	Locate the form within the current group.
		"FRM"	"GRP"	Locate the form in the specified group.
Locating Groups				
			"GRP"	Locate the specified group.

In the previous table, ITEM refers to the name of an object type expected by the function. In other words, if the function is used to reference fields, you cannot locate the object if you give it the name of any other object type.

Many of these descriptions referred to the *first occurrence* of a particular object. This is the default search method unless an occurrence count is specified on the object name. For instance, if there are three occurrences of a given object on a particular form, you would distinguish them as ITEM\1, ITEM\2, and ITEM\3. (In practice, you do not have to specify *\1* to identify the first occurrence except with those functions that match on partial names.)

A backslash (\) is not a valid character in any object name. When found, the object functions assume that the number following the backslash identifies the particular occurrence of that named object you are requesting. Group names do not require an occurrence number because form groups are unique within the form set. The following table demonstrates several uses of occurrence indicators.

Item Name	Section Name	Form Name	Group Name	Description
"ITEM"	"IMG\2"	*omitted*	*omitted*	Find the second occurrence of IMG (a section) on the current form. If located, find ITEM on that section.
"ITEM\3"	*omitted*	"FRM\2"	*omitted*	Find the second occurrence of FRM (a form) in the current group. If located, find the third occurrence of ITEM on that form.
"ITEM"	"IMG\5"	*omitted*	"GRP"	Find the fifth occurrence of IMG (a section) within the group, GRP. If located, find ITEM on that section.

Finally, if a named object, or occurrence of that object, cannot be located the search ends in failure. For instance, if in the last example there are not five occurrences of IMG within the named group, then the function stops looking for the item and returns without success.

Where DAL Functions are Used

You use DAL functions to enhance the collection of data during either the form entry process (Documaker Workstation) or in the forms processing cycle (Documaker Server). All DAL functions can be used during the form entry process and most can be used during the form processing cycle. The following table shows you where the various functions affect processing.

Function/Procedure	Affects form entry (Documaker Workstation)	Affects form processing (Documaker Server)
@	Yes	Yes
?	No	Yes
ABS	Yes	Yes
AddAttachVAR	No	Yes
AddBlankPages	Yes	Yes
AddComment	No	Yes
AddDocusaveComment	No	Yes
AddForm	Yes	No
AddForm_Propagate	Yes	Yes
AddImage	Yes	No
AddImage_Propagate	Yes	Yes
AddOvFlwSym	No	Yes
AFELog	Yes	No
AppendText	Yes	Yes
AppendTxm	Yes	Yes
AppendTxmUnique	Yes	Yes
AppIdxRec	Yes	Yes
ApplyInserts	Yes	Yes
Ask	Yes	No
AssignWIP	Yes	No
Avg	Yes	Yes
BankRound	Yes	Yes

* While these XML-related functions affect both Documaker Workstation and Documaker Server, Documaker Server can have a default variable that refers to the transaction loaded in the XML extract. No such variable would exist automatically within Documaker Workstation.

Function/Procedure	Affects form entry (Documaker Workstation)	Affects form processing (Documaker Server)
Beep	Yes	No
BitAnd	Yes	Yes
BitClear	Yes	Yes
BitNot	Yes	Yes
BitOr	Yes	Yes
BitRotate	Yes	Yes
BitSet	Yes	Yes
BitShift	Yes	Yes
BitTest	Yes	Yes
BitXor	Yes	Yes
BreakBatch	No	Yes
CFind	Yes	Yes
ChangeLogo	Yes	Yes
CodeInList	Yes	Yes
Complete	Yes	No
CompressFlds	Yes	Yes
ConnectFlds	Yes	Yes
CopyForm	Yes	Yes
Count	Yes	Yes
CountRec	No	Yes
Cut	Yes	Yes
DashCode	Yes	Yes
Date	Yes	Yes
Date2Date	Yes	Yes
DateAdd	Yes	Yes

* While these XML-related functions affect both Documaker Workstation and Documaker Server, Documaker Server can have a default variable that refers to the transaction loaded in the XML extract. No such variable would exist automatically within Documaker Workstation.

Function/Procedure	Affects form entry (Documaker Workstation)	Affects form processing (Documaker Server)
DateCnv	Yes	Yes
Day	Yes	Yes
DayName	Yes	Yes
DaysInMonth	Yes	Yes
DaysInYear	Yes	Yes
DBAdd	Yes	Yes
DBClose	Yes	Yes
DBDelete	Yes	Yes
DBFind	Yes	Yes
DBFirstRec	Yes	Yes
DBNextRec	Yes	Yes
DBOpen	Yes	Yes
DBPrepVars	Yes	Yes
DBUnloadDFD	Yes	Yes
DBUpdate	Yes	Yes
DDTSourceName	No	Yes
Dec2Hex	Yes	Yes
DeFormat	Yes	Yes
DelBlankPages	Yes	Yes
DelField	Yes	Yes
DelForm	Yes	No
DelImage	Yes	No
DelLogo	Yes	Yes
DelWIP	Yes	No
DestroyList *	Yes	Yes

* While these XML-related functions affect both Documaker Workstation and Documaker Server, Documaker Server can have a default variable that refers to the transaction loaded in the XML extract. No such variable would exist automatically within Documaker Workstation.

Function/Procedure	Affects form entry (Documaker Workstation)	Affects form processing (Documaker Server)
DeviceName	Yes	Yes
DiffDate	Yes	Yes
DiffDays	Yes	Yes
DiffHours	Yes	Yes
DiffMinutes	Yes	Yes
DiffMonths	Yes	Yes
DiffSeconds	Yes	Yes
DiffTime	Yes	Yes
DiffYears	Yes	Yes
DupForm	Yes	No
EmbedLogo	Yes	Yes
Exists	No	Yes
FieldFormat	Yes	No
FieldName	Yes	Yes
FieldPrompt	Yes	Yes
FieldRule	No	Yes
FieldType	Yes	Yes
FieldX	Yes	Yes
FieldY	Yes	Yes
FileDrive	Yes	Yes
FileExt	Yes	Yes
FileName	Yes	Yes
FilePath	Yes	Yes
Find	Yes	Yes
Format	Yes	Yes

* While these XML-related functions affect both Documaker Workstation and Documaker Server, Documaker Server can have a default variable that refers to the transaction loaded in the XML extract. No such variable would exist automatically within Documaker Workstation.

Function/Procedure	Affects form entry (Documaker Workstation)	Affects form processing (Documaker Server)
FormDesc	Yes	Yes
FormName	Yes	No
FrenchNumText	Yes	No
FullFileName	Yes	Yes
GetAttachVAR	No	Yes
GetData	No	Yes
GetFormAttrib	Yes	Yes
GetINIBool	Yes	Yes
GetINIString	Yes	Yes
GetListElem *	Yes	Yes
GetOvFlwSym	No	Yes
GetValue	No	Yes
GroupName	Yes	Yes
GVM	Yes	Yes
HaveField	Yes	No
HaveForm	Yes	No
HaveGroup	Yes	No
HaveGVM	Yes	Yes
HaveImage	Yes	No
HaveLogo	Yes	Yes
HaveRecip	No	Yes
Hex2Dec	Yes	Yes
Hour	Yes	Yes
ImageName	Yes	Yes
ImageRect	Yes	Yes

* While these XML-related functions affect both Documaker Workstation and Documaker Server, Documaker Server can have a default variable that refers to the transaction loaded in the XML extract. No such variable would exist automatically within Documaker Workstation.

Function/Procedure	Affects form entry (Documaker Workstation)	Affects form processing (Documaker Server)
IncOvFlwSym	No	Yes
InlineLogo	Yes	Yes
Input	Yes	No
Insert	Yes	Yes
INT	Yes	Yes
IsPrintObject	Yes	Yes
IsXMLError *	Yes	Yes
JCenter	Yes	Yes
JLeft	Yes	Yes
JRight	Yes	Yes
JustField	Yes	Yes
KickToWIP	Yes	Yes
LeapYear	Yes	Yes
Left	Yes	Yes
LEN	Yes	Yes
ListInList	Yes	Yes
LoadINIFile	Yes	Yes
LoadLib	Yes	Yes
LoadXMLList *	Yes	Yes
Logo	Yes	Yes
Lower	Yes	Yes
MailWIP	Yes	No
MajorVersion	Yes	Yes
MAX	Yes	Yes
MIN	Yes	Yes

* While these XML-related functions affect both Documaker Workstation and Documaker Server, Documaker Server can have a default variable that refers to the transaction loaded in the XML extract. No such variable would exist automatically within Documaker Workstation.

Function/Procedure	Affects form entry (Documaker Workstation)	Affects form processing (Documaker Server)
MinorVersion	Yes	Yes
Minute	Yes	Yes
MLEInput	Yes	No
MLETranslate	Yes	No
MOD	No	Yes
Month	Yes	Yes
MonthName	Yes	Yes
MSG	Yes	No
NL	Yes	Yes
NUM	Yes	Yes
Numeric	Yes	Yes
NumText	Yes	Yes
PAD	Yes	Yes
PageImage	No	Yes
PageInfo	No	Yes
PaginateForm	Yes	No
ParseListCount	Yes	Yes
ParseListItem	Yes	Yes
PathCreate	Yes	Yes
PathExist	Yes	Yes
POW	Yes	Yes
Print	Yes	No
Print_It	Yes	Yes
PrinterClass	Yes	Yes
PrinterGroup	Yes	Yes

* While these XML-related functions affect both Documaker Workstation and Documaker Server, Documaker Server can have a default variable that refers to the transaction loaded in the XML extract. No such variable would exist automatically within Documaker Workstation.

Function/Procedure	Affects form entry (Documaker Workstation)	Affects form processing (Documaker Server)
PrinterID	No	Yes
PrinterOutputSize	No	Yes
PutFormAttrib	Yes	Yes
PutINIBool	Yes	Yes
PutINIString	Yes	Yes
RecipBatch	Yes	Yes
RecipCopyCount	Yes	Yes
RecipientName	No	Yes
RecipName	No	Yes
Refresh	Yes	No
RemoveAttachVAR	No	Yes
RenameLogo	Yes	Yes
ResetFld	Yes	Yes
ResetOvFlwSym	No	Yes
Retain	Yes	Yes
Right	Yes	Yes
RootName	Yes	Yes
Round	Yes	Yes
RouteWIP	Yes	No
RPErrorMsg	No	Yes
RPLogMsg	No	Yes
RPWarningMsg	No	Yes
SaveINIFile	Yes	Yes
SaveWIP	Yes	No
Second	Yes	Yes

* While these XML-related functions affect both Documaker Workstation and Documaker Server, Documaker Server can have a default variable that refers to the transaction loaded in the XML extract. No such variable would exist automatically within Documaker Workstation.

Function/Procedure	Affects form entry (Documaker Workstation)	Affects form processing (Documaker Server)
SetDeviceName	Yes	Yes
SetEdit	Yes	No
SetFld	Yes	Yes
SetFont	Yes	Yes
SetFormDesc	Yes	Yes
SetGVM	Yes	Yes
SetImagePos	Yes	Yes
SetLink	Yes	Yes
SetProtect	Yes	Yes
SetRecip	Yes	No
SetRequiredFld	Yes	Yes
SetWIPFld	Yes	No
Size	Yes	Yes
SlipAppend	Yes	No
SlipInsert	Yes	No
SpanField	Yes	Yes
SrchData	No	Yes
STR	Yes	Yes
STRCompare	Yes	Yes
SUB	Yes	Yes
SUM	Yes	Yes
SuppressBanner	No	Yes
Table	Yes	No
Time	Yes	Yes
Time2Time	Yes	Yes

* While these XML-related functions affect both Documaker Workstation and Documaker Server, Documaker Server can have a default variable that refers to the transaction loaded in the XML extract. No such variable would exist automatically within Documaker Workstation.

Function/Procedure	Affects form entry (Documaker Workstation)	Affects form processing (Documaker Server)
TimeAdd	Yes	Yes
TimeZone	Yes	Yes
TimeZone2TimeZone	Yes	Yes
TotalPages	No	Yes
TotalSheets	Yes	Yes
TriggerFormName	No	Yes
TriggerImageName	No	Yes
TriggerRecsPerOvFlw	No	Yes
Trim	Yes	Yes
Upper	Yes	Yes
UniqueString	Yes	Yes
UserID	Yes	No
UserLvl	Yes	No
WeekDay	Yes	Yes
WhatForm	Yes	Yes
WhatGroup	Yes	Yes
WhatImage	Yes	Yes
WIPExit	Yes	No
WIPFld	Yes	No
WIPKey1	Yes	No
WIPKey2	Yes	No
WIPKeyID	Yes	No
XMLAttrName *	Yes	Yes
XMLAttrValue *	Yes	Yes
XMLFind *	Yes	Yes

* While these XML-related functions affect both Documaker Workstation and Documaker Server, Documaker Server can have a default variable that refers to the transaction loaded in the XML extract. No such variable would exist automatically within Documaker Workstation.

Function/Procedure	Affects form entry (Documaker Workstation)	Affects form processing (Documaker Server)
XMLFirst *	Yes	Yes
XMLFirstAttrib *	Yes	Yes
XMLFirstText *	Yes	Yes
XMLGetCurName *	Yes	Yes
XMLGetCurText *	Yes	Yes
XMLNext *	Yes	Yes
XMLNextAttrib *	Yes	Yes
XMLNextText *	Yes	Yes
XMLNthAttrName *	Yes	Yes
XMLNthAttrValue *	Yes	Yes
XMLNthText *	Yes	Yes
Year	Yes	Yes
YearDay	Yes	Yes

* While these XML-related functions affect both Documaker Workstation and Documaker Server, Documaker Server can have a default variable that refers to the transaction loaded in the XML extract. No such variable would exist automatically within Documaker Workstation.

@

Use this function to return the current value contained in a section field. The @ function is also called the *get field* function. The @ symbol is used because it is easy to recognize in script statements and it reduces the amount of typing required.

You can use this function to get text values from the special page numbering fields, FORMSET PAGE NUM, FORMSET PAGE NUM OF, FORM PAGE NUM, and FORM PAGE NUM OF.

NOTE: Although you can also set these page numbering fields, these fields are maintained by the system and the value you set them to will be overwritten.

You can also use this function to get page number field values within scripts that execute during the batch printing process. You can use this, for instance, during the Banner processing with the GenPrint program to check the page number fields on certain pages.

Keep in mind that during GenData processing, page numbering is not usually done unless you are also doing single-step printing. Even then, page numbering does not occur until the print process begins.

Syntax

```
@(Field, Section, Form, Group)
```

Parameter	Description
Field	Enter the name of a section field. The default is the current field.
Section	Enter the name of a section that contains the field named. The default is the current section.
Form	Enter the name of a form that contains the section and/or field named. The default is the current form.
Group	Enter the name of the form group that contains the form, section, or field. The default is the current group.

The system uses the parameters you provide to search for one field on a section and return that field's data. If the field is defined as a numeric data type, the system returns a number. Otherwise, the result is a string of text.

NOTE: If you omit the Field parameter, make sure you include quotation marks, as shown in the second and third example below.

Example

For these examples, assume the current field value is 1234.23 and is named MyField. Also, assume that a second occurrence of MyField appears on the form, MyForm, and contains the value *automobile*.

For the third example, assume the current form is the third page of the form set being processed. For the fourth example, assume the section *Header3* is on the second page of the form *ABC*.

Function	Result	Explanation
Return(@( ))	1234.23	Returns the value in the current field.
Return(@("MyField"))	1234.23	Returns the value in the named field, located on the current section.
Return (@("Formset Page Num"))	3	Returns the value in the field named "Formset Page Num" on the current section.
Return (@(Form Page Num"), "Header3","ABC"))	2	Returns the value in the field named "Form Page Num" in section, "Header3" on form "ABC".

See also

Field Functions on page 61

Field Formats on page 62

Locating Fields on page 64

NUM on page 325

?

Use this function to retrieve data from a record in the extract file. This function only uses the specified entry (LookUpName) in the XDB database to determine the:

- Rule to use to retrieve the data (the default is the Move_It rule)
- Search mask to use
- Offset and length
- Format mask

Syntax

```
?(LookUpName, Occurrence)
```

Parameter	Description
LookUpName	Specify the entry name in the XDB that defines the data to retrieve.
Occurrence	(Optional) Define which occurrence-record in the extract file to retrieve data from. You can omit this parameter for the first occurrence.

NOTE: Keep in mind the XDB database must be structured to handle symbolic lookup. See the Using Dictionaries chapter in the Documaker Studio User Guide, which describes how to define extract file records and fields in the XDB database.

Assume you have these entries defined in the XDB:

Spreadsheet View

Name	Parent	Offset	Length	Conditional	Required	Rule	Mask	Data
Pol_Rec		0	0	No	Not			1,PolRec
Pol_Num	POL_REC	10	10	No	Not	MOVE_IT		
M_Initial	POL_REC	50	1	No	Not	MOVE_IT		
L_Name	POL_REC	55	25	No	Not			
Issue_Date	POL_REC	90	6	No	Not	DATEFMT	11	
F_Name	POL_REC	20	25	No	Not	MOVE_IT		

And the extract record, *pol_rec*, has the following data:

```
0...    1...     2...    5...  5...     9...
1...    0...     0...    0...  5...     0...

PolRec  GRA0001  Morris  V     Vanelli  09221957
PolRec  GRA0001  Sandra  J     Vanelli  09211959
PolRec  GRA0001  Vincent M     Vanelli  12311981
```

Example 1

Assume the Driver field has this script and uses the DAL rule.

```
Return ( ?("f_name") & " " & ?("m_initial") & ". " & ("l_name") );
```

The DAL script retrieves data from the XDB entries (1_f_name, 1_m_initial, and 1_l_name), which it concatenates with spaces and a period to form the driver's name. The result is shown here:

```
Morris V. Vanelli
```

Example 2

In this example, assume there are ten fields (driver01, driver02, and so on) on the section, the first field includes this script and it uses the DAL rule.

```
Call ("drivers.dal")
```

The external DAL script (DRIVERS.DAL) contains these statements:

```
*   Determine number of 'pol_rec' records exist in the transaction.

#drivers = CountRec("?pol_rec");
#occur   = 1;

*   Create the driver's full name and store in appropriate *
*   field.

While (#occur !> #drivers)
   d_name = ( ?("f_name", #occur) & " " & ?("m_initial" , #occur) /
            & ". " & ("l_name" , #occur) );
   field_name = "driver" & Format(#occur, 'n', '99');
   SetFld (d_name, field_name);
   #occur += 1;
Wend;
```

This script determines there are three (3) records and would loop three (3) times; creating the driver's name and storing it in the proper field. The results are shown here:

```
Morris V. Vanelli
Sandra J. Vanelli
Vincent M. Vanelli
```

Example 3

In this example, assume the License Issued field has this script and uses the DAL rule.

```
Return ( ?("issue_date") );
```

The DateFmt rule would be executed using specified format (11) and would return this result to the field:

```
September 21, 1957
```

See also

FieldRule on page 235

GetData on page 253

Documaker Server Functions on page 58

Field Formats on page 62

Locating Fields on page 64

ABS

Use this function to return the absolute value of a number. The absolute value of a number is its positive value.

Syntax

```
ABS (Number)
```

Parameter	Description
Number	Enter a number data type. The default is the value of the current field.

The system returns the absolute value of a number. Absolute values are always positive numbers.

Example

Here are some examples:

(Assume the current field contains the number 250.)

Function	Result	Explanation
Return(ABS ( ))	250	Defaults to the current field.
Return(ABS (-101.25))	101.25	Returns the absolute value of the given value. Note that this function retains the decimal.
Return(ABS (10 / -2))	5	10 is divided by -2 resulting in -5. The absolute value of -5 is returned.

See also Mathematical Functions on page 72

AddAttachVAR

Use this function to add a string value as an attachment variable. You can use this function when creating print comments using Documaker Bridge.

Syntax

```
AddAttachVAR (Name,Value,DSIqueue)
```

Parameter	Description
Name	Enter the name of the attachment variable.
Value	Enter the value you want to add.
DSIqueue	(Optional) Enter one (1) for input or two (2) for output. The default is two (2).

The system returns one (1) on success or zero (0) on failure.

See also Docupresentment Functions on page 60

GetAttachVAR on page 252

RemoveAttachVAR on page 358

AddBlankPages

Use this procedure to add blank or filler pages to a form set. You add these pages to make sure each physical printed page has a front and back. This lets you change a simplex form set or a form set which contains both simplex and duplex forms into a fully duplexed form set.

For instance, you can use this to make it easier to add OMR marks, which are often printed on the back, to simplex forms.

Syntax

```
AddBlankPages (FAP)
```

Parameter	Description
FAP	Enter the name of the FAP file you want the system to use as a filler page. The default is blank.

Omit the path and extension of the FAP file.

Example

One way to add blank pages is by using banner page processing in the GenPrint program. You can specify a DAL script which runs at the start of each transaction. The DAL script calls the AddBlankPages procedure.

This tells the system to convert each transaction into a fully duplexed form set with blank pages added as needed. To do this, you need these INI settings:

```
< Printer >
    EnableTransBanner = TRUE
    TransBannerBeginScript = PreBatch
< DALLibraries >
    LIB = BANNER
```

Here is an example of the BANNER.DAL file:

```
BeginSub PreBatch
AddBlankPages()
EndSub
```

NOTE: See Documaker Administration Guide for more information on using banner processing.

Here is a table which shows when blank pages will be added, based on the duplex setting of the two current pages and the duplex setting of the next page. *Blank* means a blank page will be added, *As is* means no blank page is needed and the form will be left as is.

	And the next page is					
If the current page is	**Unknown**	**Front**	**Back**	**None**	**Short**	**Rolling**
Unknown	Blank	Blank	As is	Blank	Blank	Blank
None	Blank	Blank	As is	Blank	Blank	Blank
Front	Blank	Blank	As is	Blank	Blank	As is
Short	Blank	Blank	As is	Blank	Blank	As is
Rolling (Front)	Blank	Blank	As is	Blank	Blank	As is
Back	As is	As is	Blank	As is	As is	As is
Rolling (Back)	As is	As is	Blank	As is	As is	As is

NOTE: You can also add blank or filler pages using custom code or by using the DPRAddBlankPages function, which is available with Docupresentment. See Using the Documaker Bridge for more information on the DPRAddBlankPages function.

The API to call from custom code is as follows:

```
DWORD _VMMAPI FAPAddBlankPages(
                VMMHANDLE objectH,      /* formset or form handle */
                char FAR * imagename)    /* if NULL, "Blank Page" */
```

If the section name is NULL, a blank page is created when a filler page is needed. If the section name is not NULL, the section name is loaded when a filler page is needed. If you include a section name, include only the name of the FAP file—omit the path and file extension.

See also

DelBlankPages on page 208

SuppressBanner on page 403

Page Functions on page 75

Miscellaneous Functions on page 73

Creating a DAL Script Library on page 5

AddComment

Use this procedure to add a comment to the print stream. Products like Oracle Insurance's Docusave and IBM's OnDemand use comments in the print stream as an archive key.

In addition, you can also use this procedure to add comments to your PCL print string using PJL (Printer Job Language). PJL commands are supported by most PCL printers.

You call the AddComment procedure from an external script loaded using an INI option in the printer group. Here are some examples:

```
< PrtType:PCL >
    PJLCommentScript   = name of the external DAL script
< PrtType:AFP >
    OnDemandScript     = name of the external DAL script
    DocusaveScript     = name of the external DAL script
< PrtType:XER >
    DocusaveScript     = name of the external DAL script
```

If you call AddComment from the GenData program, you will receive an error. For more examples see DAL Script Examples on page 35.

Syntax

```
AddComment (Comment, Convert)
```

Parameter	Description
Comment	Enter the string you want used as a comment in the print stream or the name of a section variable field that contains the comment.
Convert	Enter one of these options: 0 - (zero) convert the string to EBCDIC 1 - convert the string to ASCII 2 - do not convert the string For OnDemand, you will always want EBCDIC comments. The default is zero (0).

Example

Here are some examples:

Procedure	Result	Explanation
AddComment('This is an example')	1 or 0	Adds the comment, "This is an example", to the print stream.
* Add a comment to PCL print stream Comment = AppIdxRec( ); AddComment (comment, 1);	1 or 0	Adds a comment containing the archive record ID. The second parameter (1) indicates that the string is to be added as an ASCII string.

See also

Printer and Recipient Functions on page 76

AddDocusaveComment

Use this procedure to add a Docusave comment to the print stream. Docusave uses comments in the print stream as an archive key.

You should only call this procedure from a script loaded via the DocusaveScript specified in the AFP, Metacode, or PCL printer control group.

If you call this procedure from the GenData program, DAL will return an internal error.

Syntax

```
AddDocusaveComment (Comment, Convert)
```

Parameter	Description
Comment	Enter the string to be written as a comment in the print stream.
Convert	(Optional) Choose from these options: 0 - (zero) convert the string to EBCDIC 1 - convert the string to ASCII 2 - do not convert the string. For Docusave, you will always want EBCDIC comments. The default is zero (0).

Example

Here are some examples:

```
AddDocusaveComment('This is an example')
AddDocusaveComment(@('INSURED NAME',,, GROUPNAME()))
```

See also

Printer and Recipient Functions on page 76

DAL Script Examples on page 35

AddForm

Use this procedure/function to add a new form to a document.

Syntax

```
AddForm (Form, Insert, Group)
```

Parameter	Description
Form	Enter the name of a form in the specified group.
Insert	Enter the name of a form *after* which the new form should be inserted. The default is to append after the last form in the group.
Group	Enter the name of a group to contain the specified form. The default is the current group.

The system optionally returns one (1) on success or zero (0) on failure.

This procedure adds a new copy of the specified form to the document set. The form named must be a valid form for the given group. You cannot add a form defined for one group into another group. The insert (form) name may be specified using the occurrence indicator.

If you include the Group parameter, it must reference a group included in the form set. You cannot add a group or add forms to a group that was not specified during form selection.

Example

Here are some examples:

Procedure	Result	Explanation
AddForm(“Form1”)	1 or 0	Add the named form after the last form in the current group.
AddForm(“Form”, “Form\1”, GRP”)	1 or 0	Insert the named form after the first occurrence of that form within the named group.

See also

AddForm_Propagate on page 120

CopyForm on page 174

DupForm on page 228

TriggerForm on page 414

WIP Functions on page 88

AddForm_Propagate

Use this procedure/function to add a new form to a document and propagate global data onto the new form.

Syntax

```
AddForm_Propagate (Form, Insert, Group)
```

Parameter	Description
Form	Enter the name of a form in the specified group.
Insert	Enter the name of a form after which the new form should be inserted. The default is to append after the last form in the group.
Group	Enter the name of a group to contain the specified form. The default is the current group.

The system optionally returns one (1) on success or zero (0) on failure.

The form named must be a valid form for the given group. You cannot add a form defined for one group into another group. You can specify the insert (form) name using the occurrence indicator.

If you include the Group parameter, it must reference a group included in the form set. You cannot add a group or add forms to a group that were not specified during form selection.

Keep in mind...

- This procedure should only be used from GenData. For Documaker Workstation, use AddForm. If called from Documaker Workstation, AddForm_Propagate works exactly like AddForm.
- Global multi-line variable field data *is not* propagated to the added form.
- If you use this procedure to add forms and you also import and export those forms, be sure to set the IgnoreInvalidImage option in the ImpFile_CD control group. Otherwise, users will receive an error message. For detailed instructions, see the Documaker Workstation Administration Guide.

Example

Here are some examples:

Procedure	Results	Explanation
AddForm_Propagate(“0002EA”)	1 (success) or 0 (failed)	Add the named form, 0002EA, after the last form in the current group.
AddForm_Propagate (“C22510WGIM”, “C22510WGIM \1”, Sales”)	1 (success) or 0 (failed)	Insert the named form, C22510WGIM, after the first occurrence of specified form, C22510WGIM, within the named group, Sales. See sample output.

Original form: C22510WGIM

Employer:	Oracle Insurance	form name =	C22510WGI M
Employee:	J. Stewart	section name =	GENRCHDR
Date of Loss:	12/11/10		
File Number:	12345		
State Case Num:			

Samford and Son

Sincerely,

Workers' Compensation Unit

cc: J. Stewart

Added form: C22510WGIM\2

Note the missing data (CC: J. Stewart4) for the field, Copies, that has section scope. The fields, employer, employee, date of lost, and file number, that are defined as global scope appear on the added form, C22510WGIM\2.

Employer:	Oracle Insurance	form name =	C22510WGIM\2
Employee:	J. Stewart	section name =	GENRCHDR
Date of Loss:	12/11/10		
File Number:	12345		
State Case Num:			

Samford and Son

Sincerely,

Workers' Compensation Unit

See also

AddForm on page 119

CopyForm on page 174

DupForm on page 228

TriggerForm on page 414

WIP Functions on page 88

AddImage

Use this procedure/function to add a new section to a form in the current document. You can also use the Paginate parameter to specify whether form pagination should occur after the section is added. Form pagination includes the application of section origin rules to determine whether new pages are required for the pre-defined page sizes.

Syntax

```
AddImage (FAP, Section, Form, Group, Flag, Paginate)
```

Parameter	Description
FAP	Enter the name of the section file to load and add to the form.
Section	Enter the name of a section which will precede the new section. The default is the current section.
Form	Enter the name of a form in the form set. If you specify the Section parameter, that section must occur on this form. The default is the current form.
Group	Enter the name of a group that contains the specified form. The default is the current group.
Flag	Determines if the section is inserted on the same page or on a new page. 0 - (zero) new page 1 - same page The default is zero (0).
Paginate	(Optional) This parameter follows the Flag parameter. If you enter anything other than a zero (0), it tells the system you do want form pagination to occur upon successful inclusion of the new section. If the section does contain an origin rule and you omit the Paginate option or set it to zero (0), the section origin rule executes upon insertion. Whether the inserted section has an origin rule or not, the positioning of this section when the Paginate option is omitted or zero (0) does not cause the entire form to be re-paginated. This means if the placement of the section causes it to overlap another section or to be out of the page boundary, no additional re-pagination occurs. If you are manipulating multiple sections in series, you may want to conclude your script with a call to PaginateForm to make sure the entire form is re-paginated. Here is an example: `AddlImage( "myFAP", "mainImage" , , , 1,1)` This example omits the Form and Group parameters, but does specify the Flag parameter as well as the Pagination parameter. Note: If you enter zero (0) or omit this parameter, the function works as it prior to version 11.2. The default is zero (0).

The system optionally returns one (1) on success or zero (0) on failure.

This procedure adds a copy of the section you specify to a form. The system loads the new section onto the page after the section, form, or group you specified or onto a new page which it creates *after* the section, form, or group you specified. The section added does not have to be predefined for the form.

NOTE: If you use this procedure to add sections to forms and you also plan to import and export those forms, be sure to set the IgnoreInvalidImage option in the ImpFile_cd control group in the FSISYS.INI file. Otherwise, users will receive an error message. For detailed instructions, see the Documaker Workstation Administration Guide.

Any section you add using this procedure is positioned the same way as other sections. The specific location of sections is determined by your master resource setup.

NOTE: If the section parameter specifies one of multiple sections on the same page, the new section is added after the section, form, or group you specified. The system does not move sections already defined for a page. Therefore, you can overlay existing sections on the page. Make sure you do not unintentionally overlay an existing section. Move the new section using the ImageRect and SetImagePos procedures.

Use the Refresh procedure after this procedure to refresh the screen display.

NOTE: When adding a section, there is no way for you to specify what section options or recipients you want included on the new section. So, the AddImage procedure takes the missing information from an associated section.

The system will, however, exclude the In-lined, Copy on Overflow, Duplex Front, Duplex Back, and Caused by Overflow settings. These options are not normally associated with a section being added via DAL.

Example

Here are some examples:

Procedure	Result	Explanation
AddImage (“IMG1”)	1 - if successfully added. 0 - if not added.	Insert the named section, IMG1, on a new page after the current page.
AddImage(“NEW1”, “IMG\3”,,”GRP”)	1 - if successfully added. 0 - if not added.	Insert the named section, NEW1, after the third occurrence of IMG, within GRP. This section is placed on a new page after the third occurrence of the specified section.
AddImage (“IMG1”,,,, 1)	1 - if successfully added. 0 - if not added.	Insert the named section, IMG1, after the current section on the same page.
AddImage(“NEW1”, “IMG\3”, , , 1)	1 - if successfully added. 0 - if not added.	Insert the named section, NEW1, after the third occurrence of IMG on the same page.

See also

DelImage on page 212

ImageRect on page 278

PaginateForm on page 333

SetImagePos on page 382

Refresh on page 357

Section Functions on page 77

AddImage_Propagate

Use this procedure/function during GenData processing to add a new section and propagate global data onto the newly added section as needed.

NOTE: This DAL procedure should only be used with the GenData program. Documaker Workstation users should use the AddImage procedure. If called from Documaker Workstation, this procedure will work exactly like the AddImage procedure.

Syntax

```
AddImage_Propagate (FAP, Section, Form, Group, Flag)
```

Parameter	Description
FAP	Enter the name of the section file to load and add to the form.
Section	Enter the name of a section which will precede the new section. The default is the current section.
Form	Enter the name of a form in the form set. If you specify the Section parameter, that section must occur on this form. The default is the current form.
Group	Enter the name of a group that contains the specified form. The default is the current group.
Flag	Determines if the section is inserted on the same page or on a new page. 0 - new page 1 - same page The default is zero (0).

Optionally, this procedure returns one (1) on success or zero (0) on failure.

This procedure adds a copy of the section you specify to a form. The system loads the new section onto the page after the section, form, or group you specified or onto a new page which it creates *after* the section, form, or group you specified. The section added does not have to be predefined for the form.

Keep in mind...

- Global multi-line variable field data is not propagated to the added form.
- The system does not move sections already defined for a page. Therefore, you can overlay existing sections on the page.
- Make sure you do not unintentionally overlay an existing section. You can move the new section using the ImageRect and SetImagePos procedures.
- If you use this procedure to add sections to forms and you also import and export those forms, be sure to set the IgnoreInvalidImage option in the ImpFile_CD control group. Otherwise, users will receive an error message. For detailed instructions, see the Documaker Workstation Administration Guide.

- When adding a section, there is no way for you to specify what section options or recipients you want included on the new section. So, the AddImage_Propagate procedure takes the missing information from an associated section. The system will, however, exclude the In-lined, Copy on Overflow, Duplex Front, Duplex Back, and Caused by Overflow settings. These options are not normally associated with a section being added via DAL.

Example

Here are some examples:

Procedure	Result	Explanation
AddImage_Propagate (“IMG1”)	1 - if successfully added. 0 - if not added.	Insert the named section, IMG1, on a new page after the current page.
AddImage_Propagate(“ NEW1”, “IMG\3”,,”GRP”)	1 - if successfully added. 0 - if not added.	Insert the named section, NEW1, after the third occurrence of IMG, within GRP. This section is placed on a new page after the third occurrence of the specified section.
AddImage_Propagate (“IMG1”,,,, 1)	1 - if successfully added. 0 - if not added.	Insert the named section, IMG1, after the current section on the same page.
AddImage_Propagate(“ NEW1”, “IMG\3”, , , 1)	1 - if successfully added. 0 - if not added.	Insert the named section, NEW1, after the third occurrence of IMG on the same page.

See also

AddImage on page 122

WIP Functions on page 88

AddOvFlwSym

Use this procedure/function to create an overflow symbol. This procedure provides DAL with an equivalent to the Documaker Server SetOvFlwSym rule that is placed in the AFGJOB.JDT file.

Syntax

```
AddOvFlwSym (Form, Symbol, MaxRecords)
```

Parameter	Description
Form	Enter the name of the form that contains the fields on which overflow processing will occur.
Symbol	Enter the character you want to use as the overflow symbol.
MaxRecords	Enter the maximum number of overflow records to be processed for the section per page of output.

The system optionally returns one (1) on success or zero (0) on failure.

This procedure creates an overflow symbol associated with the section you specified.

Example

Here are some examples:

Assume that the section, CP0101NL, has three overflow lines and the extract file is a standard Documaker Server extract file.

```
#add_rc = AddOvFlwSym ("CP0101NL", "Loc_Cnt", 3)
```

In this example, an overflow variable called *Loc_Cnt* would be associated with the section, *CP0101NL* and the number of overflow lines would be set to three (3). The DAL integer variable, *#add_rc*, would be set to a one (1) on success or zero (0) on failure.

You define the search mask for the field or the XDB name associated with the field, as follows:

```
@GetRecUsed, CP0101NL, Loc_Cnt/10,HeaderRec 50,20
```

Here is another example:

Assume the extract file is in XML format and includes an element/node, Location, that can repeats or occurs multiple times.

```
AddOvFlwSym ("Loc_Cnt", "XML")
```

In this example, an overflow variable called *Loc_Cnt* would be defined. You would use this variable in the XPath *predicate* for repeating elements/nodes. You would define the XPath search mask for the field or the XDB name associated with the field, as follows:

```
!/DOCC/InsuranceSvcRq/PolicyPrintRq/
ClPropLineBusiness[**Loc_Cnt**]/Location
```

See also

GetOvFlwSym on page 262

IncOvFlwSym on page 280

ResetOvFlwSym on page 361

Documaker Server Functions on page 58

AddresseeCount

Use this function to determine the number of addressees for a named recipient parameter. Use this function with recipients using the Addressee Map capability and with the GetAddresseeValues function.

Syntax

```
AddresseeCount(RecipName)
```

Parameter	Description
RecipName	Enter the name of the recipient for which you want to know the number of addressees. This name should be a recipient name associated with the document set.

The return value is the number of addressees associated with the named recipient. If the recipient named is not part of the document set or has no addressees associated with it, the system returns zero (0).

Note: To determine whether a recipient is actually used in the document set and not the number of addressees, use the HaveRecip function.

Example

Here are some examples.

For this example, the mapped data shows six memo addresses for a given document set. The document set is not distributed to the Agent and the Insured recipient does not support addressees.

Function	Result	Explanation
#Cnt = AddresseeCount(“Memo”)	6	The #Cnt variable contains *6* to reflect the number of unique addressees associated with the memo recipient documents for the document set.
#Cnt = AddresseeCount(“Agent”)	0	The document set is not distributed to the Agent recipient.
#Cnt = AddresseeCount(“Insured”)	0	The Insured recipient does not support Addressees; therefore the returned value is zero (0).

See also

GetAddresseeValues on page 250

HaveRecip on page 274

Have Functions on page 69

AFELOG

Use this procedure/function to write a custom message to the AFELOG file.

Syntax

```
AFELog (String)
```

Parameter	Description
String	Enter a valid string.

This procedure writes a string of characters to the AFELOG file. The message can be up to 100 characters in length.

Example

Here is an example:

Procedure	Result	Explanation
AFELog ("Point1");	Point1	The character string "Point1" is written to the AFELOG file.

```
afelogmsg = INPUT("Input a custom message to be written to the AFELOG
file", "AFELOG Test Case", 100);
RETURN AFELOG(afelogmsg);
```

This DAL script displays a window entitled *AFELOG Test Case* with a message which states:

Input a custom message to be written to the AFELOG file

The input field has a length of up to 100 characters. When the user clicks OK after entering a message, the system writes the message to the AFELOG file. If the user clicks Cancel, blanks are written to the file.

See also

WIP Functions on page 88

Always

Use this function to return TRUE (Always).

Syntax

```
Always ()
```

There are no parameters for this function.

This function is typically used as a placeholder or stub.

Example

```
Always()
```

See also Miscellaneous Functions on page 73

APPEND

The Append function is obsolete and is no longer supported. Use one of these functions instead:

To	Use
Append text into a multi-line field from an external multi-line text area.	AppendTxm
Append text into a multi-line field from an external multi-line text area and rename the fields imported from the external text area so they have unique names.	AppendTxmUnique

See also Field Functions on page 61

APPENDTEXT

Use this procedure/function to attach additional text to the end of a multi-line text field from an external ASCII text file. This procedure only works on multi-line text fields.

Syntax

```
AppendText (File, Field, Section, Form, Group)
```

Parameter	Description
File	Enter the name of an external text file including any file extension. This text is appended to the field you specify.
Field	Enter the name of a field that identifies a multi-line text area. This is the field that receives the appended text. The default is the current field.
Section	Enter the name of a section that contains the field you specified. The default is the current section.
Form	Enter the name of a form that contains the section and/or field you specified. The default is the current form.
Group	Enter the name of the form group that contains the form, section, and/or field you specified. The default is the current group.

The system optionally returns one (1) on success or zero (0) on failure.

This procedure opens the external text file and appends the text from that file to the specified field. If successful, the newly inserted text is reformatted appropriately in the destination field.

If the external text file name does not include a specific path, the system tries to locate the file in the default directory where form sections are typically found.

When used with Documaker Workstation, use the Refresh procedure to make sure all appended text appears in the field.

Example

Here are some examples:

Procedure	Result	Explanation
AppendText (“MyFile.txt”)	1 or 0	The current field receives the text from the file named, *MyFile.Txt*. The named file does not specify a path, therefore the system tries to locate the file where form sections are normally located.
AppendText (“C:\MyFile.txt ”, “MyField”)	1 or 0	*MyField* will be located on the current section. If found, the field receives the text from the file named, *MyFile.Txt*. The named file specifies a path, therefore the system looks for the file in that location.
AppendText (“MyFile.txt”, “MyField”, , “MyForm”)	1 or 0	The field, *MyField*, will be located on the form, *MyForm*. Since a section was not specified, it may occur on any section on that form. Once located, the text from the specified file is appended to the field.

See also

Field Functions on page 61

Field Formats on page 62

Locating Fields on page 64

Refresh on page 357

AppendTxm

Use this procedure/function to append text to the end of a multi-line text field from a text area on another section (FAP) file. This procedure only works on multi-line text fields.

Syntax

```
AppendTxm (FAP, InsertFld, Field, Section, Form, Group)
```

Parameter	Description
FAP	Enter the name of the section file which contains the text area you want to append to the field you specify in the Field parameter. If you omit the path, the system looks for this section in the forms directory you specified using the File, Library Setup option.
InsertFld	This parameter determines where in the tabbing sequence any embedded variable fields will be placed. Use this parameter to specify the name of the variable field (on the current section) before which you want the embedded fields in the imported text area inserted. For example, if your form contains three variable fields (Y1, Y2, Y3). The text area to be inserted contains two variable fields (Z1, Z2). By specifying Y2 as the InsertFld, you tell the system to tab to fields Z1 and Z2 before tabbing to Y2 when in entry mode. The default is to append after the last field on the section.
Field	Enter the name of the field that identifies the multi-line text area which will receive the appended text. The default is the current field, which *must* be a multi-line text field.
Section	Enter the name of the section that contains the field you specified in the Field parameter. The default is the current section.
Form	Enter the name of the form that contains the section you specified in the Section parameter. The default is the current form.
Group	Enter the name of the group that contains the form you specified in the Form parameter. The default is the current group.

The system optionally returns a one (1) if successful and zero (0) if unsuccessful.

This procedure opens the section you defined in the FAP parameter and copies the text from the *first* text area field found on that section. It then appends that text in the field you specified in the Field parameter. If necessary, the text will be reformatted appropriately for the destination field.

When used with Documaker Workstation, use the Refresh procedure to make sure all appended text appears in the field.

Example

Here are some examples:

Procedure	Result	Explanation
#rc = AppendTxm ("Message", , "Name_Line"); Refresh ( );	1 or 0	The text in the first text area on the section named *Message* is appended to the multi-line text field, called *Name_Line*. The system then refreshes the display.
#rc = AppendTxm (".\mstrres\messages\msg 1", , "Name_Line", "Mailer"); Refresh ( );	1 or 0	The path, .*mstrres**message*\\, is appended to the multi-line text field, called *Name_Line*, which is on the section named *Mailer*. The system then refreshes the display.
#rc = AppendTxm("message", "Address1" , "Name_Line"); Refresh ( );	1 or 0	The fields in the text area are inserted before the variable field named *Address1*, in the tabbing sequence. The system then refreshes the display.

See also

Field Functions on page 61

Field Formats on page 62

Locating Fields on page 64

AppendTxmUnique on page 136

Refresh on page 357

AppendTxmUnique

Use this procedure/function to append text into a multi-line field from an external text area. This procedure also renames the fields imported from the external text area so they have unique names. You can use this procedure in these specific situations:

- This procedure lets you import paragraphs with embedded fields, when you know that those fields should never inherit data from the existing section and you expect the user to tab through the imported field and enter new data.
- This procedure lets you import the same section multiple times and have the field data for each instance uniquely named.

NOTE: When it renames fields, this procedure makes sure the field names are unique for the entire form, not just the section that contains the text area. This prevents naming conflicts with prior sections.

This procedure only works on multi-line text fields.

Syntax

```
AppendTxmUnique (FAP, InsertFld, Field, Section, Form, Group)
```

Parameter	Description
FAP	Enter the name of a section file that contain a text area. This text area is appended to the field specified. If there are several text areas in the section file, the system grabs the text from the first text area it finds. If you omit the path, the system looks for the form in the forms directory specified using the File, Library Setup option.
InsertFld	Enter the name of a field before which you want the embedded fields in the imported text area inserted. The default is to append after the last field in the section.
Field	Enter the name of the field that identifies a multi-line text area. This is the field that receives the appended text. The default is the current field.
Section	Enter the name of a section that contains the field. The default is the current section.
Form	Enter the name of a form that contains the section and/or field. The default is the current form.
Group	Enter the name of the group that contains the form, section, and/or field. The default is the current group.

The system optionally returns a one (1) if successful and zero (0) if unsuccessful.

This procedure opens the section and appends the text from the first text area field found in that section. If successful, the new text will be formatted appropriately in the destination field.

The system locates the section in the directory where sections (FAP files) are typically found. Use the Refresh procedure to make sure all appended text appears in the field.

This procedure is similar to the AppendTxm procedure. For instance, suppose your paragraph section looks like this, where X1 is a reference to the Name field and X2 is a reference to the City field.

```
X1 of X2. X1 please let me know if you received this by mistake.
```

The Name field is embedded twice and the City field once. If you were to use the AppendTxm procedure on this section three times, the result would look like this:

```
X1 of X2. X1 let me know if you received this by mistake.
X1 of X2. X1 let me know if you received this by mistake.
X1 of X2. X1 let me know if you received this by mistake.
```

However, there would be only one Name and City field defined on the section. If you set Name to Tom and City to Marietta, the paragraph would look like this.

```
Tom of Marietta. Tom let me know if you receive this by mistake.
Tom of Marietta. Tom let me know if you receive this by mistake.
Tom of Marietta. Tom let me know if you receive this by mistake.
```

Using the AppendTxmUnique procedure, if you append this section three times, the first line would likely still reference Name and City (it would depend upon whether there was already a field named Name or City on the section).

The second occurrence however, would be renamed to NAME #002 and CITY #002. The third occurrence would be renamed to NAME #003 and CITY #003.

So, instead of two fields, you now have six fields to tab through and each subsequent occurrence can hold a different value.

```
Tom of Marietta. Tom let me know if you receive this by mistake.
John of Athens. John let me know if you receive this by mistake.
Albert of Atlanta. Albert let me know if you receive this by mistake.
```

Notice that multiple references to the same field in a paragraph still associate to the same field. So although there are three embedded locations in each paragraph, there are only two separate fields being referenced.

NOTE: This procedure renames the field uniquely for the entire form, not just the section that contains the multi-line text field. This occurs because a multi-line text field can span pages and you don't want the field names to duplicate.

For instance, suppose you have a paragraph with one embedded field. The first time you append it, it is named *Field* (assuming field is the original name and does not conflict. Each time you append it you get a unique name:

```
FIELD #002
FIELD #003
and so on...
```

Eventually, an AppendTxmUnique procedure could cause a the text to overflow to a new page. Let's assume you were up to *FIELD #010* when that occurred.

If you run the AppendTxmUnique procedure again, the name *FIELD* does not occur on the second page, but it did on the first. You want *FIELD #011* to be next. This is why the names unique at the form level and not the section level.

Example Here are some examples:

Procedure	Result	Explanation
#rc = AppendTxmUnique ("message", , "name_line"); Refresh ( );	1 if successful, 0 if not.	The first text area in the Message FAP file is appended to the *Name_line* multi-line text field.
#rc = AppendTxmUnique (".\mstrres\messages\msg1", , "name_line", "mailer"); Refresh ( );	1 if successful, 0 if not.	The first text area in the MSG1 FAP file located in the \mstrres\message\ directory is appended to the *Name_line* multi-line text field, which is in the Mailer FAP file.
#rc = AppendTxmUnique ("message", "address1" , "name_line"); Refresh ( );	1 if successful, 0 if not.	The first text area in the Message FAP file is appended to the *Name_line* multi-line text field. Any embedded variable fields in the text area are inserted before the Address1 variable field, based on the tabbing sequence.

See also Field Functions on page 61

Locating Objects on page 94

Field Formats on page 62

Locating Fields on page 64

AppendTxm on page 134

Refresh on page 357

AppIdxRec

Use this function to get an archive record based on the APPIDX.DFD file and settings in the Trigger2Archive control group.

Syntax

```
AppIdxRec ()
```

There are no parameters for this function.

Example

Here are some examples. Assume that...

- The rundate is 01/10/2009
- The sub-string of extract record being processed is:

```
SCO1234567HEADERREC00000…
```

- The FORM.DAT file contains the following:

```
;SAMPCO;LB1;Libby;;R;;letter|D<INSURED(1),COMPANY(1),AGENT(1)>;
```

Also assume these INI options exist:

```
< Trigger2Archive >
    Company     = Company
    LOB         = LOB
    PolicyNum   = PolicyNum
    RunDate     = RunDate
```

Function	Result	Explanation
Comment = AppIdxRec() Print_It (Comment)	1 or 0	

See also

Documaker Server Functions on page 58

Print_It on page 342

DAL Script Examples on page 35

ApplyInserts

Use this procedure/function to force the insertion of items associated with applying logos, state stamps, and signatures to a form set.

Normally, you apply a logo, state stamp, or signature when transactions are opened or completed. This procedure lets you trigger the insertions when the user tabs off of the field or a DAL script associated with the field is executed. This lets the user see the form exactly as it would appear when printed or archived.

Syntax

```
ApplyInserts()
```

There are no parameters for this procedure.

Optionally, this procedure returns one (1) on success or zero (0) on failure. A return of one (1) indicates that you had a valid WIP transaction loaded in memory. Success, however, does not mean that any sections were added or changed.

NOTE: See Inserting State Stamps and Signatures in the Documaker Workstation Supervisors Guide for more information on how inserted sections are determined and applied.

Example

Here is an example:

```
ApplyInserts()
```

See also

Section Functions on page 77

Ask

Use this procedure/function to create a message to which the user must respond with *Yes* or *No*. The message is created as a message window for the system interface.

A *Yes* response results in a value of 1. A *No* response or terminating the window without responding results in a value of zero (0).

Syntax

```
Ask (Msgline1, Msgline2, Msgline3, Title, Defans)
```

Parameter	Description
Msgline1	Enter the first line of the message. The default is Yes.
Msgline2	Enter the second line of message.
Msgline3	Enter the third line of message.
Title	Enter the title of message window.
Defans	Enter one (1) or zero (0) to specify which button (Yes or No) should be selected as the default. The default is one (1), which makes the Yes button the default.

This procedure requires a user response each time the script executes. Therefore, use this procedure *only* in scripts that execute *once* during entry. Do not use this procedure for scripts that execute each time a user tabs to a new field.

NOTE: A DAL script executes once during entry. A DAL calc executes each time the user tabs to a new field. You make the DAL script or DAL calc designation in the Properties window.

Example

Here are some examples:

Procedure	Result	Explanation
#result =Ask (“Are you sure you made the correct entry?” , , , “Sample Message”)	1, if the user answers Yes 0, if the user answers No	“Sample Message” is the title of the entry box. “Are you sure you made the correct entry?” is the message the user answers.
#result =Ask (“This is line 1”, “This is line 2”, “Is this line 3?”, “Please Respond”)	1, if the user answers Yes 0, if the user answers No	“Please Respond” is the title of the entry box. '“This is line 1” “This is line 2” “Is this line 3?” is the message the user answers.

See also

Documaker Workstation Functions on page 59

AssignWIP

Use this procedure/function to assign the work-in-process and its associated data to a different user ID.

Syntax

```
AssignWIP (UserID)
```

Parameter	Description
UserID	Enter a valid user ID.

The system returns success if no error occurred during the process, otherwise a failure.

This procedure assigns the current work-in-process (form set) to a new user ID in the WIP data base, and writes a comment to the AFELOG file that it was assigned to a new user ID.

This procedure performs the same operation as the WIP, Assign option. This procedure only works with the Entry module and does not work with the data entry mode of Studio or Image Editor.

Example

Here is an example:

Procedure	Result	Explanation
AssignWIP (MVV)	User ID for the work-in-process is changed to *MVV*, the form set is saved in the WIP directory, and you return to the main menu.	The user ID in the WIP database is set to *MVV* for the current WIP.
AssignWIP ( )	The Assign window appears. Use this window to make the assignment.	If you omit the user ID, the system instead displays the Assign window, just as if you had selected the Formset, Assign Document option.

See also

WIP Functions on page 88

Documaker Workstation Administration Guide

Documaker Workstation User Guide

Avg

Use this function to return the decimal average of a group of fields which have names that begin with common characters. The result of the operation is returned.

Syntax

```
Avg (PartialName, Section, Form, Group)
```

Parameter	Description
PartialName	Enter a valid string. The string must be the common (prefix) portion of a set of field names that occur on the current section. The default is the current field.
Section	Enter the name of a section that contains the field you specified. The default is the current section.
Form	Enter the name of a form that contains the section and/or field you specified. The default is the current form.
Group	Enter the name of the form group that contains the form, section, and/or field you specified. The default is the current group.

The system and returns the average of the values of all fields that begin with the specified partial name.

Example

An example of field names that have a common start are:

Myfield1

Myfield2

Myfield20

Each of these fields is included if you specify the partial name using any of the leading characters of *myfield*. The first field will be excluded if you enter *myfield2*, but will match the other two field names.

The average is calculated by summing those fields that have values and dividing by the number of those fields included in the sum. Note that zero (0) is a valid field value. Fields which have never been given a value are excluded from the calculation.

NOTE: Include the PartialName parameter. Fields must have unique names within a section. Using the default will probably not give the expected result, unless you created the form and understand the naming conventions.

Example

This table is used by the examples. The table shows the layout of two forms in the same group. Both forms share two sections (IMG A and IMG B). Each section has fields of the same name as a field in the other section.

Field	Section	Form	Group	Value
MyField1	IMG A	FRM A	GRP	100.24
MyField2	IMG A	FRM A	GRP	200.16

Field	Section	Form	Group	Value
MyField1	IMG B	FRM A	GRP	98.60
MyField2	IMG B	FRM A	GRP	* no value yet *
MyField1	IMG A	FRM B	GRP	0.00
MyField2	IMG A	FRM B	GRP	* no value yet *
MyField1	IMG B	FRM B	GRP	70.77
MyField2	IMG B	FRM B	GRP	* no value yet *

Assume the current field is MyField1, on the first section of the first form. Reference the previous table for field values.

Function	Result	Explanation
Return(AVG ( ))	100.24	Without any other information, the function assumes the current field and section. There will only be one value included in the average.
Return(AVG (“Myfield2”))	200.16	Again, there is only one field included in this result.
Return(AVG(“MyField”))	150.20	In this example, the current section contains two fields that begin with the name “MyField”. The equation is as follows: (100.24 + 200.16) / 2
Return(AVG(“MyField”, “IMG B”))	98.60	Although two fields on IMG B have a matching name, only one field actually has a value.
Return(AVG(“MyField”, , “FRM A”))	133.00	No section is specified in this example, so the entire form is searched. Four fields match the name criteria, but only three have values: (100.24 + 200.16 + 98.60) / 3
Return(AVG(“MyField”, “IMG B”, , “GRP”))	84.685	This example specifies a section and group, but no form. There are four fields that match the name criteria, but only two have values: (98.60 + 70.77) / 2
Return(AVG(“MyField”, , , “GRP”))	93.954	This example names the group without a form or section. Eight fields meet the naming criteria, but only five fields actually have values: (100.24 + 200.16 + 98.60 + 0.00 + 70.77) / 5

See also

Mathematical Functions on page 72

Field Formats on page 62

Locating Fields on page 64

BankRound

Use this function to round numbers based on Banker's rounding. With Banker's rounding, values below 0.5 go down and values above 0.5 go up. Values of exactly 0.5 go to the nearest even number. In contrast, the Round function always rounds 0.5 upwards.

Note: When you add values which have been rounded using the standard method of always rounding .5 in the same direction, the result includes a bias that grows as you include more rounded numbers. Banker's rounding is designed to minimize this.

Syntax

```
BankRound(Value)
```

Parameter	Description
Value	Enter the value you want the system to round.

Example Here are some examples that compare BankRound with Round:

With BankRound		Whereas, with Round	
This	Returns	This	Returns
BankRound(123.425)	123.42	Round(123.425)	123.43
BanKRound(123.435)	123.44	Round(123.435)	123.44

See also String Functions on page 78

Round on page 365

BEEP

Use this procedure to tell the system to emit a warning, message, or error sound. The sound emitted depends on the installed options of the operating system that executes the system. There is no return value from this procedure.

Syntax

```
Beep (Integer)
```

Parameter	Description
Integer	Choose from these options: 0 - Warning sound 1 - Message sound 2 - Error sound The default is two (2).

This procedure emits the sound specified by the parameter.

Example

Here are some examples:

Procedure	Result	Explanation
Beep ( )	Emits error sound.	Defaults to 2.
Beep (0)	Emits warning sound.	The operating system emits the installed option for the warning sound.

See also Documaker Workstation Functions on page 59

BitAnd

Use this function to return the result of a bitwise AND operation performed on two numeric values.

Syntax

```
BitAnd (Value1, Value2)
```

The parameters specify the numeric values on which the bitwise AND operation is performed. If either parameter is not an integer, it will be converted to an integer before the bitwise AND operation is performed.

The bitwise AND operation compares each bit of value1 to the corresponding bit of value2. If both bits are 1, the corresponding result bit is set to 1. Otherwise, the corresponding result bit is set to zero (0). Note that integer values have 32 bits to compare.

The following table shows the result of a bitwise AND operation:

Value1 bit	Value2 bit	Result bit
0	0	0
0	1	0
1	1	1
1	0	0

Example

Here is an example:

```
x = 3   (3 is 0011 in binary)
y = 6   (6 is 0110 in binary)

z = BitAnd(x,y)
z = 2   (2 is 0010 in binary)
```

See also

BitClear on page 148

BitNot on page 149

BitOr on page 150

BitRotate on page 151

BitSet on page 153

BitShift on page 154

BitTest on page 156

BitXor on page 157

Bit/Binary Functions on page 42

BitClear

Use this function to return the result after clearing the specified bit in a value.

Syntax

```
BitClear(value1, bitpos)
```

The parameters specify the numeric value and the bit position on which the operation is performed. The specified bit is set to a zero (0) in the value provided. If the bit was not on, the value is unchanged. Specifying a negative or zero bit position does not result in any change to the value.

Note that integer values have 32 bits. When looking at the value in binary form, bit 1 is on the left and bit 32 is on the right.

```
Bit 32 -->0000 0000 0000 0000 0000 0000 0000 0000<-- Bit 1
```

Example

Here is an example:

```
y = 6  (6 is 0110 in binary)
z = BitClear(x,1)
z = 6  (6 is 0110 in binary) (bit 1 was already zero)

y = 6  (6 is 0110 in binary)
z = BitClear(x,2)
z = 4  (4 is 0100 in binary)
```

See also

BitAnd on page 147

BitNot on page 149

BitOr on page 150

BitRotate on page 151

BitSet on page 153

BitShift on page 154

BitTest on page 156

BitXor on page 157

Bit/Binary Functions on page 42

BitNot

Use this function to return the result of a bitwise logical NOT operation performed on a numeric value.

Syntax

```
BitNot(value1)
```

The parameter specifies the numeric value on which the bitwise logical NOT operation is performed. If the parameter is not an integer, it will be converted to an integer before the bitwise logical NOT operation is performed.

The bitwise logical NOT operation reverses the sense of the bits in the value. For each value bit that is 1, the corresponding result bit will be set to zero (0). For each value bit that is zero (0), the corresponding result bit will be set to 1.

It is especially important to note that integer values have 32 bits to compare when examining the results of a NOT operation. All bits of the integer will be altered by this operation.

The following table shows the result of a bitwise logical NOT operation:

Value1 bit	Result bit
0	1
1	0

Example

Here is an example:

```
x = 3  (3 is 0000 0000 0000 0000 0000 0000 0000 0011 in binary)

z = BitNot(x)
z = -4 (-4 is 1111 1111 1111 1111 1111 1111 1111 1100 in binary)
```

Notice that the NOT operation affects all bits of the integer.

See also

BitAnd on page 147

BitClear on page 148

BitOr on page 150

BitRotate on page 151

BitSet on page 153

BitShift on page 154

BitTest on page 156

BitXor on page 157

Bit/Binary Functions on page 42

BitOr

Use this function to return the result of a bitwise inclusive OR operation performed on two numeric values.

Syntax

```
BitOr(value1, value2)
```

Parameters specify the numeric values on which the bitwise OR operation is performed. If either parameter is not an integer, it will be converted to an integer before the bitwise OR operation is performed.

The bitwise inclusive OR operation compares each bit of value1 to the corresponding bit of value2. If either bit is 1, the corresponding result bit is set to 1. Otherwise, the corresponding result bit is set to zero (0). Note that integer values have 32 bits to compare.

The following table shows the result of a bitwise OR operation:

Value1 bit	Value2 bit	Result bit
0	0	0
0	1	1
1	1	1
1	0	1

Example

Here is an example:

```
x = 3   (3 is 0011 in binary)
y = 6   (6 is 0110 in binary)

z = BitOr(x,y)
z = 7   (7 is 0111 in binary)
```

See also

BitAnd on page 147

BitClear on page 148

BitNot on page 149

BitRotate on page 151

BitSet on page 153

BitShift on page 154

BitTest on page 156

BitXor on page 157

Bit/Binary Functions on page 42

BitRotate

Use this function to return the result of a bit shift-and-rotate operation performed on a numeric value.

Syntax

```
BitRotate(value1, shiftAmt)
```

The first parameter specifies the numeric value on which the bitwise shift-and-rotate operation is performed. The second parameter specifies the number of bit positions to shift. If either parameter is not an integer, it will be converted to an integer before the bitwise shift-and-rotate operation is performed.

This is a shift-and-rotate operation. This means that bits shifted off the end of a value are rotated back onto the value at the *other* end. In other words, the bits rotate in what might be thought of as a circular pattern — thus no bits are ever lost.

NOTE: See the BitShift on page 154 function for logical shift operations that do not shift-and-rotate.

A positive shiftAmt value causes the bit pattern in value1 to shift-and-rotate left the number of bits specified by shiftAmt. Bits that rotate off the left (high) end of the value return on the right (low) end.

A negative shiftAmt value causes the bit pattern in value1 to shift-and-rotate right the number of bits specified by shiftAmt. Bits that rotate off the right (low) end of the value return on the left (high) end. Note that integer values have 32 bits.

The following table shows the result of a bitwise shift-and-rotate operation:

Value1 bits	Shift	Result value bits
6 (0110)	1	12 (1100)
6 (0110)	2	24 (0001 1000)
6 (0110)	3	48 (0011 0000)
6 (0110)	4	96 (0110 0000)
6 (0110)	-1	3 (0011)
6 (0110)	-2	-2147483647 (1000 0000 0000 0000 0000 0000 0000 0001)
6 (0110)	-3	-1073741824 (1100 0000 0000 0000 0000 0000 0000 0000)
6 (0110)	-4	1610612736 (0110 0000 0000 0000 0000 0000 0000 0000)

Example

Here is an example:

```
z = BitRotate(6,-8)
z = 100663296 (0000 0110 0000 0000 0000 0000 0000 0000)
```

See also

BitAnd on page 147

BitClear on page 148

BitNot on page 149

BitOr on page 150

BitSet on page 153

BitShift on page 154

BitTest on page 156

BitXor on page 157

Bit/Binary Functions on page 42

BitSet

Use this function to return the result after setting the specified bit on in a value.

Syntax

```
BitSet(Value1, BitPos)
```

The parameters specify the numeric value and the bit position on which the operation is performed. The specified bit is set to a 1 in the value provided. If the bit was already on, the value is unchanged. Specifying a negative or zero bit position does not result in any change to the value.

Note that integer values have 32 bits. When looking at the value in binary form, bit 1 is on the left and bit 32 is on the right.

```
Bit 32 -->0000 0000 0000 0000 0000 0000 0000 0000<-- Bit 1
```

Example

Here is an example:

```
y = 6  (6 is 0110 in binary)
z = BitSet(x,1)
z = 7  (7 is 0111 in binary)

y = 6  (6 is 0110 in binary)
z = BitSet(x,4)
z = 15 (15 is 1110 in binary)
```

See also

BitAnd on page 147

BitClear on page 148

BitNot on page 149

BitOr on page 150

BitRotate on page 151

BitShift on page 154

BitTest on page 156

BitXor on page 157

Bit/Binary Functions on page 42

BitShift

Use this function to return the result of a bit logical shift operation performed on a numeric value.

Syntax

```
BitShift(Value1, ShiftAmt)
```

The first parameter specifies the numeric value on which the bitwise shift operation is performed. The second parameter specifies the number of bit positions to shift. If either parameter is not an integer, it will be converted to an integer before the bitwise shift operation is performed.

This is a logical shift, as opposed to a shift-and-rotate operation. This means bits shifted off the end of a value are considered lost.

NOTE: See the BitRotate on page 151 function for shift-and-rotate.

A positive shiftAmt value causes the bit pattern in value1 to be shifted left the number of bits specified by ShiftAmt. Bits vacated by the shift operation are zero-filled.

A negative shiftAmt value causes the bit pattern in value1 to be shifted right the number of bits specified by ShiftAmt. Bits vacated by the shift operation are zero-filled.

Note that integer values have 32 bits. Attempting to shift more than 31 bit positions will result in a zero (0) being returned, as all bits are cleared.

The following table shows the result of a bitwise SHIFT operation:

Value1 bits	Shift	Result value bits
6 (0110)	1	12 (1100)
6 (0110)	2	24 (0001 1000)
6 (0110)	3	48 (0011 0000)
6 (0110)	4	96 (0110 0000)
6 (0110)	-1	3 (0011)
6 (0110)	-2	1 (0001)
6 (0110)	-3	0 (0000)
6 (0110)	-4	0 (0000)

Example

Here is an example:

```
z = BitShift(6,8)
z = 1536  (1536 is 0110 0000 0000 in binary)
```

See also

BitAnd on page 147

BitClear on page 148

BitNot on page 149

BitOr on page 150

BitRotate on page 151

BitSet on page 153

BitTest on page 156

BitXor on page 157

Bit/Binary Functions on page 42

BitTest

Use this function to return TRUE (1) if the specified bit in a value is a 1; otherwise return FALSE (0).

Syntax

```
BitTest(Value1, BitPos)
```

Parameters specify the numeric value and the bit position on which the operation is performed. The specified bit is tested for a 1 value. If the bit is a 1, then 1 is returned. If the bit is zero (0), then zero (0) is returned. Specifying a negative or zero bit position will result in zero (0) being returned.

Note that integer values have 32 bits. When looking at the value in binary form, bit 1 is on the left and bit 32 is on the right.

```
Bit 32 -->0000 0000 0000 0000 0000 0000 0000 0000<-- Bit 1
```

Example

Here is an example:

```
y = 6  (6 is 0110 in binary)
z = BitTest(x,1)
z = 0  (bit 1 was not on)

y = 6  (6 is 0110 in binary)
z = BitTest(x,2)
z = 1  (bit 2 was on)
```

See also

BitAnd on page 147

BitClear on page 148

BitNot on page 149

BitOr on page 150

BitRotate on page 151

BitSet on page 153

BitShift on page 154

BitXor on page 157

Bit/Binary Functions on page 42

BitXor

Use this function to return the result of a bitwise exclusive OR operation performed on two numeric values.

Syntax

```
BitXor(Value1, Value2)
```

The parameters specify the numeric values on which the bitwise XOR operation is performed. If either parameter is not an integer, it will be converted to an integer before the bitwise XOR operation is performed.

The bitwise exclusive OR operation compares each bit of value1 to the corresponding bit of value2. If one bit is zero (0) and the other bit is 1, the corresponding result bit is set to 1. Otherwise, the corresponding result bit is set to zero (0). Note that integer values have 32 bits to compare.

The following table shows the result of a bitwise XOR (exclusive OR) operation:

Value1 bit	Value2 bit	Result bit
0	0	0
0	1	1
1	1	0
1	0	1

Example

Here is an example:

```
x = 3   (3 is 0011 in binary)
y = 6   (6 is 0110 in binary)

z = BitXor(x,y)
z = 5   (5 is 0101 in binary)
```

See also

BitAnd on page 147

BitClear on page 148

BitNot on page 149

BitOr on page 150

BitRotate on page 151

BitSet on page 153

BitShift on page 154

BitTest on page 156

Bit/Binary Functions on page 42

BreakBatch

Use this function to tell the Documaker Server to break the output print stream file for the current recipient batch after processing the current recipient, including post transaction banner processing.

Syntax

```
BreakBatch()
```

This procedure is typically called in the transaction banner DAL script. You must use the SetDeviceName function to specify a new device name. Otherwise, the new file has the same name as the old file and overwrites its contents.

After the GenPrint program finishes processing the current transaction, it closes the current output device file. This includes executing any post-batch banner processes. It then continues processing the recipient batch.

If you have assigned a new output device file name using the SetDeviceName function, the system will create and start writing to a new print stream file with that name. The best place to call the BreakBatch function is in the post-transaction banner DAL script.

Here is an example of DAL script logic that might appear in a post-transaction banner DAL script. This example requires that a pre-transaction banner DAL script save the current recipient name in a variable called *CurrRecip,* as shown here:

```
CurrRecip = RecipName()
```

The post-transaction banner DAL script would then include the following:

```
IF TotalSheets(CurrRecip) > 16000
#COUNTER += 1
CurFile = DeviceName()
Drive = FileDrive(CurFile)
Path = FilePath(CurFile)
Ext = FileExt(CurFile)
RecipBatch = RecipBatch()
NewFile = FullFileName(Drive,Path,RecipBatch & #COUNTER,Ext)
SetDeviceName(NewFile)
BreakBatch()
END
```

NOTE: See FileDrive, FileExt, FileName, FilePath, and FullFileName for information on using DAL functions to manipulate file names.

Keep in mind...

- These print drivers are supported: AFP, MET, PCL5, PCL6, and PST.
- These print drivers *are not* supported: EPT, GDI, HTML, PDF, RTF, and XML.
- All platforms are supported, but note that while UniqueString is supported on z/OS, z/OS does not support long file names.
- Both multi-step and single-step processing are supported.

The only DAL function actually involved in splitting the print stream is BreakBatch. The others make it easier to implement this functionality. For example, since you need to name the new print stream, you use the SetDeviceName procedure. To find the name of the current device, you use the DeviceName function. If you need to create unique file names, you can use the UniqueString function.

NOTE: The BreakBatch and SetDeviceName functions are not applicable in Entry since it does not use the batch printing engine. The other functions, DeviceName and UniqueString, are applicable to both Entry and Documaker Server.

See also

Printer and Recipient Functions on page 76

DeviceName on page 217

SetDeviceName on page 373

UniqueString on page 421

CALL

Use this function to temporarily suspend one calculation and execute another calculation file. A CALL statement must begin with *CALL*.

Syntax

```
CALL (File)
```

Parameter	Description
File	Enter the name of the calculation file you want the system to execute.

The calculation file that is called must contain a RETURN statement if the original calculation expects a returned value.

Example

Here is an example:

```
CALL( 'TestCalc' )
```

This tells the system to call the calculation file TestCalc. After the calculations in TestCalc are completed, processing returns to the current script. In this example, TestCalc is not expected to return a value.

See also

Miscellaneous Functions on page 73

CHAIN

Use this function to calls another calculation language file. A Chain statement must begin with *CHAIN*. There is no limit to the number of Chain statements you can use.

Syntax

```
CHAIN (Script)
```

Parameter	Description
Script	Enter the name of the DAL script file. You can omit the extension.

Example

Here are some examples:

```
CHAIN 'LastCalc'
```

or

```
CHAIN( 'LastCalc' )
```

These examples permanently call the calculation file named LastCalc. Processing does not return to the current script. No statements from the original script will be evaluated after the Chain statement.

See also

Miscellaneous Functions on page 73

CFind

Use this function to search a text string and return the first position of any character found within a specified set of characters. The search is not case sensitive.

Syntax

```
CFind (String, Charset, Integer)
```

Parameter	Description
String	Enter a valid string. This is the string that is searched. The default is the value of the current field text.
Charset	Enter a set of one or more characters, any of which may be found in the target string.
Integer	Enter zero (0) for a left to right search. Enter one (1) for a right to left search. The default is zero (0).

The system returns a zero (0) if none of the search characters are found in the text string.

The default search order is *left to right*. You can also specify a right to left search order. Both search methods returns the position relative to the first (left-hand) character of the string parameter.

Example

Here are some examples:

(Assume the current field contains the text *Your Name*.)

Function	Result	Explanation
Return(CFind (“This is the answer”, “ws”))	4	Searching from left to right, *s* was first found at position 4.
Return(CFind (“This is the answer”, “ws”, 1))	16	Searching from right to left, *w* was first found at position 16.
Return(CFind (, “n”))	6	The first occurrence of an *n* in the current field *Your Name* is at position 6. Note the search is not case sensitive.
Return(CFind(, “xz”))	0	Neither *x* nor *z* is contained in the current text field.

See also String Functions on page 78

CHANGELOGO

Use this procedure/function to replace a bitmap graphic (LOG file) on a section with a different graphic.

Syntax

```
ChangeLogo (LOGFile, Graphic, Section, Form, Group)
```

Parameter	Description
LOGFile	Enter the name of a file that contains a valid graphic.
Graphic	Enter the name of the current graphic in a section.
Section	Enter the name of a section that contains the graphic you specified. The default is the current section.
Form	Enter the name of a form that contains the section or graphic. The default is the current form.
Group	Enter the name of a group to use to locate the object. The default is the current group.

The system optionally returns a one (1) if successful and zero (0) if unsuccessful.

This procedure expects to locate the named graphic in the same way and location used to load any other graphic. You must include the Graphic parameter.

If you omit the LOGFile parameter or the graphic cannot be loaded, the system will insert an empty graphic. A placeholder appears during entry to indicate the graphic position, however, nothing will print if a graphic is not loaded. This procedure lets you remove a signature from a form if necessary.

The Graphic parameter tells the system to look for the name that appears in the Name field on the Graphic Options in Studio or Image Editor. If there is no entry in this field, this procedure will not work correctly.

NOTE: When you use this procedure with Documaker Workstation, you must follow this procedure with the Refresh procedure. The ChangeLogo procedure does not redraw the section after it changes the graphic.

When you use the ChangeLogo procedure with Documaker, you must include the CheckImageLoaded rule as one of the section level rules for the section or else set the LoadCordFAP option in the RunMode control group to Yes in your FSISYS.INI file.

Example

Here are some examples:

(Assume the section has a graphic named *sign*.)

Procedure	Result	Explanation
ChangeLogo (“johndoe”, “sign”)	1 or 0	Replaces the existing graphic contained by sign with a new graphic (johndoe). The existing graphic is assumed to exist in the current section.
ChangeLogo (, “sign”, “IMG”)	1 or 0	Locate the specified section on the current form. If found replace the existing graphic contained by sign with an empty graphic.

See also

DelLogo on page 214

HaveLogo on page 272

InlineLogo on page 282

RenameLogo on page 359

Logo on page 303

Refresh on page 357

Graphics Functions on page 71

CHAR

Use this function to convert an integer into a single character.

Syntax

```
Char (Integer)
```

Parameter	Description
Integer	An integer value that ranges zero (0) to 255.

Example

Here is an example:

```
what_char = Char (64)
```

The variable, *what_char*, is set to the character: ‘@’.

See also

CharV on page 166

String Functions on page 78

CharV

Use this function to convert a single character into an integer value.

Syntax

```
CharV (String)
```

Parameter	Description
String	A character string. If the string contains more than one character, only the first character is converted. The remaining characters are ignored.

Example

In this example, assume the variable, char_to_convert, contains the single character: “@”.

```
#_the_integer = CharV(char_to_convert)
```

The integer variable, *#_the_integer*, is set the value: *64*.

In this example, assume the variable, *the_string*, contains the characters: “@( )”.

```
#_the_integer = CharV(the_string)
```

The integer variable, *#_the_integer*, is set the value: *64*. The remaining characters are ignored.

See also

Char on page 165

String Functions on page 78

CODEINLIST

Use this function to search for a string in a list of a strings.

Syntax

```
CodeInList (String,List)
```

Parameter	Description
String	Enter the string you want to search for. Keep in mind the system considers spaces when matching strings and that the strings *must* match exactly.
List	Enter the name of the list of strings. Use commas to separate each string entry you want to search for.

The function returns a number that indicates which string entry was found. For instance, if the third string entry was found, the function returns a three (3).

Example

Here is an example:

```
CodeInList( "ABC",  "ABC,AB,DE,A,GFHI,ABCD" )returns 1
CodeInList( "AB",   "ABC,AB,DE,A,GFHI,ABCD" )returns 2
CodeInList( "DE",   "ABC,AB,DE,A,GFHI,ABCD" )returns 3
CodeInList( "A",    "ABC,AB,DE,A,GFHI,ABCD" )returns 4
CodeInList( "GFHI", "ABC,AB,DE,A,GFHI,ABCD" )returns 5
CodeInList( "ABCD", "ABC,AB,DE,A,GFHI,ABCD" )returns 6
CodeInList( "XYZ",  "ABC,AB,DE,A,GFHI,ABCD" )returns 0
CodeInList( "",     "ABC,AB,DE,A,GFHI,ABCD" )returns 0
CodeInList( "ABC",  "" )                     returns 0
CodeInList( "",     "" )                     returns 1
```

If you omit the first parameter, you get the data from the current field. If you omit the second parameter, you receive this error message:

```
Wrong number of parameters
```

Here is another example:

Assume that GetValue(col_name1) results in the string: EE. And the variable col_name1_codes contains the string: EEacb,XXEE,EE,AEEAC.

```
#rc = CodeInList( GetValue(col_name1), col_name1_codes ) returns 3
```

Keep in mind...

- The search *is not* case sensitive. This means that *A* will match *a*.
- Spaces *are* considered. This means the system will find no matches in these examples:

```
CodeInList("Steel", " Steel,Aluminum")
CodeInList("Steel", "Steel ,Aluminum")
CodeInList("Steel", "Aluminum,Steel ")
```

and will return zero (0) each time.

See also

String Functions on page 78

COMPLETE

Use this procedure/function to complete the work-in-process.

Syntax

```
Complete (PrintFlag, ExportFlag, ExportType, ExportFile)
```

Parameter	Description
PrintFlag	Indicates whether the system should print the form set. The default is No.
ExportFlag	Indicates whether the system should export the work-in-process data to a file. The default is No.
ExportType	TD, SI, and so on. Indicates the type of export file. The default is TD.
Exportfile	The file name for the Standard Export file, if specified in the INI options.

This procedure performs the same processes as the File, Complete option except the windows which request information from the user do not appear if you enter all values. This procedure starts the following processes, as specified by INI options:

- Prints (immediate or batch) the form set
- Archives the form set
- Exports work-in-process data to a file

The standard export format is the only file format supported. This procedure returns success (1) if no error occurred during the complete process. If an error occurred, the procedure returns a zero (0).

Example

Here is an example:

Procedure	Result	Explanation
Complete ( )	Completes the work-in-process.	Performs the processes as specified by archive INI options.
Complete,,, (EXPORT.TXT)	Completes the work-in-process and writes the data to a file named EXPORT.TXT.	Performs the processes as specified by archive INI options and writes the data to a file named EXPORT.TXT.

See also

WIP Functions on page 88

Documaker Workstation Administration Guide

Documaker Workstation User Guide

CompressFlds

Use this function/procedure to compress blank space by moving field data. This function moves field data from one field to a prior named field to compress the space between the fields. Typically you use this function to compress vertical space, as in address lines, but the fields do not have to be vertical relative to each other. You can compress any field.

NOTE: The data moves between the fields; the actual location of each physical field remains the same.

CompressFlds can be used as a procedure or as a function.

Syntax

```
CompressFlds (FieldList, Section, Form, Group)
```

Parameter	Description
FieldList	Enter a list of the fields you want to compress, separated by commas. Here is an example: `"FIELD1, FIELD2, FIELD3"`
Section	(Optional) Enter the name of a section that contains the fields you specified. The default is the current section.
Form	(Optional) Enter the name of a form that contains the section and/or field you specified. The default is the current form.
Group	(Optional) Enter the name of the form group that contains the form, section, or fields. The default is the current group.

NOTE: When using this function in Documaker Server processing, make sure the fields exist on the section. Some implementations that use versions of the system prior to version 11.0 do not load FAP files in all cases, and fields will not be created when data mapping did not place any data into the field.

Keep in mind...

- Each subsequent field with data is mapped into the first available empty field which you included in the list.
- Fields are defined in FAP sections with a tabbing order. This tabbing order typically matches the order in which field level rules are processed during Documaker Server processing. Unlike the SetAddr rules, the CompressFlds function can compress fields in any order, and the field spaces do not have to be *compressed up* following the tabbing order.
- The last movement of that field determines the final location of a given field's data.
- Always specify a set of unique field names. Do not attempt to name a field more than once within a field list as this can produce unpredictable results.
- This function does not work with bar code or multi-line text fields.

Example

For this example, assume the following fields and data:

This field	Contains
FIELD_A	*ABCDEFG*
FIELD_B	is empty
FIELD_C	is empty
FIELD_D	*TUVWXYZ*

Assume your field list looks like this:

```
"FIELD_A, FIELD_B, FIELD_C, FIELD_D"
```

FIELD_A does not move because there is no field named before it.

FIELD_B and FIELD_C are empty; therefore, the data from FIELD_D moves into FIELD_B, which is the first available empty field.

The result looks like this:

This field	Contains
FIELD_A	*ABCDEFG*
FIELD_B	*TUVWXYZ*
FIELD_C	is empty
FIELD_D	is empty

If you had specified the field list parameter had been specified like this:

```
"FIELD_D, FIELD_C, FIELD_B, FIELD_A"
```

The result would be as follows:

This field	Contains
FIELD_A	is empty
FIELD_B	is empty
FIELD_C	*ABCDEFG*
FIELD_D	*TUVWXYZ*

See also Field Functions on page 61

CONNECTFLDS

Use this function/procedure to move fields (change field coordinates) in such a way as to make the field's text appear to be concatenated. This function does not literally concatenate the fields but instead repositions and aligns field text along a common horizontal coordinate so the field's data appears concatenated. It does not move fields vertically.

This function automatically loads the section — either the FAP file or the compiled version of the FAP file — if the section has not already been loaded. FAP files must be loaded to provide some of the information required to perform the operation.

Syntax

```
ConnectFlds (FieldList, Section, Form, Group)
```

Parameter	Description
FieldList	A list of the fields you want to connect, preceded by a movement flag and separated with commas. Here is an example: `"FIELD1, FIELD2, FIELD3"` If a field name is not preceded by a movement flag or if it is preceded by the *F* movement flag, which indicates it is a fixed field, the field is not moved. The first field you name in the parameter must be a fixed field. The rest of the field names in your list indicate fields you want moved adjacent to the fixed field. Each field you name is moved according to the use described by the movement flag that precedes its name.
Section	(Optional) Enter the name of a section that contains the fields you specified. The default is the current section.
Form	(Optional) Enter the name of a form that contains the section and/or field you specified. The default is the current form.
Group	(Optional) Enter the name of the form group that contains the form, section, or fields. The default is the current group.

In the FieldList parameter you must specify a fixed field and at least one field to move (visually concatenate) to the left or right side of the fixed field. You can specify multiple fields to move.

NOTE: This function does not move fields vertically. Fields are only moved horizontally. You should set the vertical alignment of fields when you create the section.

By default, each concatenation will be placed the distance of one space character from the fixed field, unless the parameter indicates otherwise. You can include these movement flags in the FieldLIst parameter:

Flag	Description
L	Tells the system to move the specified field so it appears to be appended to the left of the fixed field.
R	Tells the system to append the specified field to the right of the fixed field.

Flag	Description
NO	Tells the system you want no spacing between the two fields.

Here is an example:

```
"F=FIELD1,RNO=FIELD2"
```

Here, the contents of FIELD2 are placed immediately adjacent to the end of the contents of FIELD1 without an intervening space.

Keep in mind...

- As each field is appended to the fixed field, the fixed rectangle grows. By growing the fixed rectangle, additional fields that append move based upon where the prior appended field ended.
- If a field specified for appending does not contain any data or is not valid, then no space, or space holder, is included in the concatenation.
- If a field contains centered or right justified data padded with spaces then the results may appear to be incorrect. This function calculates the width of a field based upon the entire contents and will not remove spaces, or any other white space characters, in the fields.
- Naming a field to move more than once in the first parameter can cause unpredictable results.
- The last movement of a field will determine the final location of a field's data.
- During any movement operation, the field being moved cannot also be named as the fixed field.
- This function does not work with bar code or multi-line text fields.
- This function does not handle rotated fields.

Example

For the following examples, make these assumptions:

This field	Contains
FIELD1	ABC
FIELD2	DEF
FIELD3	XYZ

If you enter:

```
ConnectFlds("F=FIELD1,R=FIELD2")
```

You get this result:

```
ABC DEF
```

If you enter:

```
ConnectFlds("F=FIELD1,L=FIELD2,R=FIELD3")
```

You get this result:

```
DEF ABC XYZ
```

This example appended FIELD2 to the left side of FIELD1 and appended FIELD3 to the right side of FIELD1. The fixed field, FIELD1, did not move. FIELD2 and FIELD3 moved to align with FIELD1. During this operation, FIELD1 never moved.

If you enter:

```
ConnectFlds("FIELD1,LNO=FIELD2,RNO=FIELD3")
```

You get this result:

```
DEFABCXYZ
```

This example is similar to the prior example but uses the NO parameter.

If you enter:

```
ConnectFlds("F=FIELD1,R=FIELD2,R=FIELD3")
```

You get this result:

```
ABC DEF XYZ
```

In this example, two fields are appended to the right of the fixed field. The first appended field expanded the rectangle, which allows the next one to append after the last.

If you enter:

```
ConnectFlds("F=FIELD1,R=FIELD2,F=FIELD2,R=FIELD3")
```

You get this result:

```
ABC DEF XYZ
```

Notice that the result of this example is the same as the previous example. In this case, the fixed field was changed to FIELD2 after FIELD2 had moved adjacent to FIELD1. Then FIELD3 was moved adjacent to FIELD2 in its new location.

If you enter:

```
ConnectFlds("F=FIELD1,R=FIELD2,R=FIELD2")
```

You get this result:

```
ABC     DEF
```

In this case, FIELD2 is defined to move twice. Since the operations are sequential, the field first moved adjacent to FIELD1. This movement expanded the fixed rectangle used by subsequent movements. When the field was named again, it moved relative to the newly expanded rectangle, resulting in the field appearing farther to the right, a distance equal to the size of the text in the field plus the width of two spaces.

See also Field Functions on page 61

CopyForm

Use this procedure/function to locate a form and copy that form and its field contents (data) into a new form. With this procedure, you can also specify another form and group as the insertion point for the new form.

NOTE: When you use the AddForm procedure, the only data duplicated is the global data that propagates into the fields. When you use the DupForm procedure, only those forms with the Multicopy option checked can be duplicated. With the CopyForm procedure, any form within the document can be copied.

Syntax

```
CopyForm (Form, Group, InsAtForm, InsAtGroup)
```

Parameter	Description
Form	Enter the name of the form you want to copy
Group	(Optional) Enter the name of the group if the form is not in the current group.
InsAtForm	Enter the name of the form after which you want the system to insert the form it copies.
InsAtGroup	(Optional) Enter the name of the group for the insertion point form, specified in the InsAtForm parameter if that form is not in the current group.

If you do not specify an insertion point, the system appends the new form to the end of the form group of the original form.

If the procedure is successful in copying the form, it returns a non-zero value, otherwise zero (0) is returned. This procedure can fail for these reasons:

- Could not locate the form or form group specified
- Lack of available memory

You can use this procedure in scripts hosted by AFEMain or other Entry-related applications and also in batch applications using the GenData program.

See also

AddForm on page 119

AddForm_Propagate on page 120

DupForm on page 228

TriggerForm on page 414

WIP Functions on page 88

COUNT

Use this function to count the number of fields that have values and have names that begin with common characters. The result of the operation is returned.

Syntax

```
Count (PartialField, Section, Form, Group)
```

Parameter	Description
PartialField	Enter a valid string. The string must be the common (prefix) portion of a set of field names that occur on the current section. The default is the current field.
Section	Enter the name of a section that contains the field you specified. The default is the current section.
Form	Enter the name of a form that contains the section and/or field you specified. The default is the current form.
Group	Enter the name of the form group that contains the form, section, and/or field you specified. The default is the current group.

This function returns the number of fields that have values that begin with the specified partial field name.

An example of field names that have a common start are:

Myfield1
Myfield2
Myfield20

Each of these fields will be included if the partial field name is using any of the leading characters of *myfield*. The first field will be excluded if you enter *myfield2*, but will match the other two field names.

Note that zero (0) is a valid field value. A field that has never been given a value is excluded from the count.

NOTE: As a general rule, include the PartialField parameter. Fields in a section must have unique names. Using the default will probably not give the expected result, unless you created the form and understand the naming conventions.

Example

Here are some examples:

The following table will be used by the examples. The table represents the layout of two forms in the same group. Both forms share two sections (IMG A and IMG B). Each section has fields of the same name as a field in the other section.

Field	Section	Form	Group	Value
MyField1	IMG A	FRM A	GRP	100.24
MyField2	IMG A	FRM A	GRP	200.16

Field	Section	Form	Group	Value
MyField1	IMG B	FRM A	GRP	98.60
MyField2	IMG B	FRM A	GRP	* no value yet *
MyField1	IMG A	FRM B	GRP	0.00
MyField2	IMG A	FRM B	GRP	* no value yet *
MyField1	IMG B	FRM B	GRP	70.77
MyField2	IMG B	FRM B	GRP	* no value yet *

(Assume the current field is MyField1, on the first section of the first form. Reference the previous table for field values.)

Function	Result	Explanation
Count( )	1	Without any other information, the function will assume the current field and section. There will only be one value included in the count.
Return(Count ("Myfield2"))	1	Again, there is only one field included in this result.
Return(Count ("MyField"))	2	In this example, the current section contains two fields that begin with the name "MyField".
Return(Count ("MyField", "IMG B"))	1	Although two fields on IMG B have a matching name, only one field actually has a value.
Return(Count ("MyField", , "FRM A"))	3	No section is specified in this example, so the entire form is searched. Four fields match the name criteria, but only three have values.
Return(Count ("MyField", "IMG B", , "GRP"))	2	This example specifies a section and group, but no form. There are four fields that match the name criteria, but only two have values.
Return(Count ("MyField", , , "GRP"))	5	This example names the group without a form or section. Eight fields meet the naming criteria, but only five fields actually have values.

See also

Mathematical Functions on page 72

Field Formats on page 62

Locating Fields on page 64

COUNTREC

Use this function to count the number of records in an extract file transaction that match a search mask parameter. In addition, you can also make sure that at least a minimum number of records match the search mask parameter.

Syntax

```
CountRec (SearchMask, MinNumber)
```

Parameter	Description
SearchMask	The search mask you want to use for the search.
MinNumber	(Optional) Number of records that must exist in the transaction. Set this parameter to 1 if you want to know if a record exists that matches the search mask.

This function returns the total number of records found, the MinNumber of records if they exist, or zero (0) if no records match the search mask or there are less than the MinNumber of records.

Example

Lets assume there are five records in a transaction with the following values in the applicable columns.

```
0               3
1               1

Address1        AA
Address2        BB
Address3        BB
Address4        BB
Address5        CC
```

Function	Result	Explanation
CountRec (“1,Address”)	5	The function returns five (5) because there are five records that match the search mask in the transaction.
CountRec (“1,Address,31,BB”, 2)	2	The function returns two (2) because there are at least two records that match the search mask in the transaction.
CountRec (“1,Address”, 6)	0	The function returns zero (0) because there are less than six records in the transaction that match the search mask.
CountRec ("1,Address,31,AA", 2)	0	The function returns a zero (0) because there are less than two records that match the search mask.

See also

Documaker Server Functions on page 58

Field Formats on page 62

Locating Fields on page 64

Cut

Use this function to remove characters from a string at a specified position and return the result.

Syntax

```
Cut (String, Position, Length)
```

Parameter	Description
String	Enter a valid string. The default is the value of the current field text.
Position	Enter the position within the first parameter to begin cutting. The default is one (1).
Length	Enter the length to cut from text. The default is zero (0).

This function returns a string equivalent to parameter 1 with the portion identified by the position and length parameters removed. If no position is given, or it is zero (0), the cut starts at position 1 in the string.

If no length is given, or it is zero (0), nothing is removed from the string and the return value is the same as the original string parameter.

Example

Here are some examples:

(Assume the current field contains the text *Your Name*.)

Function	Result	Explanation
Return(Cut ( ))	Your Name	No length is specified for the cut function; therefore the field remains the same.
Return(Cut (, , 5))	Name	Five characters are cut from the current field beginning at position 1.
Return(Cut ("Complete all the blanks.", 10, 4))	Complete the blanks	Goes to position 10 to begin the cut and removes four characters.
Return(Cut ("Complete all the blanks.", ,9))	all the blanks	Defaults to position 1 to begin the cut and cuts nine characters.

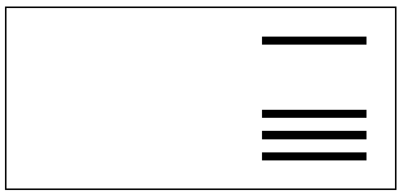
See also

String Functions on page 78

DASHCODE

Use this function to build a value to assign to a series of fields from the binary value of an integer. This is sometimes called a *dash code*. A dash code is a type of OMR mark that is read by certain mail, binding, or inserting equipment.

A dash code is a series of horizontal lines aligned in a column — each usually around 1/2 to one inch in length — that are typically on the left or right edge of the paper. The marks are usually expected to be in a uniform (fixed) position. Here is an example of a dash code:



Dash codes can be used, for instance, to represent the beginning or end of a set of pages that are associated in some way. The marks might indicate sequencing, first page, last page, staple requirements, additional pages to be inserted at a given point, the envelope size, or binding requirements.

NOTE: The exact meaning, order, and position of each mark depends on the finishing equipment you are using. Check the specifications that came with your equipment and assign the values appropriately.

Syntax

```
DashCode(Value, Bits, RootName, Section, Form, Group, OnString,
OffString, Direction, AltLens)
```

Parameter	Description
Value	Each bit of the value parameter is tested for a one (1) or zero (0). If the bit is one (1), it is considered on and the character you specify in the OnString parameter is appended to the string result being built. If the bit is zero (0), the OffString parameter is appended to the string result.
Bits	This parameter identifies how many of the bits from the value need to be evaluated. By default all 32 bits are evaluated. If you specify a negative or zero value, you'll get an empty string.
RootName	This parameter identifies the initial portion of a series of field names that are to be the repository for the OnString and OffString filled values. The bit number referenced will be appended to each name to form the final name expected to be found on the resulting section. For instance, if *MVALUE_* is passed as the RootName, the first fill value is assigned to *MVALUE_1*, the second to *MVALUE_2*, *MVALUE_3*, and so on, until the maximum number of bits specified are all mapped. If all 32 bits are mapped, the last field would be *MVALUE_32*. The associated fields will be filled to their defined length. In most dash code (bar code) type situations, you will want all the fields to be the same length.

Parameter	Description
Section	Enter the name of a section that contains the field you specified. You can enter an asterisk (*) to tell the function to search all sections. Keep in mind, however, that including an asterisk (*) degrades performance.
Form	Enter the name of a form that contains the section and/or field you specified. You can enter an asterisk (*) to tell the function to search all forms. Keep in mind, however, that including an asterisk (*) degrades performance.
Group	Enter the name of the form group that contains the form, section, or field. You can enter an asterisk (*) to tell the function to search all groups. Keep in mind, however, that including an asterisk (*) degrades performance.
OnString	By default, OnString is an underscore (_). You can specify alternative OnString and OffString values and each can be more than one character. The two parameters do not have to be the same length. If you define multiple characters, the fill value will repeat those characters as necessary to fill the entire field. If the field length is not evenly divisible by the length of the string you enter, a partial copy of the string can appear at the end. For instance, suppose the field length is five; OnString is *ABC*; and OffString is *XY*. If the bit value for this field is one (1), the fill value generated will be: *ABCAB*. If the bit value is zero (0), the fill value generated for this field will be *XYXYX*.
OffString	By default, Offstring is a space ( ). You can specify alternative OnString and OffString values and each can be more than one character. The two parameters do not have to be the same length. If you define multiple characters, the fill value will repeat those characters as necessary to fill the entire field. If the field length is not evenly divisible by the length of the string you enter, a partial copy of the string can appear at the end.

Direction	Note that integer values have 32 bits. When looking at the value in binary form, bit 1 is on the right and bit 32 is on the left. To override the default behavior, you can supply a non-zero Direction parameter.
	`0000 0000 0000 0000 0000 0000 0000 0000`
	`Bit 32 |                              | Bit 1`

AltLens	The final parameter is a comma-delimited pattern string to identify alternate lengths for each field associated with the bits. By default, each field is assigned a value equal to its defined length. If you want to use a different length, supply the appropriate lengths in string form separated by commas. The order of the length values starts with the field associated with the first bit, followed by the length for the second field, and so on. Remember the *first bit* is determined by the direction parameter. If you do not provide enough length values to match the number of bits you are using, the undefined positions will default to the default field length.

The return value indicates the number of fields assigned. A return value of zero (0) means that no fields were found.

Example

Here are some examples:

#val = 11 (which is 1011 in binary)

```
DASHCODE(#val, 4, "BFLD");
```

Assuming that *BFLD* is a root field name and matching fields are located on the current section, the following assignments are made. Further assume that each field is five characters in length.

BFLD1 is assigned "_____"
BFLD2 is assigned "_____"
BFLD3 is assigned "   " (five spaces)
BFLD4 is assigned "_____"

```
DASHCODE(#val, 4, "BFLD", , , , "A", "B");
```

This example uses the parameters to supply different OnString and OffString parameters.

BFLD1 is assigned "AAAA"
BFLD2 is assigned "AAAA"
BFLD3 is assigned "BBBB"
BFLD4 is assigned "AAAA"

```
DASHCODE(#val, 4, "BFLD", , , , "A","B",1);
```

Note the Direction parameter was used to reverse the order of the bits interpretation.

BFLD1 is assigned "AAAA"
BFLD2 is assigned "BBBB"
BFLD3 is assigned "AAAA"
BFLD4 is assigned "AAAA"

```
DASHCODE(#val, 4, "AB", "XYZ", 0, "1,2,3,5");
```

In this example, the last parameter applies differing lengths to the fields you are mapping. This example also uses alternate OnString and OffString parameters and uses text greater than one character. In this case, the string may be truncated or repeated as necessary to fill the field length.

BFLD1 is assigned “A”
BFLD2 is assigned “AB”
BFLD3 is assigned “XYZ”
BFLD4 is assigned “ABAB” or “ABABA”

Note that the last example indicates two possible results. During Documaker Workstation entry, the field length is considered paramount and cannot be overridden. During batch operations, it is possible for the data length to override the field length.

See also

Bit/Binary Functions on page 42

Date

Use this function to build a date from a given date, or from the current date.

Syntax

```
Date (Format, Day, Month, Year)
```

Parameter	Description
Format	Enter a date format. The default is format 1 (MM/DD?YY).
Day	Enter an integer day value. The default is based on the current day.
Month	Enter an integer month value. The default is based on the current month.
Year	Enter an integer year value. The default is based on the current year.

The system returns a date string that contains a formatted date value. If you omit any of the Day, Month, or Year parameters, the system uses a value based on the current date.

NOTE: To change to some date formats, make sure the variable field's Type field (on the field's Properties window) is set to alphanumeric.

Example

Here are some examples:

(Assume the current date is 07/01/10.)

Function	Result	Explanation
Return(Date( ))	07/01/2010	No parameters entered, defaults to current date in date format 1.
Return(Date("44 "))	July 1, 2010	Date format 4 selected, with a four-digit year length. Defaults to the current date in the selected format.
Return(Date (,18,5,2009))	05/18/2010	Defaults to date format 1 using the given values.
Return(Date("I2" ,18,5))	10/138	Date format I selected with a two-digit year length. Enters the given date values in the selected format.

See also

Date Functions on page 51

Date Formats on page 52

Using INI Options on page 8

DATE2DATE

Use this function to convert a date from one format to a new format.

Syntax

```
Date2Date (Date, Format, NewFormat)
```

Parameter	Description
Date	Enter a date string. The system assumes your entry to be in the format specified in the Format parameter. The default is the current date.
Format	Enter a date format string that describes the contents of the Date parameter. The default is date format 1 (MM/DD/YY).
NewFormat	Enter the date format you want to convert to. The default is date format 1.

This function converts a date string from one format to another. The new value is formatted according to the *NewFormat* parameter.

Example

Here are some examples:

(Assume the current date is 07/01/09 and the variable field called, *arc_date*, contains the hexadecimal value, *BC6792D0*)

Function	Result	Explanation
Return(Date2Date( ))	07/01/2009	No parameters entered—defaults to current date in date format 1.
Return(Date2Date("02/01/09", "1", "44"))	February 1, 2009	Changes the given date (02/01/09) from date format "1" to date format "4", with a four-digit year.
Return(Date2Date("09/138", "G"))	05/18/09	Changes the given date (09/138) from date format G to the default date format 1.
Return(Date2Date (, , "X"));	BB273650	Returns the current date in a eight character hexadecimal representation.
Return(Date2Date (@ ("arc date"), "X", "4"));	February 29, 2008	Converts the hexadecimal date to month name DD, YYYY without leading zeros.

See also

Date Functions on page 51

Date Formats on page 52

Using INI Options on page 8

DATEADD

Use this function to add a specified number of days, months, and/or years to a date.

Syntax

```
DateAdd (Date, Format, Days, Months, Years)
```

Parameter	Description
Date	Enter a date string. The system assumes your entry to be in the format specified in the Format parameter. The default is the current date.
Format	Enter a date format string that describes the contents of the Date parameter. The default is date format 1 (MM/DD/YY).
Days	Enter the number of days. The default is zero (0).
Months	Enter the number of months. The default is zero (0).
Years	Enter the number of years. The default is zero (0).

This function adds a specified number of days, months, and years to a given date. The result is formatted according to the Format parameter.

The *Days*, *Months*, and *Years* parameters can be negative or positive. If you enter a negative parameter, the system subtracts the specified days, months, or years.

You do not have to divide the values into components. For example, you can add 300 days and 40 months to a date. The result reflects the appropriate year, month, and day.

NOTE: This function tells the system to add days, months, and years—*in that order*. For instance, if you tell the system to add one day and one year to the date 02/28/2007, the result is 03/01/2008—not 02/29/2008.

To get 02/29/2008 as the result, you would use two calculations, first adding the year, then adding the day.

Example

Here are some examples (assume the current date is 07/01/09):

Function	Result	Explanation
Return(DateAdd (Date( ), "1", 10))	07/11/2009	Defaults to the current date which is specified as Date( ) and adds 10 days.
Return(DateAdd (02/01/09, , , 44))	10/01/2012	Uses the given date (02/01/09) and adds 44 months. (Note that if you enter "44" as a string, it is automatically converted to an integer.)
Return(DateAdd ("09/138", "I" , , , -3))	06/139	The given date (09/138) using date format *I* is May 18, 2009. Subtracting three years results in the date May 18, 2006. Because 2008 is a leap year, the correct day of the year (counting consecutively from January 1) is 139. The resulting date is returned in the same date format.

See also Date Functions on page 51

Date Formats on page 52

DateCnv

Use this function to convert two-digit years into four-digit years.

Syntax

```
DateCnv (Date, Format, DivideYear, Century)
```

Parameter	Description
Date	Enter a date string. The system assumes your entry to be in the format specified in the Format parameter. The default is the current date.
Format	Enter a date format string that describes the contents of the Date parameter. The default is date format 1 (MM/DD/YY).
DivideYear	A dividing year value used to determine if the date value belongs to the specified century or the next. The default is the current year plus 40.
Century	The century to assign if the date falls in the dividing year. Otherwise, the result is this century plus one. The default is the current century.

Use this function to convert a date value to the proper century. The resulting date value will have a four-digit year. Since the system has no way of knowing whether a date represents a birthday (from the past) or a maturity date (in the future), a dividing year is required to make the century decision. If the dividing year is not provided, it will default using the equation ((current year + 40) % 100).

The century number is optional and defaults to the current century. If the two-digit year from the date value is greater than the dividing year, the system assumes the date is in the century given. Otherwise, the system assumes date is in the next century.

Example

Assume the current date is 07/01/99. This means the default dividing year is determined as: ((1999 + 40) % 100) = 39.

Note: In this case, % means *modulo*, or *modulus*, which means the value that remains after dividing one number evenly into another. Here is an example: 100 divides into 2,035 twenty even times. 20 times 100 equals 2000. 2035 minus 2000 leaves 35. Therefore, 2035 % 100 = 35.

Function	Result	Explanation
Return(DateCnv( ))	07/01/1999	Defaults to the current date and format 1. Since 99 is greater than 39, this date assumes the current century.
Return(DateCnv (“07/01/00”))	07/01/2000	Since 00 is not greater than 39, this date assumes the next century.
Return(DateCnv (“50/138” , “I”, 50))	2050/138	The given date (50/138) in date format I is May 18, 50. Since 50 is not greater than the dividing year of 50, the result assumes the next century.
Return(DateCnv (“99/138” , “I”, 50))	1999/138	The given date (99/138) in date format I is May 18, 99. Since 99 is greater than the dividing year of 50, the result assumes the current century.

See also Date Functions on page 51

Date Formats on page 52

Using INI Options on page 8

Day

Use this function to get the day portion of a date as an integer.

Syntax

```
Day (Date, Format, Locale)
```

Parameter	Description
Date	Enter a date string. The system assumes your entry to be in the format specified in the Format parameter. The default is the current date.
Format	Enter a date format string that describes the contents of the Date parameter. The default is date format 1 (MM/DD/YY).
Locale	(Optional) Enter the locale code. If you omit this parameter, the system checks the Locale INI option. If the Locale INI option offers no value, the system defaults to USD (United States/English)..

The system determines the day portion of the given date based on the format you specify in the Format parameter.

Example

Here are some examples:

(Assume the current date is 07/01/09.)

Function	Result	Explanation
Return(Day( ))	1	Defaults to the current date and enters the integer 1.
datestring = DateAdd(, , 15); Return(Day (datestring))	16	First the DateAdd function defaults to the current date and adds 15 days which results in a date of July 16, 2009. This date is returned to the target variable *datestring*. The date is then used by the Day function and the integer value of 16 is returned.
Return(Day("09/ 138", "I"))	18	The given date (09/138) in date format I is May 18, 2009. Therefore, the integer value of 18 is returned.

See also

Date Functions on page 51

Locales on page 55

Date Formats on page 52

DateAdd on page 184

DayName

Use this function to enter the name of the day of the week.

Syntax

```
DayName (DayOfWeek, Locale)
```

Parameter	Description
DayOfWeek	Enter an integer to designate the day of the week. 1 - Sunday 2 - Monday 3 - Tuesday 4 - Wednesday 5 - Thursday 6 - Friday 7 - Saturday The default is the current day of the week.
Locale	(Optional) Enter the locale code. If you omit this parameter, the system checks the Locale INI option. If the Locale INI option offers no value, the system defaults to USD (United States/English).

This function is typically used with the WeekDay function. The WeekDay function determines the day of the week number from a given date.

Example

Here are some examples:

(Assume the current date is Saturday, January 3, 2009.)

Function	Result	Explanation
Return(DayName( ))	Saturday	Defaults to the current day of the week and returns Saturday.
DayName(WeekDay (“09/33”, “I”))	Monday	First the WeekDay function determines the day of the week number for the given date and format. DayName then uses this number to return the correct day name: Monday.
DayName (WeekDay(DateAdd(,,-1)))	Friday	First the DateAdd function uses the current date and subtracts one day. WeekDay then determines the number for the day of the week. DayName then determines that the given date is Friday, January 2, 2009 and returns the day name: Friday.
Return(DayName (,"ZAA"))	Saterdag	It returns the name of the current day of the week based and translates that name into Afrikaans.

See also

Date Functions on page 51

Locales on page 55

Using INI Options on page 8

DateAdd on page 184

WeekDay on page 424

DaysInMonth

Use this function to get the number of days in the specified month of a given year.

Syntax

```
DaysInMonth (Month, Year)
```

Parameter	Description
Month	Enter a month number from 1 to 12, with January being 1 and December being 12. The default is the current month.
Year	Enter a year. The default is the current year.

The year value is only used when the month number is 2 (February). The result for February is different if the given year is a leap year. This function is typically used with the Month function. The Month function extracts the month number from a given date.

Example

Here are some examples:

(Assume the current date is 07/01/09.)

Function	Result	Explanation
DaysInMonth ( )	31	Defaults to the current date and returns the value 31 since July has 31 days.
DaysInMonth (Month (“04/15/ 2009”))	30	The Month function extracts the number 04 (April) from the given date. The DaysInMonth function then determines that there are 30 days in April and returns that value.
DaysInMonth(2, 2008)	29	The year 2008 was a leap year, February had 29 days. Therefore the integer 29 is returned.

See also

Date Functions on page 51

Month on page 321

DaysInYear

Use this function to get the number of days in the specified year.

Syntax

```
DaysInYear (Year)
```

Parameter	Description
Year	Enter the year. The default is the current year.

This function returns 365 or 366, depending on whether the year parameter is a leap year. This function is typically used with the Year function. The Year function extracts the year number from a given date.

Example

Here are some examples:

(Assume the current date is 07/01/2008.)

Function	Result	Explanation
DaysInYear ( )	366	2008 is a leap year, therefore the returned value is 366.
DaysInYear (09)	365	The year 2009 is not a leap year and has 365 days.
DaysInYear (Year("2009/09/09", "34"))	365	First the Year function extracts the year number (2009) from the given date using the format specified. The DaysInYear function then determines that the given year has 365 days and returns the integer 365.

See also

Date Functions on page 51

Year on page 448

DBAdd

Use this procedure/function to add a new record to a database table.

Syntax

```
DBAdd (Table, PrefixVariable)
```

Parameter	Description
Table	Enter the name of an open table.
PrefixVariable	(Optional) Enter the name of a DAL variable to associate with the record fields of the table. The default is Table.

The system optionally returns one (1) on success and zero (0) on failure.

Unlike for the DBFirstRec and DBNextRec procedures, the PrefixVariable parameter and the associated fields should have already been defined. For some database handlers, these column names are case sensitive. Columns not required can be left blank.

The actual variable names appended with a prefix are taken from the DFD file. The DFD file is determined by your entry in the Table parameter or by using the column names found in the table if there is no DFD file associated with that table.

Possible causes for failure to add the record include:

- A required column was left blank
- Database specific failure

Example

Here is an example:

Procedure	Result	Explanation
RECORD.Company="Oracle "; RECORD.Lob="Util"; RECORD.Rundate=DATE(); DBAdd("APPIDX" , "RECORD")	1 or 0	Assuming the table APPIDX has the columns *Company*, *Lob*, and *Rundate*, a new record will be added to the table whose values in those columns are Oracle, Util, and the current date, respectively.

See also Database Functions on page 43

DBClose

Use this procedure/function to close a database table.

Syntax

```
DBClose (Table)
```

Parameter	Description
Table	Enter the name of the table you want to close.

The system closes the table and returns one (1) if the table was successfully closed. If the table cannot be closed, it may be because...

- The table was not open, such as if it had already been closed
- There was a database-specific failure

Example

Here is an example:

Procedure	Result	Explanation
DBClose("APPIDX")	1 or 0	Closes the table named APPIDX.

See also

Creating Variable Length Records from Flat Files on page 200

Setting Up Memory Tables on page 50

DBOpen on page 199

Database Functions on page 43

DBDELETE

Use this procedure/function to delete all records which match the key criteria from the database table.

Syntax

```
DBDelete (Table, KeyName1, KeyValue1, KeyName2, KeyValue2,...)
```

Parameter	Description
Table	Enter the name of an open table.
KeyName, KeyValue, ...	Each KeyName refers to the name of a column to search. For some database handlers, this may be a case-sensitive comparison. Each KeyValue is the value of the corresponding KeyName for which to search. For some database handlers, this may be a case-sensitive comparison. At least one KeyName/KeyValue pair are required.

NOTE: You will *not* be prompted for confirmation when deleting records.

The system optionally returns one (1) on success or zero (0) on failure.

This procedure lets you enter as many KeyName and KeyValue combinations as necessary to identify the specific keyed record you want to delete.

This procedure first locates the records using the key you specify. If located, the records will be deleted. If the procedure returns failure, possible causes include:

- There are no records in the table meeting the given criterion
- The column specified in KeyName is not a searchable column
- Database-specific failure

Example Here is an example:

Procedure	Result	Explanation
DBDelete ("APPIDX" , "Company" , "SAMPCO" , "Lob" , "Util")	1 or 0	Assuming *Company* and *Lob* are valid key components for the APPIDX table, the procedure will delete all records with the value SAMPCO in the column named *Company* and the value Util in the column named *Lob*.

See also Database Functions on page 43

DBFIND

Use this procedure/function to retrieve the first record from a database table which satisfies the key criteria.

Syntax

```
DBFind (Table, Variable, KeyName1, KeyValue1, KeyName2,
KeyValue2,...)
```

Parameter	Description
Table	Enter the name of an open table.
Variable	Enter the name of a DAL variable to associate with the record fields retrieved by the procedure. The default is Table.
KeyName, KeyValue, ...	Each KeyName specifies a column to search. For some database handlers, it may be a *case-sensitive*. Each KeyValue is the value of the corresponding KeyName for which to search. For some database handlers, this may be a *case-sensitive* comparison. At least one pair of KeyName/KeyValue are required.

The system optionally returns one (1) on success or zero (0) on failure.

If the Variable parameter has not been defined, it will be created. You can access the table record fields assigned this prefix using the dot (.) operator. For example, assume *Record* is a prefix variable and the table record contains the columns *Company*, *Lob*, and *Policynum*. The values of the individual fields would be referenced as *Record.Company*, *Record.Lob*, and *Record.Policynum*, respectively.

The variable names appended with a prefix are taken from the DFD file associated with the table you specified in the Table parameter or by using the column names found in the table if there is no format file associated with the table.

NOTE: The variable name is truncated to eight characters when you use a long name. Variable names are limited to eight characters if you do not use the DBPrepVars procedure and nine characters if you do. A variable name plus the stem name cannot exceed 32 characters.

This procedure supports a variable number of parameters. As many KeyName and KeyValue combinations required to identify the specific keyed record to retrieve may be defined as parameters. If the record cannot be retrieved, possible causes include:

- There are no records in the table that meet the criteria
- The column specified in KeyName is not a searchable column
- Database specific failure

Example Here is an example:

Procedure	Result	Explanation
DBFind("APPIDX", "RECORD", "Company","Oracle", "Lob", "DM")	1 or 0	Assuming that the APPIDX table has columns named *Company* and *Lob*, and that these columns are a key, the first record containing "Oracle" and "DM" in the appropriate column will be retrieved and associated with the prefix variable RECORD.

See also Database Functions on page 43

DBPrepVars on page 201

DBFirstRec

Use this procedure/function to retrieve the first record in a database table.

Syntax

```
DBFirstRec (Table, PrefixVariable)
```

Parameter	Description
Table	Enter the name of an open table.
PrefixVariable	Enter the name of a DAL variable to associate with the record fields retrieved by the procedure. The default is Table.

The system optionally returns one (1) on success or zero (0) on failure.

If the PrefixVariable parameter has not been defined, it will be created. You can access the table record fields assigned this prefix using the dot (.) operator.

For example, assume *Record* is a prefix variable and the table record contains the columns *Company*, *Lob*, and *Policynum*. The values of the individual fields would be referenced as *Record.Company*, *Record.Lob*, and *Record.Policynum*, respectively.

The actual variable names appended with a prefix are taken from the DFD file associated with the table you specified in the Table parameter or using the column names found in the table if there is no DFD file associated with the table.

Possible causes for failure to retrieve the first record include:

- The table contains no records
- Database-specific failure

Example

Here is an example:

Procedure	Result	Explanation
DBFirstRec (“APPIDX” , “RECORD”)	1 or 0	Retrieves the first record from the APPIDX table and associates the columns with the prefix variable RECORD.

See also

Database Functions on page 43

DBNextRec on page 198

DBNextRec

Use this procedure/function to retrieve the next record in sequence from a database table.

Syntax

```
DBNextRec (Table, PrefixVariable)
```

Parameter	Description
Table	Enter the name of an open table.
PrefixVariable	Enter the name of a DAL variable to associate with the record fields retrieved by the procedure. The default is Table.

The system optionally returns one (1) on success or zero (0) on failure.

If the PrefixVariable parameter has not been defined, it will be created. You can access the table record fields assigned this prefix using the dot (.) operator.

For example, assume *Record* is a prefix variable and the table record contains the columns *Company*, *Lob*, and *Policynum*. The values of the individual fields would be referenced as *Record.Company*, *Record.Lob*, and *Record.Policynum*, respectively.

The actual variable names appended with a prefix are taken from the DFD file associated with the table you specified in the Table parameter or using the column names found in the table if there is no DFD file associated with the table.

If the record cannot be retrieved, possible causes include:

- There are no more records to retrieve
- Some databases require you to call DBFirstRec before you call DBNextRec
- Database specific failure

Example

Here is an example:

Procedure	Result	Explanation
DBNextRec (“APPIDX” , “RECORD”)	1 or 0	Will retrieve the next record from the table APPIDX and associate the field columns with the prefix variable RECORD.

See also

Database Functions on page 43

DBFirstRec on page 197

DBOPEN

Use this procedure/function to open the specified database table in the mode you request.

The DBOpen procedure supports having multiple:

- Simultaneous ODBC connection via different ODBC drivers. See Database Functions on page 43 for more information.
- Tables open in the same database.

Syntax

```
DBOpen (Table, Handler, DFDFile, Mode, Truncate)
```

Parameter	Description
Table	Enter the name of the table you want to open.
Handler	Enter the name of the database handler to associate with the table. If you omit Handler, DBOPEN looks in the DBTable:TableName control group for the DBHandler option. If this option is not present, DBOPEN defaults to the ODBC handler.
DFDFile	Enter the name of the format file to associate with the table. If omitted, the Handler tries to query the information from the database. Note that this may not be supported by all databases.
Mode	Enter a string which specifies the mode in which to open the file. Your options are READ, WRITE, FAIL_IF_EXISTS, and CREATE_IF_NEW. These may be combined by separating them with an ampersand (&), as in “READ&WRITE&FAIL_IF_EXISTS”. You can include spaces between the tokens. If omitted, the open mode defaults to READ & WRITE & CREATE_IF_NEW.
Truncate	Include this parameter to remove all records from a database table. This lets you use dynamic tables with DAL where the tables are created on a fly, records added, and then deleted.

The system returns one (1) if the database table was successfully opened and zero (0) if the table was not opened.

Possible causes of failure include:

- The table does not exist and the Mode parameter did not include the CREATE_IF_NEW directive.
- The table exists and the Mode parameter included the FAIL_IF_EXISTS directive.
- The database handler could not be initialized.
- The table format information could not be found.
- The table is opened for exclusive use by another application.

Creating Variable Length Records from Flat Files

When you use DAL database functions, such as DBOpen and DBClose, to write flat files, the record length is usually fixed and data is padded with spaces to equal the maximum size of the record. You can, however, specify that no trailing spaces are to be output.

You would typically use this capability to output flat files used to create index information you will import into a 3rd-party application, such as FileNET.

To specify no trailing spaces, include the following syntax in your DAL script:

```
DBOPEN(FN_LogFile,"ASCII",".\deflib\filenet.dfd",
"READ&WRITE&TRUNCATE&CREATE_IF_NEW&CLIPSPACES");
```

CLIPSPACES tells the system to remove any trailing spaces.

Keep in mind that CLIPSPACES only affects flat files. For the rest of the databases, each column is set separately and no trailing space exists on the whole record.

Example

Here is an example:

Procedure	Result	Explanation
DBOpen ("APPIDX", "ODBC", ,"READ")	1 or 0	Will open the table named APPIDX for reading and associate it with the ODBC handler. Table information will be queried from the database driver, if possible.
DBOPEN("MYTABLE ","ODBC","D:\deflib\ mytable.dfd","READ& WRITE&TRUNCATE")		This DAL statement removes all rows from the table named *MYTABLE*.

See also

Setting Up Memory Tables on page 50

DBClose on page 193

Database Functions on page 43

DBPrepVars

Use this procedure/function to create the DAL variables associated with a database table record.

Syntax

```
DBPrepVars (Table, PrefixVariable)
```

Parameter	Description
Table	Enter the name of an open table.
PrefixVariable	PrefixVariable is the name of the DAL variable to associate with the record fields retrieved by the procedure. The default is Table.

The system optionally returns one (1) on success or zero (0) on failure.

If the PrefixVariable parameter has not been previously defined, it is created. The table record fields assigned this prefix may be accessed using the dot (.) operator. For example, assume RECORD is a prefix variable and the table record contains the columns COMPANY, LOB, and POLICYNUM. The values of the individual fields would be referenced as RECORD.COMPANY, RECORD.LOB, and RECORD.POLICYNUM, respectively.

The actual variable names appended with a prefix are taken from the DFD file associated with the table you specified in the Table parameter or using the column names found in the table if there is no DFD file associated with the table.

Possible causes for failure to retrieve the first record include:

- The table is not open, or undefined
- Database specific failure
- Database specific failure

Example

Here is an example:

Procedure	Result	Explanation
DBPrepVars(“APPIDX” ,”RECORD”);	1 or 0	Creates the DAL variables for the APPIDX table. Each column name is appended with the prefix variable RECORD.

See also

Database Functions on page 43

DBUnloadDFD

Use this procedure/function to streamline the use of DAL with ODBC and memory tables by creating DFD files and using only memory tables. You can use the DALRUN program to create the DFD files based on a DAL script since it is a one-time operation. You only need to run the script again after table layout changes.

Syntax

```
DBUnloadDFD (TableName, DFDName)
```

Parameter	Description
TableName	Enter the name of the table opened with DBOpen procedure.
DFDName	Enter the name of the output file. The system overwrite this file if it exists.

Keep in mind...

The file name you pass to this procedure as the output name of the DFD file must be appropriate for the platform. For instance, *AAA.DFD* will not work for z/OS.

Example

Here is an example of how you could use this procedure in a DAL script:

```
#rc = DBOpen("MYTABLE","ODBC");
if #rc = 0
* display error
end
#rc = DBUnloadDFD("MYTABLE","aaa.dfd");
if #rc = 0
* display error
end
```

This script unloads a DFD file named *AAA.DFD* which describes the table named *MYTABLE* in the current directory.

See also

Database Functions on page 43

DBUpdate

Use this procedure/function to update the database table record which satisfies the key criteria.

Syntax

```
DBUpdate (Table, Variable, KeyName1, KeyValue1, KeyName2,
KeyValue2,...)
```

Parameter	Description
Table	Enter the name of an open table.
Variable	Enter the name of the stem variable that contains the new information. This variable must first be filled by DBFind, DBFirstRec, or DBNextRec, after which you can modify individual fields before calling DBUpdate. The default is Table.
KeyName, KeyValue, ...	Each KeyName is the name of a column to search. For some database handlers, it may be case-sensitive. Each KeyValue is the value of the corresponding KeyName for which to search. For some database handlers, this may be a case-sensitive comparison. At least one KeyName/KeyValue pair is required.

The system optionally returns one (1) on success or zero (0) on failure.

The actual variable names appended with a prefix are taken from the DFD file associated with the table you specified in the Table parameter or using the column names found in the table if there is no DFD file associated with the table.

This procedure supports a variable number of parameters. As many KeyName and KeyValue pair combinations required to identify the specific keyed record to retrieve and update may be defined as parameters.

If the record cannot be retrieved and updated, possible causes include:

- There are no records in the table meeting the given criterion
- The column specified in KeyName is not a searchable column
- Database-specific failure

NOTE: Since an ASCII file is not a database, it has no ability to have keys. Therefore, you cannot use this function if the MODE is set to "ASCII."

Example Here is an example:

Procedure	Result	Explanation
DBFirstRec("APPIDX", "RECORD"); RECORD.RUNDATE = DATE(); DBUpdate("APPIDX", "RECORD","UNIQUE_ID",RECORD.UNIQUE_ID)	1 or 0	First retrieve the first record from the APPIDX table into the variable named RECORD. Next change the Rundate (assuming that this column is present in the table) to the current date, and update all records whose UNIQUE_ID field matches that in the variable RECORD (assuming that UNIQUE_ID is truly unique, it will update only the first record in the table).

See also

Database Functions on page 43

DBFind on page 195

DBFirstRec on page 197

DBNextRec on page 198

DDTSOURCENAME

Use this function to return the contents of the Source Name field in the DDT file you are currently processing. This function is only applicable during Documaker Server processing.

NOTE: As of version 11.0, DDT fields are physically stored inside FAP files.

Syntax

```
DDTSourceName ( )
```

There are no parameters for this function.

Example

Here is an example:

```
MYROOT = RootName(DDTSourceName())
```

See also

Documaker Server Functions on page 58

Dec2Hex

Use this function to return the hexadecimal equivalent of an integer value.

Syntax

```
Dec2Hex (Value1, Digits)
```

Parameter	Description
Value1	This parameter specifies a integer value to be converted into a hexadecimal string value. If the parameter is not specified as an integer, it will be converted to an integer before performing the operation. The largest hexadecimal value supported is FFFFFFFF. Keep in mind, however, that hexadecimal values are considered *unsigned* while integer values can be both positive and negative. The largest integer value 2,147,483,647 is 7FFFFFFF when represented using hexadecimal. HEX values greater than 80000000 represent negative integer values. Hex value FFFFFFFF represents the integer value -1.
Digits	This parameter defaults to zero (0) and means the resulting hexadecimal value will not have leading zeros. You can set this parameter from one (1) to eight (8) to control the minimum number of hexadecimal digits returned in the string. If you set the minimum too small to represent the value, it will be ignored.

Example

Here is an example:

```
y = 1000
z = Dec2Hex(y)
Result is z = 3E8

y = 254220
z = Dec2Hex(y,8)
Result is z = 0003E10C

y = -2
z = Dec2Hex(y)
Result is z = FFFFFFFE
```

See also

Hex2Dec on page 275

Bit/Binary Functions on page 42

DEFORMAT

Use this function to remove formatting from a specified string and return the result.

Syntax

```
DeFormat (String, FieldType, Format)
```

Parameter	Description
String	Enter a valid string of formatted text. The default is the value of current field text.
FieldType	Enter the field type indicator used to format the first parameter. The default is the value of current field type.
Format	Enter the format of the first parameter. This is the field format entered in the Properties window. The default is the value of current field format.

Some field types do not require format strings to accomplish deformatting. Numeric fields for example, ignore the format specified when deformatting. Numeric fields retain the "-" (negative) and "." (decimal) characters. If these characters were removed during deformatting a completely different value would result.

Example

Here are some examples:

Function	Result	Explanation
DeFormat ("1,234.89", "n")	"1234.89"	Deformat removes commas but retains decimal points for numeric fields.
DeFormat ("ABC.123.DEF", "C", "3,.123.")	"ABCDEF "	Deformat removes the custom format characters (.123.) after the third character, which were previously added to the string.
DeFormat ("$$$$$$11,980.00 ", "n")	11980.00	Deformat removes the "$" characters and commas but retains decimal points for a numeric field.

See also

Field Formats on page 62

String Functions on page 78

DELBLANKPAGES

Use this procedure to remove blank or filler pages in a form set. For instance, you can use this rule to remove blank pages reserved for OMR marks when creating PDF files.

Syntax

```
DelBlankPages ( )
```

There are no parameters for this procedure.

Example

One way to delete blank pages is by using banner page processing in the GenPrint program. You can specify a DAL script which runs at the start of each transaction. The DAL script calls the DelBlankPages procedure.

This will cause blank pages to be removed from each transaction. To do this, you need these INI settings:

```
< Printer >
    EnableTransBanner      = True
    TransBannerBeginScript = PreBatch
< DALLibraries >
    LIB                    = BANNER
```

Here is an example of the BANNER.DAL file:

```
BeginSub PreBatch
DelBlankPages()
EndSub
```

NOTE: You can also remove blank or filler pages using custom code or by using the DPRDelBlankPages procedure, which is available with Docupresentment. See Using Documaker Bridge for more information on the DPRDelBlankPages function.

The API to call from custom code is as follows:

```
DWORD _VMMAPI FAPDelBlankPages(
                VMMHANDLE objectH,) /* formset or form handle */
```

See also the Documaker Administration Guide for information on using banner processing.

See also

AddBlankPages on page 115

Page Functions on page 75

SuppressBanner on page 403

Miscellaneous Functions on page 73

Creating a DAL Script Library on page 5

DELFIELD

Use this procedure/function to delete a field from a section. The system only deletes the field if found and if it is not the current field.

Syntax

```
DelField (Field, Section, Form, Group)
```

Parameter	Description
Field	Enter the name of a field. The default is the current field.
Section	Enter the name of a section that contains the field named. The default is the current section.
Form	Enter the name of a form that contains the section and/or field named. The default is the current form.
Group	Enter the name of the form group that contains the form, section, and/or field named. The default is the current group.

The system returns one (1) if it finds and deletes the field or zero (0) if it does not.

NOTE: The DelField function can not be used in a script called by these AFGJOB rules: PreTransDAL and PostTransDAL.

Example

Lets assume you have the following forms in your form set; *Information* and *Multi-section* in the group named *DAL Test Company*.

The form named *Information* is comprised of two sections; *Part1* and *Part2*. Part1 has these fields: abc1, abc2, and abc3. Part2 has these fields: abc3 and abc4.

The form named *Multi-section* is comprised of three sections: Section1, Section2, and Section3. Section1 has objects with these field names: *a/n*, *date*, *yes/no*, and *multi-line*.

Section2 has the same objects with the same field names as Section1.

Section3 has following objects: *graphic*, *box*, and *input value*.

The DAL script which is executed is on a field named *Test* on *Part1* of *Information*.

Here are some examples:

Procedure	Result	Explanation
Return(DelField ("abc3"));	1	*Abc3* on *Information/Part1* is deleted because the section, field, and group parameters were omitted specified. The system defaulted to the current section, form, and group.
Return(DelField ("abc3", "part2"));	1	*Abc3* on Information/Part2 is deleted because you specified Part2 and the form defaulted to the current form, Information. Note that Abc3 will still exist on Information/Part1.
Return(DelField ("test"));	0	*Test* is not deleted because it is the current field.

Procedure	Result	Explanation
Return(DelField ("a/n"));	0	The field *a/n* is not deleted because it is not on *Information/ Part1*.
Return(DelField ("a/n", "Section1"));	0	The field *a/n* is not deleted because Section1 is not a field on the current form (Information).
Return(DelField ("a/n", "Section1", "Multi-section"));	1	The field *a/n* on *Multi-section/Section1* is deleted because this field is on the specified form/section.
Return(DelField ("a/n", , "Multi-section"));	1	The field *a/n* on *Multi-section/Section1* is deleted because field is on the specified form and the section parameter defaults to the first section on the form. Field *a/n* on *Multi-section/Section1* will still exist. If you immediately execute the script again, the field *a/n* on *Image2* would be deleted.
Return(DelField ("a/n", , , "DAL Test Company"));	1	The field *a/n* on *Multi-section/Section1* is deleted because it was is the first field in the group, *DAL Test Company*. Field *a/n* on *Multi-section/Section2* will still exist. If you immediately execute the script again, the field *a/n* on *Image2* would be deleted.
Return(DelField ("box", "Section3", "Multi-section"));	0	The field *Box* is not deleted because you can only delete variable fields. You can not delete objects such as boxes, charts, lines, text labels, text areas, notes, and so on. You can, however, use the DelLogo function to delete graphics.

See also

Field Functions on page 61

Locating Fields on page 64

DelLogo on page 214

DelForm

Use this procedure/function to remove a form from the document.

Syntax

```
DelForm (Form, Group)
```

Parameter	Description
Form	Enter the name of the form you want to remove.
Group	Enter the name of the group which contains the form you want to remove. The default is the current group.

The system optionally returns one (1) on success or zero (0) on failure.

Remove the specified form from the document set. It is not permitted to remove the form executing the script—the *current* form.

NOTE: Removing a form means that all data associated with the form will be lost.

Example

Here are some examples:

Procedure	Result	Explanation
DelForm(“FORM”)	1 or 0	Assuming FORM is located in the current group and is not the current form, it will be deleted.
DelForm (“FORM\3”, “GRP”)	1 or 0	Locate the third occurrence of FORM within the GRP and delete that form.

See also Section Functions on page 77

DELIMAGE

Use this procedure/function to remove a section from a form. You can use the Paginate parameter to specify whether form pagination should occur after the section is deleted.

Syntax

```
DelImage (Section, Form, Group, Paginate)
```

Parameter	Description
Section	Enter the name of the section.
Form	Enter the name of a form in the form set. The default is the current form.
Group	Enter the name of a group to contain the specified form. The default is the current group.
Paginate	(Optional) This parameter follows the Group parameter. If you enter anything other than a zero (0), it tells the system that you want form pagination to occur upon the successful removal of the section. If you omit this parameter or enter zero (0), the section is deleted, but no other sections are moved to occupy the space left vacant. Subsequent form re-pagination and the application of section origins may change the layout of the form. Here is an example: `DelImage( "mySection", , , 1)` This example omits the Form and Group parameters, but does include the Paginate parameter. Note: If you enter zero (0) or omit this parameter, the function works as it prior to version 11.2. The default is zero (0).

The system optionally returns one (1) on success or zero (0) on failure.

This procedure removes the specified section from the form. It cannot delete the current section. You can delete any section on the current form, as long as it is not the current section.

If the deleted section is the only section on that page, the system also removes the page from the form. If other sections occur on that page, space occupied by the deleted section is left blank.

NOTE: Removing a section means that all data associated with that section will be lost.

This procedure does not update the displayed form. Use the Refresh procedure to update the display.

Example

Here are some examples:

Procedure	Result	Explanation
DelImage(“SEC”)	1 or 0	Delete the specified section from the current form. This assumes that the named section is not the current section.
DelImage(“SEC\3”, ,”GRP”)	1 or 0	Locate the third occurrence of SEC in the specified GRP. If this is not the current section, delete the section.

See also

AddImage on page 122

PaginateForm on page 333

Section Functions on page 77

DelLogo

Use this procedure/function to delete a bitmap graphic (LOG) from a form in the current form set.

Syntax

```
DelLogo (Graphic, Section, Form, Group)
```

Parameter	Description
Graphic	Enter the name of the graphic to be deleted from a section or form. Graphic names are assigned in Studio or Image Editor.
Section	Enter the name of a section that contains the specified graphic. The default is the current section.
Form	Enter the name of a form that contains the section. The default is the current form.
Group	Enter the name of a group to use to locate the specified object. The default is the current group.

This procedure deletes the specified graphic from the section or form. The system optionally returns one (1) on success or zero (0) on failure.

NOTE: Use the Refresh procedure after you use the DelLogo procedure.

Example

Here are some examples:

Procedure	Result	Explanation
DelLogo(“LOG1”)	1 or 0	Deletes LOG1 on the current section, form, group.
DelLogo(“LOG1”, “IMH1\3”,”UpRate”)	1 or 0	Deletes LOG1 on the 3rd occurrence of the named section IMH1 on the form UpRate in the default group.

See also

ChangeLogo on page 163

HaveLogo on page 272

InlineLogo on page 282

RenameLogo on page 359

Logo on page 303

Refresh on page 357

Graphics Functions on page 71

DelWIP

Use this procedure/function to delete the work-in-process and its associated data.

Syntax

```
DelWIP ( )
```

There are no parameters for this procedure.

This procedure removes the current work-in-process (form set) information from the WIP.DFD file, deletes the associated data files (POL and DAT, if they exist) from the WIP subdirectory, and writes comments to the AFELOG file to note the work-in-process (form set) was deleted.

This procedure returns success (1) if no error occurred during the complete process, otherwise a failure (0). This procedure only works with the Entry module, it will not work in the data entry mode of Studio or Image Editor.

Example

Here is an example:

Procedure	Result	Explanation
DelWIP ( )	Deletes the work-in-process.	Deletes information associated with the work-in-process and updates the AFELOG file.

See also

WIP Functions on page 88

Documaker Workstation Administration Guide

Documaker Workstation User Guide

DestroyList

Use this function to destroy the XML tree created by the LoadXMLList function.

Syntax

```
DestroyList (%xXMLTree)
```

Parameter	Description
%xXMLTree	Enter a list type DAL variable that passes the XML tree handle.

The system returns one (1) for success or zero (0) for failure. The returned DAL variable is of the integer type.

Example

For an example, see the DAL script in Scenario 2 on page 90.

See also

XML Functions on page 89

LoadXMLList on page 302

DeviceName

Use this function to return the current output device file name, such as the name of the current print stream output file.

Syntax

```
DeviceName ( )
```

There are no parameters for this function.

Example

This example shows an example post-transaction banner DAL script:

```
IF TotalSheets() > 16000
    #COUNTER += 1
    CurFile = DeviceName()
    Drive = FileDrive(CurFile)
    Path = FilePath(CurFile)
    Ext = FileExt(CurFile)
    RecipBatch = RecipBatch()
    NewFile = FullFileName(Drive,Path,RecipBatch & #COUNTER,Ext)
    SetDeviceName(NewFile)
    BreakBatch()
END
```

NOTE: See FileDrive, FileExt, FileName, FilePath, and FullFileName for information on using DAL functions to manipulate file names.

Keep in mind...

- These print drivers are supported: PCL5, PCL6, PST, MET, AFP, PDF, HTML, and RTF. These print drivers *are not* supported: EPT, MDR, and GDI.
- All platforms are supported, but note that while UniqueString is supported on z/OS, z/OS does not support PDF or long file names, so the PDF example does not apply to z/OS.
- Both multi- and single-step processing are supported.

The only DAL function actually involved in splitting the print stream is BreakBatch. The others make it easier to implement this functionality. For example, since you need to name the new print stream, you use the SetDeviceName procedure. To find the name of the current device, you use the DeviceName function. If you need to create unique file names, you can use the UniqueString function.

NOTE: While you can call all of these DAL functions in Documaker Server or Documaker Workstation, the BreakBatch and SetDeviceName functions are not applicable in Documaker Workstation because it does not use the batch printing engine. DeviceName and UniqueString are applicable to both Documaker Workstation and Documaker Server.

See also

Printer and Recipient Functions on page 76

BreakBatch on page 158

SetDeviceName on page 373

UniqueString on page 421

DiffDate

Use this function to determine the number of days difference between two dates and enter that value.

Syntax

```
DiffDate (Date1, Format1, Date2, Format2)
```

Parameter	Description
Date1	Enter a date string. The system assumes this date string is in the format specified by the Format1 parameter. The default is the current date.
Format1	Enter a date format string that describes the Date1 parameter. The default is date format 1 (MM/DD?YY).
Date2	Enter a date string. The system assumes this date string is in the format specified by the Format2 parameter. The default is the current date.
Format2	Enter a date format string that describes the Date2 parameter. The default is date format 1.

The system returns a positive value if the first date is earlier than the second date. The result is negative if the first date is later than the second date. Use the DiffDate function when the chronological order of the dates is important.

Example

Here are some examples:

(Assume the current date is 07/01/95.)

Function	Result	Explanation
DiffDate (“7/15/95”)	-14	The second parameter defaults to the current date. The resulting difference in days is -14, because date1 is later in time than the current date.
DiffDate (“06/01/95”, “1”)	30	Note that the result is positive because the first date is earlier than the current date.
DiffDate (“October 31, 1961”, “4”, “10/31/95”, “1”)	12418	Note that two different date formats are used.

See also

Date Functions on page 51

Date Formats on page 52

DiffDays

Use this function to determine the absolute number of days difference between two dates and return that value.

Syntax

```
DiffDays (Date1, Format1, Date2, Format2)
```

Parameter	Description
Date1	Enter a date string. The system assumes this date string is in the format specified by the Format1 parameter. The default is the current date.
Format1	Enter a date format string that describes the Date1 parameter. The default is date format 1 (MM/DD?YY).
Date2	Enter a date string. The system assumes this date string is in the format specified by the Format2 parameter. The default is the current date.
Format2	Enter a date format string that describes the Date2 parameter. The default is date format 1.

The system always returns a positive number regardless of which date string parameter is later in time. The result is always given in number of days regardless of the number of months and/or years that are included.

Example

Here are some examples:

(Assume the current date is 07/01/95.)

Function	Result	Explanation
DiffDays ("7/15/95")	14	The second parameter defaults to the current date. The resulting difference in days is 14.
DiffDays ("06/01 95", "1")	30	The second parameter defaults to the current date.
DiffDays ("October 31, 1961", "4", "10/31/95", "1")	12418	Note that two different date formats are used and that the result includes several years worth of days.

See also

Date Functions on page 51

Date Formats on page 52

Using INI Options on page 8

DIFFHOURS

Use this function to calculate the absolute time difference in hours between two times. The system returns an integer value, rounded down to the number of whole hours.

Syntax

```
DiffHours (Time1, Format1, Time2, Format2)
```

Parameter	Description
Time1	Enter a time string. The system assumes this time string is in the format specified by the Format1 parameter. The default is the current time.
Format1	Enter a time format string that describes the Time1 parameter. The default is time format 1 (HH:MM:SS).
Time2	Enter a time string. The system assumes this time string is in the format specified by the Format2 parameter. The default is the current time.
Format2	Enter a time format string that describes the Time2 parameter. The default is time format 1.

The difference between two times is always positive. It does not matter which time string is larger.

Example

Here are some examples:

(Assume the current time is 10:30:10 AM)

Function	Result	Explanation
Return(DiffHours (“09:30:00 AM”,2))	1	The given time is in format 2. The difference in hours between 9:30:00 AM and the current time is one hour.
Return(DiffHours (“10:30:00 AM”,2))	0	The given time is in format 2. The difference in hours between 10:30:00 AM and the current time is zero.

See also

Time Formats on page 80

DiffMinutes

Use this function to calculate the absolute time difference in minutes between two times. The system returns an integer value.

Syntax

```
DiffMinutes (Time1, Format1, Time2, Format2)
```

Parameter	Description
Time1	Enter a time string. The system assumes this time string is in the format specified by the Format1 parameter. The default is the current time.
Format1	Enter a time format string that describes the Time1 parameter. The default is time format 1 (HH:MM:SS).
Time2	Enter a time string. The system assumes this time string is in the format specified by the Format2 parameter. The default is the current time.
Format2	Enter a time format string that describes the Time2 parameter. The default is time format 1.

The difference between two times is always positive. You can enter the Time parameters in any order. It does not matter which Time parameter is earlier.

Example Here is an example:

(Assume the current time is 4:04:34 pm.)

Function	Result	Explanation
DiffMinutes (“2:04:34PM”, 2,)	120	The second parameter defaults to the current time. The resulting difference in minutes between the given time and the current time is a total of 120 minutes.

See also Time Formats on page 80

DiffMonths

Use this function to determine the number of months difference between two dates and return that value.

Syntax

```
DiffMonths (Date1, Format1, Date2, Format2)
```

Parameter	Description
Date1	Enter a date string. The system assumes this date string is in the format specified by the Format1 parameter. The default is the current date.
Format1	Enter a date format string that describes the Date1 parameter. The default is date format 1 (MM/DD?YY).
Date2	Enter a date string. The system assumes this date string is in the format specified by the Format2 parameter. The default is the current date.
Format2	Enter a date format string that describes the Date2 parameter. The default is date format 1.

The system calculates the number of complete months between given dates. For example, from 2/10 to 3/10 is considered one month, and from 2/10 to 3/15 is also considered one month.

The system always returns a positive number regardless of which date string parameter is later in time. The result is always given in number of months regardless of the number of years included.

Example

Here are some examples:

(Assume the current date is 07/01/95.)

Function	Result	Explanation
DiffMonths (“7/15/95”)	0	The second parameter defaults to the current date. Since the value does not equal an entire month the result is 0.
DiffMonths (“05/01/95”, “1”)	2	The second parameter defaults to the current date.
DiffMonths (“October 31, 1961”, “4”, “10/31/95”, “1”)	408	Note that the result includes several years worth of months. In addition, two different date formats are used.

See also

Date Functions on page 51

Date Formats on page 52

Using INI Options on page 8

DiffSeconds

Use this function to calculate the absolute time difference in seconds between two times. The system returns an integer value.

Syntax

```
DiffSeconds (Time1, Format1, Time2, Format2)
```

Parameter	Description
Time1	Enter a time string. The system assumes this time string is in the format specified by the Format1 parameter. The default is the current time.
Format1	Enter a time format string that describes the Time1 parameter. The default is time format 1 (HH:MM:SS).
Time2	Enter a time string. The system assumes this time string is in the format specified by the Format2 parameter. The default is the current time.
Format2	Enter a time format string that describes the Time2 parameter. The default is time format 1.

The difference between two times is always positive. It does not matter which time string is larger.

Example

Here is an example:

(Assume the current time is 4:04:34 pm.)

Function	Result	Explanation
DiffSeconds ("2:04:35PM", 2,)	7199	The second parameter defaults to the current time. The resulting difference in seconds between the given time and the current time is a total of 7199 seconds.

See also

Time Formats on page 80

DIFFTIME

Use this function to calculate the difference in time between two times. The system returns a signed (positive or negative) value, given in seconds.

Syntax

```
DiffTime (Time1, Format1, Time2, Format2)
```

Parameter	Description
Time1	Enter a time string. The system assumes this time string is in the format specified by the Format1 parameter. The default is the current time.
Format1	Enter a time format string that describes the Time1 parameter. The default is time format 1 (HH:MM:SS).
Time2	Enter a time string. The system assumes this time string is in the format specified by the Format2 parameter. The default is the current time.
Format2	Enter a time format string that describes the Time2 parameter. The default is time format 1.

The system returns a positive value if Time1 is earlier than Time2. The result is negative if Time2 is earlier than Time1.

Example

Here is an example:

(Assume the current time is 4:06:50 pm.)

Function	Result	Explanation
DiffTime ("4:06:40PM", 2)	+10	The second parameter defaults to the current time. The resulting difference in time is +10 seconds.

See also Time Formats on page 80

DiffYears

Use this function to determine the number of years difference between two dates and return that value.

Syntax

```
DiffYears (Date1, Format1, Date2, Format2)
```

Parameter	Description
Date1	Enter a date string. The system assumes this date string is in the format specified by the Format1 parameter. The default is the current date.
Format1	Enter a date format string that describes the Date1 parameter. The default is date format 1 (MM/DD?YY).
Date2	Enter a date string. The system assumes this date string is in the format specified by the Format2 parameter. The default is the current date.
Format2	Enter a date format string that describes the Date2 parameter. The default is date format 1.

The system calculates the number of complete years between the given dates. For example, from 2/10/08 to 2/10/09 is considered one year, while 3/1/08 to 2/29/09 is considered zero years.

The system always returns a positive number, regardless of which date string parameter occurs later.

NOTE: When calculating leap years, February 28th and 29th are considered equal, since both represent the last day of February. For example, February 29, 2008 to February 28, 2009, is considered one year.

Example

Here are some examples (assume the current date is 07/01/09):

01/31/2008 to 01/30/2009 = zero years difference (it will not be a year until 01/31/2009 as the year 2008 is a leap year)

Function	Result	Explanation
DiffYears ("7/15/09")	0	The second parameter defaults to the current date. Since the value is not an entire year, the result is zero (0).
DiffYears ("01/31/2009", "4", "01/30/2009", "1")	0	The result will not become one (1) until January 31, 2009.
DiffYears ("01/010/05", "1")	4	The second parameter defaults to the current date.
DiffYears ("October 31, 1975", "4", "10/31/09", "1")	34	Note that the result includes numerous years. In addition, two different date formats are used.

See also Date Functions on page 51

Date Formats on page 52

Using INI Options on page 8

DupForm

Use this procedure/function to duplicate a form. No data is duplicated, except global data that propagates in naturally.

NOTE: For the system to be able to duplicate a form, you must first check the Multicopy option in that form's Properties window.

Syntax

```
DupForm (Form, Group)
```

Parameter	Description
Form	Enter the name of the form you want to duplicate
Group	(Optional) Enter the name of the group if the form is not in the current group.

This procedure locates the named form and duplicates it if the form flags indicate that it can be duplicated. The system inserts the duplicated form immediately after the original. You cannot specify another insertion point.

If the procedure is successful in duplicating the form, it returns a non-zero value, otherwise zero (0) is returned. This procedure can fail for these reasons:

- Could not locate the form or form group specified
- The Multicopy option is not checked for the form
- Lack of available memory

You can only use this procedure in scripts hosted by AFEMain or other Entry-related applications.

Syntax

AddForm on page 119

AddForm_Propagate on page 120

CopyForm on page 174

TriggerForm on page 414

WIP Functions on page 88

EMBEDLOGO

Use this procedure/function to save graphic data, including full color data, inside the NAFILE.DAT file. This lets you capture and archive form set specific section data such as pictures, scans, or signatures along with the form set.

Syntax

```
EmbedLogo (Graphic, Section, Form, Group)
```

Parameter	Description
Graphic	Enter the name of the graphic you want to embed.
Section	Enter the name of a section that contains the graphic. If the current section does not contain the graphic being referenced this parameter is required to locate the section; otherwise this parameter is optional.
Form	Enter the name of the form that contains the graphic you specified. If the current form does not contain the section for the graphic being referenced this parameter is required to locate the graphic; otherwise, this parameter is optional.
Group	Enter the name of the form group that contains the graphic you specified. If the current form is not in the form group that contains the graphic being referenced this parameter is required to locate the graphic; otherwise, this parameter is optional.

Execute this DAL procedure for each graphic on the form or section. This procedure sets the embedded graphic flag in the graphic bitmap structure. Documaker Workstation and Documaker Server check for this flag when they write to the NAFILE.DAT file.

If the flag is not set, the graphic data is not written to the NAFILE.DAT file. Place this procedure in the data field of the IF or DAL rule when used with Documaker Server.

This procedure returns success (1) if no error occurred during the complete process, otherwise a failure (0).

NOTE: If the LoadCordFAP in the RunMode control group is set to No; then Documaker Server execution requires you to include the section level rule, CheckImageLoaded.

Example

Here is an example:

Procedure	Result	Explanation
rc = EmbedLogo("JaneDoe");	1	The embedded graphic flag in the JaneDoe bitmap structure will be set to On.

See also

Section Functions on page 77

Exists

Use this function to determine if a DAL symbolic variable exists. This can be useful because referencing a variable that does not exist will cause a runtime syntax error. You can use this function to verify that DAL variables which are created external to your script have been created before you try to reference them.

Syntax

```
Exists (Symbol)
```

Parameter	Description
Symbol	Specify the name of a DAL symbolic variable. This can be from an expression or from another string variable.

The system returns (1) if the variable exists, otherwise it returns zero (0).

Example

Here is an example. Assume the string variables 'tbl_1', 'tbl_2', 'tbl_3', and 'tbl_4' respectively contain: '*Ford*', '*Chev*', '*Olds*', and '*VW*'.

```
If Exists("tbl_" & #line) Then
        Return ( GetValue("tbl_" & #line) )
    Else
        Return (" ")
End
```

In this example, if *#line* is set to 3, the string '*Olds*' is returned. If *#line* is set to 5, a 'blank' is returned.

See also

GetValue on page 263

Miscellaneous Functions on page 73

FIELDFORMAT

Use this function to return the format string associated with the field's type.

Syntax

```
FieldFormat (Field, Section, Form, Group)
```

Parameter	Description
Field	Enter the name of a field. The default is the name of the current field.
Section	Enter the name of the section that contains the field you specified in the Field parameter. The default is the current section.
Form	Enter the name of the form that contains the section and/or field you specified. The default is the current form.
Group	Enter the name of the form group that contains the form, section, and/or field you specified. The default is the current group.

Certain field types (like date and numeric data types) will sometimes have additional format information specified. Typically, a user will not be concerned with this type since the fields are designed appropriately for data entry. However, a script may be written that does not assume the field's format and must query the information to be accurate.

The value returned from this function is a string. If a field cannot be located matching the specified information, an empty string will be returned.

Example

Here are some examples:

Function	Result	Explanation
Return(FieldFormat ("First"))	ZZ9.99	Locate the field and return its format. This example assumes that the field was a numeric type with a format of ZZ9.99.
Return(FieldFormat ("Second"))		This example returns an empty string. This either means the field has no format string or could not be located.
Return(FieldFormat ("Third", , "FRM"))	1/4	Locate the form specified within the current form group. Then locate Third anywhere on that form. If found, the field's format is returned which may be an empty string. This example returned a format "1/4" which is a particular date format.

See also

Field Functions on page 61

Field Formats on page 62

Locating Fields on page 64

FIELDNAME

Use this function to return the name of a field relative to another field.

Syntax

```
FieldName (Count, Field, Section, Form, Group)
```

Parameter	Description
Count	Enter positive or negative number. The system uses your entry to move beyond the field you specify. The default is zero (0).
Field	Enter the name of a field. The default is the current section.
Section	Enter the name of a section that contains the field named. The default is the current section.
Form	Enter the name of a form that contains the section and/or field named. The default is the current form.
Group	Enter the name of the form group that contains the form, section, and/or field named. The default is the current group.

Keep in mind you can use an asterisk (*) as the object name to match parent objects. This lets you find objects without explicitly knowing the parent names.

NOTE: For optimal performance, avoid using wildcards (*) when searching for field, section, or form names.

At first glance, FieldName may seem like an odd function. After all, one of its parameters is a field name. This function first locates the specified field. If you omit the FieldName parameter, the system uses the current field. Then the count is used to move to another field on the section.

A positive or negative number can be used for the count parameter. A positive count moves forward from the located field. A negative count moves backward from the located field. Forward and backward refer to the order that the field appears in the section's edit list, not necessarily to physical position on the section. All fields are included in the search regardless of whether they are editable or not.

If the system cannot find a field that matches the information you specified, it returns an empty string.

Example

Here are some examples: (Assume the section has three fields named First, Second, and Third, which occur in that order.)

Function	Result	Explanation
Return(Field Name (1, “Second”))	Third	Locate the field named Second and then move to the next field.

Function	Result	Explanation
Return(Field Name (-1, “Second”))	First	Locate the field named Second and then move to the previous field.
Return(Field Name (8, “MyField”, , “FRM”))	a name or ““	Locate the form specified within the current form group. Then locate MyField anywhere on that form. If found, move forward eight more fields. If a field matches this criteria, its name will be returned, otherwise an empty string is returned.

See also Name Functions on page 74

Field Formats on page 62

Locating Fields on page 64

FIELDPROMPT

Use this function to return the text of the prompt for a field.

Syntax

```
FieldPrompt (Field, Section, Form, Group)
```

Parameter	Description
Field	Enter the name of a field. The default is the current field.
Section	Enter the name of a section that contains the field named. The default is the current section.
Form	Enter the name of a form that contains the section and/or field named. The default is the current form.
Group	Enter the name of the form group that contains the form, section, and/or field named. The default is the current group.

The system returns a string unless it cannot find a field that matches your criteria. If the system cannot find a field that matches the criteria you specified, it returns an empty string.

NOTE: For optimal performance, avoid using wildcards (*) when searching for field, section, or form names.

Example

Here are some examples:

Function	Result	Explanation
Return(FieldPrompt(“Name”))	Name	Locates the field on the current section and returns its prompt.
Return(FieldPrompt(“Address1”))	Street Address	Locates the field on the current section and returns its prompt.

See also

Field Functions on page 61

Field Formats on page 62

Locating Fields on page 64

FieldRule

Use this procedure/function when you need to execute a field-level rule in a DAL script. See the Rules Reference for more information on field-level rules.

NOTE: The FieldRule procedure requires a section to be able to process. It cannot be used in an external DAL script called by the SETRCPTB.DAT file, a custom rule, the RecipIf rule, or placed in the SETRCPTB.DAT custom rule parameters field.

Syntax

```
FieldRule ( )
```

There are no parameters for this procedure.

This procedure lets you execute field-level rules from within a DAL script. The DAL script is called by one of these Documaker processing rules: DAL or IF. This procedure requires the same number of parameters as are required for a field level rule. While not all fields must contain data, you *must* include the correct number of delimiters.

You can use overflow variables if the called field level rule supports overflow. Generally, the IF rule does not support overflow but it can be supported using the FieldRule procedure. See the examples for this procedure for more information.

NOTE: All semicolons in a field-level rule *must be* replaced with two colons (::). If any of your DDT parameters contain quotation marks ("), use instead apostrophes (') to send in the DDT information. Here is an example:

FIELDRULE('::0::1::FIELD_NAME::45::4::FIELD_NAME::0::4::::move_it::::!
/field1/field2[field3="Yes"]::N::N::N::N:::::::')

Here is a list of parameters for this procedure with sample entries. The entries illustrate the following example. An asterisk indicates the parameter is generally required, depending on the rule you are using.

Parameter	Description	Example
File number	* (required by TblLkUp)	0
Record number	* (required for overflow)	1
Source field name	* (required by TblText)	Town_State
Source field offset	*	55
Source field length	*	9
Destination field name	*	Rec-Town_State
Destination field offset	*	0
Destination field length	*	25

Parameter	Description	Example
Format mask	*	blank
Field rule name	*	KickToWip
Rule parameters	* (also called *data*)	blank
Flag1	(also called *not required*)	N
Flag2	(also called *host required*)	N
Flag3	(also called *operator required*)	Y
Flag4	(also called *either required*)	N
X position		3001
Y position		5602
Font ID		11010

Example

For example, suppose you want the transaction sent to WIP when the record PRODAREC, at offset 11, contains a string of four characters ("0000") starting at position 20. And, you always want the system to get 25 characters of data from PRODAREC, starting at position 65. Furthermore, you want the system to remove any trailing spaces.

For this scenario, you would use the FieldRule procedure to call the KickToWIP field level rule and use the standard IF rule to do the rest. The script for this example would look like this:

```
::A={11,PRODAREC 20,4}::B={11,PRODAREC 65,25}:: IF(A='0000')::
FieldRule("::0::1::Town_State::55:9::;Rec-Town_State::0::25::::
KickToWip::::N::N::Y::N::3001::5602::11010::")::Else::B=Trim(B)::
Return("^" & B & "^")::End::Return("^" & 1 & "^");
```

Here's another example. Suppose you want to move multiple lines of text from *N* number of specific external extract records to the output buffer when the HEADERREC record (at offset 11) contains an *F* in position 1.

For this scenario, you could use the FieldRule procedure to call the MoveExt rule and use the standard IF rule to do the rest. The script for this example would look like this:

```
CON={11,HEADERREC 1,1}:: A=FIELDRULE("::0::1::E::45::4::PREM/OPS
RATE1::0::4::::moveext::@GETRECSUSED,QCPVR5,OVSYM1/
11,CLSSCDREC::N::N::N::N::::::::")::if(CON='F')::return("^" & A &
"^")::end ;N;N;Y;N;12461;2119;16010
```

See also

Documaker Server Functions on page 58

Field Formats on page 62

Locating Fields on page 64

FieldType

Use this function to return the data type information associated with the section field.

Syntax

```
FieldType (Field, Section, Form, Group)
```

Parameter	Description
Field	Enter the name of a field. The default is the current field.
Section	Enter the name of a section that contains the field you specified. The default is the current section.
Form	Enter the name of a form that contains the section and/or field you specified. The default is the current form.
Group	Enter the name of the form group that contains the form, section, and/or field you specified. The default is the current group.

Typically, a field type will be a token of one or two characters used to control the display of the variable data in the field. Typically, a user will not be concerned with this value, since the form should be designed appropriately for data entry. However, a script may be written that does not assume the field's type and must query the information to be accurate.

The value returned from this function is a string. If a field cannot be located matching the specified information, an empty string will be returned.

Example Here are some examples:

Function	Result	Explanation
Return(FieldType("First"))	n	Locate the field and return its type. This example assumes that the field was a numeric type.
Return(FieldType("Second"))	k	This example returns *K* which corresponds to the International Alphanumeric data type.
Return(FieldType("MyField", , "FRM"))	m	Locate the form specified within the current form group. Then locate MyField anywhere on that form. If found, the field's type is returned. In this example, *M* corresponds with the X or Space field type.

See also Field Functions on page 61

Field Formats on page 62

Locating Fields on page 64

FieldX

Use this function to return the X coordinate of a variable field object.

Syntax

```
FieldX (Field, Section, Form, Group)
```

Parameter	Description
Field	Enter the name of a field. The default is the current field.
Section	Enter the name of the section that contains the field you specified. The default is the current section.
Form	Enter the name of a form that contains the section and/or field you specified. The default is the current form.
Group	Enter the name of the form group that contains the form, section, and/or field you specified. The default is the current group.

You can use this function and the FieldY function to get the X and Y coordinates of a field object. Coordinates are stored in FAP units — 2400 units per inch. This means that an object located at (2400, 2400) occurs one inch from the top and one inch from the left.

Example

Here are some examples:

(Assume the field named *MyField* is located at X coordinate 1250.)

Function	Result	Explanation
Return(FieldX( ))	1250	Returns the current field's X coordinate.
Return(FieldX ("MyField"))	1250	Returns the field's X coordinate if the field is located on the current section.
Return(FieldX("My Field", "IMG\2", ,"GRP"))	1250	Returns the X coordinate of MyField located on the second occurrence of IMG within the specified form set group.

See also

Field Functions on page 61

Field Formats on page 62

Locating Fields on page 64

FieldY on page 239

FIELDY

Use this function to return the Y coordinate of a variable field object.

Syntax

```
FieldY (Field, Section, Form, Group)
```

Parameter	Description
Field	Enter the name of a field. The default is the current field.
Section	Enter the name of the section that contains the field you specified. The default is the current section.
Form	Enter the name of the form that contains the section and/or field you specified. The default is the current form.
Group	Enter the name of the form group that contains the form, section, and/or field you specified. The default is the current group.

You can use this function and the FieldX function to get the X and Y coordinates of a field object. Coordinates are stored in FAP units — 2400 units per inch. This means that an object located at (2400, 2400) occurs one inch from the top and one inch from the left.

Example

Here are some examples:

(Assume the field named *MyField* is located at Y coordinate 6020.)

Function	Result	Explanation
Return(FieldY( ))	6020	Return the current field's Y coordinate.
Return(FieldY ("MyField"))	6020	Returns the field's Y coordinate if located on the current section.
Return(FieldY("My Field", , "FRM"))	6020	Returns the first occurrence of MyField on the specified form.

See also

Field Functions on page 61

Field Formats on page 62

Locating Fields on page 64

FieldX on page 238

FileDrive

Use this function to get the drive component of a file name.

Syntax

```
FileDrive (FullFileName)
```

Parameter	Description
FullFileName	Enter a string that contains a fully qualified file name, such as: `"d:\mypath\myfile.ext"`

The system returns a string that contains the drive component of that file name.

Example

Here is an example:

```
MYDRIVE = FileDrive("d:\mypath\myfile.ext")
```

In this example, MYDRIVE would contain:

```
"d:"
```

See also

FilePath on page 243

FileName on page 242

FileExt on page 241

FullFileName on page 249

File and Path Functions on page 68

FILEEXT

Use this function to get the extension component of a file name.

Syntax

```
FileExt (FullFileName)
```

Parameter	Description
FullFileName	Enter a string that contains a fully qualified file name, such as: `"d:\mypath\myfile.ext"`

The system returns a string that contains the extension component of that file name.

Example

Here is an example:

```
MYEXT = FileExt("d:\mypath\myfile.ext")
```

In this example MYEXT would contain:

".ext"

See also

File and Path Functions on page 68

FullFileName on page 249

FileDrive on page 240

FilePath on page 243

FileName on page 242

FILENAME

Use this function to get the name component of a file name.

Syntax

```
FileName (FullFileName)
```

Parameter	Description
FullFileName	Enter a string that contains a fully qualified file name, such as: `"d:\mypath\myfile.ext"`

The system returns a string that contains the name component of that file name.

Example

Here is an example:

```
MYNAME = FileName("d:\mypath\myfile.ext")
```

In this example, MYNAME would contain:

"myfile"

See also

File and Path Functions on page 68

FullFileName on page 249

FileDrive on page 240

FilePath on page 243

FileExt on page 241

FILEPATH

Use this function to get the path component of a file name.

Syntax

```
FilePath (FullFileName)
```

Parameter	Description
FullFileName	Enter a string that contains a fully qualified file name, such as: `"d:\mypath\myfile.ext"`

The system returns a string that contains the path component of that file name.

Example

Here is an example:

```
MYPATH = FilePath("d:\mypath\myfile.ext")
```

In this example, MYPATH would contain:

"\mypath\"

See also

File and Path Functions on page 68

FullFileName on page 249

FileDrive on page 240

FileExt on page 241

FileName on page 242

FIND

Use this function to return the position of a substring within another string.

Syntax

```
Find (String, Substring, Integer)
```

Parameter	Description
String	Enter a valid string. The default is the value of the current field.
Substring	A string of one or more characters that will be located in parameter one.
Integer	Choose from these options: 0 - a left to right search 1 - a right to left search Both search options return a position relative to the first (left-hand) character of the string parameter. The default is zero (0).

The system returns a zero (0) if the substring is not found in the search string, otherwise it returns the position of the substring. The search is not case sensitive.

Example

Here are some examples:

(Assume the current field contains the text *Insured's responsibility*.)

Function	Result	Explanation
Return(Find (, "RESP"))	11	Defaults to the current field and finds the first occurrence of "RESP" at position 11. Note that the search is not case sensitive.
Return(Find (, "usual and customary"))	0	The term "usual and customary" is not found in the current field.
Return(Find ("Complete all the blanks.", "all"))	10	Searching left to right, "all" was first found at position 10.
Return(Find ("Complete all the blanks.", "all", 1))	10	Searching right to left, "all" was first found at position 10.

See also

String Functions on page 78

FORMAT

Use this function to format a string field and return the result.

Syntax

```
Format (String, FieldType, Format)
```

Parameter	Description
String	Enter a valid string of non-formatted text. The default is the current field.
FieldType	Enter the field type indicator you want the system to use to format the first parameter. The default is the current field type.
Format	Enter the field format you want the system to use to format the first parameter. The default is the current field format.

The system applies formatting to a given string. Some field types do not require format strings to accomplish formatting. For example, the *X* field type indicator automatically uppercases all letters in a string without requiring a format.

NOTE: The variable field *must be* the same length as the format mask.

Example

Here are some examples:

Function	Result	Explanation
Return(Format (“1234.89”, “n”, “zzz,zzz.99”))	1,234.89	Formats the field as numeric, by adding a comma and using two decimal positions, as specified in the Format parameter.
Return(Format (“ABCDEF”, “C”, “3,.123.”))	ABC.123. DEF	Custom formats the field by adding *.123.* after the third input character.
Return(Format (“222334444”, “n”, “999-99-9999”))	222-33-4444	Formats the field as a numeric, by adding hyphens as specified in the Format parameter.

See also

Field Formats on page 62

String Functions on page 78

FormDesc

Use this function to retrieve the description specified in the FORM.DAT file for a specific form.

Syntax

```
FormDesc (Count, StartForm, Group)
```

Parameter	Description
Count	An index reference to locate a form before or after the specified form. To move backwards, enter a negative number. The default is zero (0).
StartForm	Enter the name of a form from which to start the search. The default is the current form.
Group	Enter the name of a group which contains the form you specified. The default is the current group.

The system lets you get the description specified in the FORM.DAT file for the specified form, relative to a known form. If you omit all parameters, the system returns the description of the current form.

The Count parameter tells the system to move a number of forms forwards or backwards from the specified form before it returns the form description.

If the system cannot locate the starting form or the Count parameter tells the system to move beyond the number of forms contained in the group, the system returns an empty string.

Example

Here are some examples:

Assume there are three forms: FORMA, FORMB, and FORMC. Also assume the current form is FORMB and its description is Fire Form # 2345.

Function	Result	Explanation
FormDesc( )	Fire Form # 2345	No parameters will result in returning the current form description.
FormDesc (2, "FormC")	Empty string	Returns an empty string if the form cannot be located.
FormDesc (-1, "FormC")	Fire Form # 2345	Locates FORMC in the current group. Then returns the description of the form that occurs before this form.

See also

FormName on page 247

ImageName on page 277

Name Functions on page 74

DAL Script Examples on page 35

FormName

Use this function to get the name from a form.

Syntax

```
FormName (Count, StartForm, Group)
```

Parameter	Description
Count	Enter an index reference to use to locate a form before or after the specified form. The default is zero (0).
Startform	Enter the name of the form from which to start the search. The default is the current form.
Group	Enter the name of a group that contains the form you specified. The default is the current group.

The system returns the name of the form it located.

Keep in mind you can use an asterisk (*) as the object name to match parent objects. This lets you find objects without explicitly knowing the parent names.

NOTE: For optimal performance, avoid using wildcards (*) when searching for field, section, or form names.

The system lets you get the name of a form relative to a known form. If you omit all parameters, the system returns the name of the current form. The Count parameter moves a number of forms forwards or backwards (negative) from a located form before returning the form name.

If the starting form cannot be located or the Count parameter causes the system to move beyond the number of forms contained in the group, the system returns an empty string.

If there is more than one copy of the form, the name returned contains the occurrence notation used by DAL functions to locate forms. For instance, a name like FORM\3 identifies the third copy of FORM within the same group.

Example

Here are some examples:

(Assume there are three forms: FORMA, FORMB, and FORMC. Also assume the current form is FORMB.)

Function	Result	Explanation
FormName( )	FORMB	No parameters will result in returning the current form name.
FormName (-1, “FormC”)	FORMB	Locates FORMC in the current group. Then returns the name of the form that occurs before this form.

See also

FormDesc on page 246

Name Functions on page 74

FRENCHNUMTEXT

This function is a French version of the NumText function. The NumText function provides written numeric equivalents, such as *One Hundred and Twenty* for *120*. The FrenchNumText function serves the same purpose, but its output is in French.

Syntax

```
FrenchNumText (Number, DollarWord, CentWord, Decimode)
```

Parameter	Description
Number	Enter a valid amount. The default is the current field value.
DollarWord	Enter the word you want to use instead of *dollars*. The default is: `"dollars et"`
CentWord	Enter the word you want to use instead of *cents*. The default is: `"cents"`
Decimode	Choose from these options: 1 - numeric decimal amount 2 - spell decimal amount 3 - suppress zero, numeric decimal amount 4 - suppress zero, spell decimal amount The default is one (1).

Example

Please note the system returns only lowercase letters. For instance, if you entered 2000000, the system would return:

```
deux millions de dollars et 0 cents
```

(Assume the current field value is 2,000,000.)

Function	Result	Explanation
Return(French NumText ( ))	deux millions de dollars et 0 cents	The current field value is returned in a written form using *dollars et* and *cents*. The zero (0) is displayed in a numeric decimal amount format.
Return(French NumText (123.45,,,2))	cent vingt-trois dollars et quarante-cing cents	The written equivalent for 123.45 is displayed using Decimode *2* with the decimal spelled out.

See also

String Functions on page 78

NumText on page 327

FULLFILENAME

Use this function to make the full file name.

Syntax

```
FullFileName (Drive, Path, Name, Ext)
```

Parameter	Description
Drive	Enter the drive letter, followed by a colon.
Path	Enter the full path.
Name	Enter the file name, omitting the extension.
Ext	Enter the file extension.

The system accepts a string containing the drive, path, name, and extension components of a fully qualified file name, assembles them, and returns a string that contains the full file name.

Here is an example:

```
MYFILENAME = FullFileName("d:","\mypath\","myfile",".ext")
```

In this example, MYFILENAME would contain:

"d:\mypath\myfile.ext"

NOTE: If, in this example, *mypath* had no trailing slash, the FullFileName function would have added it for you.

Here is a z/OS example:

```
FullFileName(,"DD:DEFLIB()","MEMBER")
```

In this example, the result would be:

DD:DEFLIB(MEMBER)

See also

File and Path Functions on page 68

FileDrive on page 240

FileExt on page 241

FileName on page 242

FilePath on page 243

GetAddresseeValues

Use this function to get the mapped values of addressee record members, as defined in the Extract Dictionary (XDD) Addressee record. You can use these values to update documents or mailer pages at print time. Use this function with the AddresseeMap capability and the AddresseeCount function.

Syntax

```
GetAddresseeValues(RecipName, Index, TagName)
```

Parameter	Description
RecipName	Enter the name of the recipient for whom you want to know the values of the corresponding addressee record. The name you enter should be a valid recipient name in the document set.
Index	(Optional) Enter the addressee index to identify the addressee you want to reference. The default is one (1).
TagName	(Optional) Enter the name of a prefix for the addressee record member names this function returns. For instance, if you enter *CST*, the addressee Name value would be referenced as *CST.Name*. The default is the RecipName. The TagName must adhere to DAL variable naming convention. If the RecipName is used and this name contains spaces, the system replaces the spaces with underscores.

When used as a function, GetAddresseeValues returns a one (1) if successful or zero (0) if the recipient or named index was not found. If it returns a zero (0), the system creates the variables with empty values.

Example Here are some examples.

Function	Result	Explanation
If GetAddresseeValues (“Memo”,2)		If the Memo recipient contains at least two addressee members, the second member is retrieved and assigned to the appropriate DAL variables. If not, the GetAddresseeValues function returns a zero (0) which indicates a false result to the If statement.
#Cnt = GetAddresseeValues (“Agent”)	0	The system tries to locate the first addressee defined for the Agent recipient. If Agent is not a valid recipient or does not have any addressees mapped, the system sets #Cnt to zero (0).
#CNT2 = GetAddresseeValues("Recipient",1, "CST");	0	This says that you want your recipient values tagged with “CST”, so you end up with variables you can reference like CST.NAME, CST.CITY, CST.STATE. Without the quotes, you are saying that whatever value in CST will be the token name. So, you either get an error because this is an undefined variable, or if the variable happens to contain an invalid value, you would get the 0 return value.

Function	Result	Explanation
XYZ=”ABC” #CNT2 = GetAddresseeValues ("Recipient",1, XYZ);		This shows how a variable can define the text to tag to the recipient variables.
CST=”” #CNT2 = GetAddresseeValues ("Recipient",1, CST);	0	If you pass in an empty string as the tag, it will revert to using the recipient name as the tag — just as if you did not specify the parameter.

Here is another example:

```
If  GetAddresseeValues( “Customer”, 2)
     Return ( Customer.Name )
END
Return (“Not Provided”);
```

This example script locates the Customer recipient’s second addressee and assigns the appropriate values to the DAL variables. If the recipient index is found, a non-zero (true) result causes the If statement to proceed with the subsequent statements and return the name of the addressee member.

If the recipient or addressee index is not located, the DAL variable Customer.Name will be empty. In this example, the script does not use the empty value, but instead returns the following text:

```
Not Provided
```

See also

AddresseeCount on page 128

Have Functions on page 69

GETATTACHVAR

Use this function to return the string value of an attachment variable. You can use this function when creating print comments using Documaker Bridge.

Syntax

```
GetAttachVar (Name, DSIqueue)
```

Parameter	Description
Name	Enter the name of the attachment variable.
DSIqueue	(Optional) Enter one (1) for input or two (2) for output. The default is one (1).

See also

AddAttachVAR on page 114

RemoveAttachVAR on page 358

Docupresentment Functions on page 60

GetData

Use this function to retrieve data from a flat file extract file.

NOTE: The SrchData function, released in version 11.1 and included in version 11.0, patch 32, lets you include spaces in the search criteria, whereas the GetData function does not. Here is an example:

```
SrchData("11,HEADERREC,21(A,B, ,D)", 40, 20)
SrchData("'!/XML/Form[@form="PP 03 02"]/@form", 1,10)
```

Note the space between *A,B, ,D* and *PP 03 02*. The ability to include spaces in search criteria is important when you are using XML XPaths.

Use this function during Documaker Server processing, after the extract file has been loaded — after the LoadExtractData rule has been run.

Syntax

```
GetData (SearchMask, Occurence)
```

Parameter	Description
SearchMask	Enter the criteria that defines what data you want the system to look for. Format the search mask as shown here: `"extract search mask offset, length"`
Occurrence	This parameter lets you specify which occurrence of the data to get. The default is the first occurrence.

The system returns the data from the extract file based on the search mask.

Example

Here is an example:

```
GetData( "11,HEADEREC 40,17")
```

In this example, the GetData function finds the extract record designated by "11,HEADEREC" and returns the data at offset 40 for a length of 17. The GetData function does not format the data.

You can use an occurrence variable to get the Nth iteration of the data. Enter zero (0) to return the first record, one (1) to return the second, and so on. Here is an example:

```
GetData("11,NAMEREC 40,17", 2);
```

This example finds the 3rd record designated by "11,NAMEREC" and returns the data from offset 40 for a length of 17.

Here is an example that gets data from an XML extract file:

```
value = Trim (GetData ("!Diamond/Data/Client/Accounts/Account/
Policies/Policy/PolicyImages/PolicyImage/premium_fullterm 1,7") );
If Trim (GetData ("!Diamond/Data/Client/Accounts/Account/Policies/
Policy/PolicyImages/PolicyImage/premium_fullterm 1,7") ) = "2549"
Then;
Return ("equal -  " & GetData ("!/descendant::Personalauto/
child::Vehicles/child::Vehicle[**vehovfsym**]/vehicle_num 1,2")
Else Return ("not equal - " & value)
End;
```

In this example, the GetData function checks to see if the specified XML extract record equals 2549, if it does, the function returns the string: *equal* - concatenated with the value from another XML extract record. If not, it returns the string: *not equal* - concatenated to a value from a different XML extract record.

See also SrchData on page 395

Documaker Server Functions on page 58

GetFormAttrib

Use this function to return the content of the named user attribute (metadata) for the form you specify.

Syntax

```
GetFormAttrib (Name, Form, Group)
```

Parameter	Description
Name	Enter the name of the user attributes (metadata) to retrieve.
Form	Enter the name of a form from which to retrieve data. The default is the current form.
Group	Enter the name of the group that contains the specified form. The default is the current group.

If you omit both the Form and Group parameters, the system chooses the current form, based on where the script executes. During Entry (via the Workstation or the plug-in) this will be the form that contains the DAL script. During Documaker Server processing, the first logical form found within the document set is the current form, unless the script is executed from a section or field rule.

If you include the Form parameter, but omit the Group parameter, the system looks for the form within the current group of forms, as defined by where the script executes. During Entry (via the Workstation or the WIP Edit plug-in) this is the group that contains the form where the script executes. During Documaker Server processing, the first logical group found within the document set is the current group, unless the script is executed from a section or field rule.

If you omit the Form parameter but include the Group parameter, the system locates the first form within the group you specified.

If you define an attribute, form, or group that is not included in the current document, the system returns an empty string.

Example

For the following examples assume that form 1111 has the following metadata. Also assume form 9999 was not selected or triggered.

Name	Value
Offer	Good until cancelled
Codes	R4,79, ZW

Here is the first example:

```
xx = GetFormAttrib("Offer", "1111")
```

In this example the variable *xx* is set to:

```
Good until cancelled
```

Here is another example:

```
xx = GetFormAttrib("Codes", "9999")
```

In this example the variable *xx* is set to an empty string.

See also PutFormAttrib on page 347

Have Functions on page 69

GETINIBOOL

Use this function to retrieve from cache memory the Boolean value of an INI control group and option.

Syntax

```
GetINIBool (Context, Group, Option, Default)
```

Parameter	Description
Context	(Optional) A name (valid name) associated to a set of INI control groups and options that have been loaded into cache memory.
Group	Enter the group name (valid string) which contains the INI option Boolean value to retrieve.
Option	Enter the option name (valid string) which contains the INI Boolean value to retrieve. If the control group and option does not contain a Boolean value, the system returns a zero (0).
Default	(Optional) Enter the default string value to return from the function instead of the actual control group and option value.

The system returns one (1) if no error occurs, otherwise a zero (0) is returned.

If you omit the context, the function searches all INI files loaded in memory. If there are multiple control groups and options with the same name, this function returns the first INI control group and option string it finds.

If a context name is present, this function only searches for the control group and option in the set of control groups and options associated with the context name.

Example

Let's assume that an INI file, *TEST1.INI,* was loaded with the context name, *MVF*. The TEST1.INI file contains this control group and option:

```
< Control >
    LogEnabled = Yes
```

In addition, the FSIUSER.INI file contains this control group and option:

```
< Control >
    LogEnabled = No
```

Plus, the FSISYS.INI file contains this control group and option:

```
< Control >
    LogEnabled = Yes
```

Based on this scenario, this table shows and explains several possible results.

Function	Result	Explanation
bool_value = GetINIBool (,"Control", "LogEnabled");	The variable *bool_value* now contains a zero (0).	The function scanned the loaded INI control groups and options. It found the specified control group and option in the FSIUSER.INI first. The FSIUSER.INI set is searched first, followed by the FSISYS.INI set and then any other loaded sets, in order.

Function	Result	Explanation
bool_value = GetINIBool ("MVF", "Control", "LogEnabled");	The variable *bool_value* now contains a one (1).	The function scans only the control group and option set associated with the context name *MVF*.
bool_value = GetINIBool ("MVF", "Control", "LogEnabled", 1);	The variable *bool_value* now contains a one (1). If *Control* and *LogEnabled* are not found, *string_value* is set to zero (0).	The function scans only the control group and option set associated with the context name *MVF*.

See also

INI Functions on page 70

Using INI Options on page 8

GetINIString on page 259

LoadINIFile on page 300

GetINIString

Use this function to retrieve from cache memory the specified INI control group and option string.

Syntax

```
GetINIString (Context, Group, Option, Default)
```

Parameter	Description
Context	(Optional) A name (valid string) associated to a set of INI control groups and options which have been loaded into cache memory.
Group	Enter the control group name (valid string) which contains the INI option string to retrieve.
Option	Enter the option name (valid string) which contains the INI string value to retrieve. If the control group and option does not contain a string, the system returns a null value.
Default	(Optional) Enter the default string value to return from the function instead of the actual control group and option value.

The function returns one (1) if no error occurs, otherwise a zero (0) is returned.

If you omit the context, the function searches all INI files loaded in memory. If there are multiple control groups and options with the same name, this function returns the first INI control group and option string it finds.

If a context name is present, this function only searches for the control group and option in the set of control groups and options associated with the context name.

Example

Assume an INI file (TEST1.INI*)* was loaded with the context name, *MVV* The TEST1.INI file contains this control group and option:

```
< Control >
   Title = MVV's group/option
```

In addition, the FSIUSER.INI file contains this control group and option:

```
< Control >
   Title = Test group 1
```

Plus, the FSISYS.INI file contains this control group and option:

```
< Control >
   Title = FAP entry 1
```

Based on this scenario, the following table shows and explains several possible results.

Function	Result	Explanation
string_value = GetINIString (,"Control", "Title");	The variable *string_value* now contains this string: *Test group 1*	The function scanned the loaded INI control groups and options. It found the specified control group and option in the FSIUSER.INI first. The FSIUSER.INI set is searched first, followed by the FSISYS.INI set and then any other loaded sets, in order.
string_value = GetINIString ("MVF", "Control", "Title");	The variable *string_value* now contains this string: *MVF's group/option*	The function scans only the control group and option set associated with the context name *MVF*.
string_value = GetINIString ("MVF", "Control", "Title", "Bob's group/option");	The variable *string_value* now contains this string: *MVF's group/option* If *Control* and *Title* are not found, *string_value* is set to: *Bob's group/option*	The function scans only the control group and option set associated with the context name *MVF*.

See also

INI Functions on page 70

Using INI Options on page 8

GetINIBool on page 257

GetListElem

Use this XML function to retrieve list elements.

Syntax

```
GetListElem (%xXMLTree, SrchCriteria)
```

Parameter	Description
%xXMLTree	Enter a list type DAL variable that passes the XML tree handle.
SrchCriteria	Enter a string type DAL variable that passes the search criteria. The search criteria can be a node name, followed by up to five pairs of attribute names and values.

If successful, the system returns a text string which contains the first element that matches the search criteria.

Example

This example returns the text of the first matched element node *Form* with the attribute name *ID* and value *Agent*.

```
%xXMLTree=LoadXMLList("test.xml");
aStr= GetListElem(%xXMLTree, "Form", "ID", "Agent");
return(aStr);
```

See also

XML Functions on page 89

GetOvFlwSym

Use this function to retrieve the value stored in an overflow symbol. This is value that would be used during the next Documaker Server record overflow operation.

Syntax

```
GetOvFlwSym (Form, Symbol)
```

Parameter	Description
Form	Enter the name of the form that contains the fields on which overflow processing will occur.
Symbol	Enter the name you want to use as the overflow symbol.

The system returns the value contained in the specified overflow symbol.

Example

Here is an example:

```
#content = GetOvFlwSym ("CP0101NL", "Loc_Cnt")
```

In this example, the DAL integer variable, *#content*, would be set to the value of the overflow symbol, *Loc_Cnt*.

See also

AddOvFlwSym on page 127

IncOvFlwSym on page 280

ResetOvFlwSym on page 361

Documaker Server Functions on page 58

GetValue

Use this function to return a string that contains the contents of the DAL symbolic variable specified by the parameter. You can use this function when the name of the DAL variable is also stored in a variable, such as when a variable has to be addressed in another external script.

Syntax

```
GetValue (Symbol)
```

Parameter	Description
Symbol	Enter a string that specifies the name of a DAL symbolic variable. This can be from an expression or from another string variable.

NOTE: You will get a syntax error if you omit the Symbol parameter or if the DAL symbolic variable does not exist. It is wise to use this function with the Exists function.

Example

Here are some examples. Assume the...

- String variable 'my_variable' contains: *"Hello World"*
- Numeric variable '#_veh' contains: *20*
- String variables 'tbl_1', 'tbl_2', 'tbl_3', and 'tbl_4' respectively contain: '*Ford*', '*Chev*', '*Olds*', and '*VW*'.

In this example, the variable named *contents* is set to the string "*Hello World*":

```
variable_name = "my_variable"
contents = GetValue(variable_name)
```

This example stores the value, *20*, in the field entitled '*total # of vehicles*' in the current section:

```
SetFld ( GetValue("#_veh"), "total # of vehicles")
```

In this example, if *#line* is set to 3, the string '*Olds*' is returned. If *#line* is set to 5, a 'blank' is returned.

```
If Exists("tbl_" & #line) Then
        Return ( GetVaule("tbl_" & #line) )
    Else
        Return (" ")
End
```

See also

Exists on page 230

Miscellaneous Functions on page 73

GroupName

Use this function to get the name from a group of forms.

Syntax

```
GroupName (Count, StartGroup)
```

Parameter	Description
Count	An index reference to locate a group before or after the specified group. Enter a negative number to move backwards. The default is zero (0).
StartGroup	Enter the name of a group from which to start the search. The default is the current group.

The system returns the name of the group it located.

Keep in mind you can use an asterisk (*) as the object name to match parent objects. This lets you find objects without explicitly knowing the parent names.

Note: For optimal performance, avoid using wildcards (*) when searching for field, section, or form names.

The system returns the name of a group of forms relative to another group. If you omit the parameters, the system returns the name of the current group.

The count parameter tells the system to move forward or backwards from a located group before returning the group name.

If it cannot find the starting group cannot or the count parameter causes it to move beyond the number of groups contained in the document set, the system returns an empty string.

Groups are unique within a document set.

Example

Here are some examples:

(Assume the current group is *GROUPONE*.)

Function	Result	Explanation
GroupName( )	*GROUPONE*	No parameters will result in returning the current group name.
GroupName(-1)		Returns the name of the group before the current group.

See also

Name Functions on page 74

GVM

Use this function to retrieve the contents of a GVM variable.

Syntax

```
GVM (Name, Instance)
```

Parameter	Description
Name	Enter the name of the GVM variable.
Instance	Enter the instance number of the GVM variable. The default is one (1).

The system returns the content of the variable if it exists or a blank string if it does not.

Example

Here is an example:

Function	Result	Explanation
If (HaveGVM('Company')) AddComment(GVM('Company')) End	String or a blank string	Return the content of the GVM variable "company" if it exist.

NOTE: If the GVM variable does not exist, you will receive the error message: DM12041.

See also

Documaker Server Functions on page 58

HaveGVM on page 270

AddComment on page 117

DAL Script Examples on page 35

SetGVM on page 381

HAVEFIELD

Use this function to determine if a specified field can be located.

Syntax

```
HaveField (Field, Section, Form, Group)
```

Parameter	Description
Field	Enter the name of a field. The default is the current field.
Section	Enter the name of a section that contains the field named. The default is the current section.
Form	Enter the name of a form that contains the section and/or field named. The default is the current form.
Group	Enter the name of the form group that contains the form, section, and/or field named. The default is the current group.

The system optionally returns one (1) on success or zero (0) on failure.

Keep in mind you can use an asterisk (*) as the object name to match parent objects. This lets you find objects without explicitly knowing the parent names. For instance, if you enter

```
HaveField("FIELD", , "*" )
```

The system will find the field named *FIELD* on any form within the current group. This works because the asterisk in the form name position indicates that any form will do.

NOTE: For optimal performance, avoid using wildcards (*) when searching for field, section, or form names.

The system searches for the specified field on a particular section, form, and/or group. If the field is located, one (1) is returned. Otherwise, zero (0) is returned.

Although the return value from some of the other field's functions might be used to determine the availability of a certain field, this function merely locates the field and does not change or query any particular information about the field.

Example

Here are some examples:

Function	Result	Explanation
Return(HaveField( ))	1	If this script is associated with an entry field, it will always return one (1) if no parameters are provided.
Return(HaveField("Second"))	1 or 0	the current section will be searched for the field. A one (1) is returned if located.
Return(HaveField("Third", , "FRM"))	1 or 0	Locate the form specified within the current form group. Then locate Third anywhere on that form. If found, a one (1) is returned.

See also Have Functions on page 69

Field Formats on page 62

Locating Fields on page 64

HaveForm

Use this function to determine if a given form is contained in the document.

Syntax

```
HaveForm (Form, Group)
```

Parameter	Description
Form	Enter the name of a form. The default is the current form.
Group	Enter the name of a group to contain the specified form. The default is the current group.

The system optionally returns one (1) if the form is located or zero (0) if it cannot be found.

Keep in mind you can use an asterisk (*) as the object name to match parent objects. This lets you find objects without explicitly knowing the parent names.

NOTE: For optimal performance, avoid using wildcards (*) when searching for field, section, or form names.

Several of the DAL functions might return a value that may indicate a form is or is not a part of the document. However, those functions also intend to perform some other procedure other than searching for the form. This function simply identifies whether a given form is present in the form set.

The function does not require any parameters. However, calling it in this manner will typically return 1, since it will locate the current form.

Example

Here are some examples:

Function	Result	Explanation
HaveForm(“Form”)	1 or 0	Attempts to locate the named form. If found, returns 1.
HaveForm(“Form\3”, “GRP”)	1 or 0	Locates the third occurrence of the file named Form within the specified group. If found, returns 1.

See also

Have Functions on page 69

HAVEGROUP

Use this function to determine if a given group is part of a document.

Syntax

```
HaveGroup (Group)
```

Parameter	Description
Group	Enter the name of a group to locate. The default is the current group.

The system returns one (1) if the group is located and zero (0) if it cannot be found.

Keep in mind you can use an asterisk (*) as the object name to match parent objects. This lets you find objects without explicitly knowing the parent names.

NOTE: For optimal performance, avoid using wildcards (*) when searching for field, section, or form names.

Several DAL functions can return values that indicate a group is or is not a part of the document. However, those functions also intend to perform some other procedure other than searching for the group. The HaveGroup function simply identifies whether a given group is present in the document.

The function does not require any parameters. However, calling it in this manner will typically return 1, since it will locate the current group.

Example

Here is an example:

Function	Result	Explanation
HaveGroup (“GRP”)	1 or 0	Returns one (1) if the identified group is a part of the document.

See also Have Functions on page 69

HaveGVM

Use this function to determine if a GVM variable exists.

Syntax

```
HaveGVM (Name, Instance)
```

Parameter	Description
Name	Enter the name of the GVM variable.
Instance	Enter the instance number of the GVM variable. The default is one (1).

The system returns one (1) if it locates the GVM variable or a zero (0) if it cannot find the variable.

Example

Here is an example:

Function	Result	Explanation
If (HaveGVM('Company')) AddComment(GVM('Company')) End	1 or 0	If a GVM variable "company" exist; then add the content of the GVM variable to the print stream.

See also

Documaker Server Functions on page 58

GVM on page 265

AddComment on page 117

DAL Script Examples on page 35

SetGVM on page 381

GVM on page 265

HaveImage

Use this function to determine if a given section is contained in the document.

Syntax

```
HaveImage (Section, Form, Group)
```

Parameter	Description
Section	Enter the name of a section to locate. The default is the current section.
Form	Enter the name of an form that is assumed to contain the specified section. The default is the current form.
Group	Enter the name of a group to contain the specified section or form. The default is the current group.

The system returns one (1) if the form is located and zero (0) if it cannot be found.

Keep in mind you can use an asterisk (*) as the object name to match parent objects. This lets you find objects without explicitly knowing the parent names.

NOTE: For optimal performance, avoid using wildcards (*) when searching for field, section, or form names.

Several of the DAL functions might return a value that may indicate a section is or is not a part of the document. However, those functions also intend to perform some other procedure beyond searching for the section. This function simply identifies whether a given section is present as part of a form and/or group.

The function does not require any parameters. However, calling it in this manner will typically return 1, since it will locate the current section.

Example

Here are some examples:

Function	Result	Explanation
HaveImage("IMG")	1 or 0	Attempts to locate the named section on the current form. If found, return 1.
HaveImage("IMG\2", "Form\3", "GRP")	1 or 0	Locate the third occurrence of Form within the specified group. If found, then locate the second occurrence of IMG. If successful, return 1.

See also

Have Functions on page 69

Where DAL Functions are Used on page 97

HAVELOGO

Use this function to determine if a graphic (LOG) exists on a section or form which is in the current form set.

Syntax

```
HaveLogo (Graphic, Section, Form, Group)
```

Parameter	Description
Graphic	Enter the name of the graphic you want to find. Graphic names are assigned in Studio or Image Editor.
Section	Enter the name of a section that contains the graphic you specified. The default is the current section.
Form	Enter the name of a form that contains the section you specified. The default is the current form.
Group	Enter the name of a group to use to locate the graphic. The default is the current group.

The system returns one (1) if it finds the graphic and zero (0) if it does not.

Keep in mind you can use an asterisk (*) as the object name to match parent objects. This lets you find objects without explicitly knowing the parent names.

NOTE: For optimal performance, avoid using wildcards (*) when searching for field, section, or form names.

Example

Here are some examples:

Function	Result	Explanation
HaveLogo(“Log1”)	1 or 0	Determines if Log1 exists on the current section, form, group.
HaveLogo(“Log1”, “IMH1\3”,”UpRate”)	1 or 0	Determines if Log1 exists on the 3rd occurrence of the section, IMH1, on the form, UpRate, within the default group.

See also

ChangeLogo on page 163

DelLogo on page 214

HaveField on page 266

HaveForm on page 268

HaveGroup on page 269

HaveImage on page 271

InlineLogo on page 282

Logo on page 303

RenameLogo on page 359

Have Functions on page 69

HaveRecip

Use this function to see if the specified recipient name is defined in the form set for the specified section, form, or group.

You can use this function along with the RecipientName function in DAL scripts to place a sequence number on each page of each recipient batch.

Syntax

```
HaveRecip (Recipient, Section, Form, Group)
```

Parameter	Description
Recipient	Enter the name of a recipient.
Section	Enter the name of a section that contains the field named. The default is the current section.
Form	Enter the name of a form that contains the section and/or field named. The default is the current form.
Group	Enter the name of the form group that contains the form, section, or field. The default is the current group.

The system returns one (1) if true or zero (0) if false.

NOTE: You must enter a recipient name.

See also

RecipientName on page 355

Have Functions on page 69

Hex2Dec

Use this function to return the integer equivalent of a hexadecimal string.

Syntax

```
Hex2Dec (Value1)
```

Parameter	Description
Value1	This parameter specifies a string of characters you want converted into an integer value. If the string value does not represent a valid hexadecimal number, the results are questionable and can result in only part of the value being converted.

The largest hexadecimal value supported is FFFFFFFF. Keep in mind, however, that hexadecimal values are considered *unsigned* while integer values can be both positive and negative.

The largest integer value 2,147,483,647 is 7FFFFFFF when represented using hexadecimal. HEX values greater than 80000000 represent negative integer values. Hex value FFFFFFFF represents the integer value -1.

Example

Here is an example:

```
y = "1A2B"
z = Hex2Dec(y)
Result is z = 6699

y = "FF00"
z = Hex2Dec(y)
Result is z = 65280
```

See also

Dec2Hex on page 206

Bit/Binary Functions on page 42

Hour

Use this function to extract the number of hours from a time.

Syntax

```
Hour (Time1, Format1)
```

Parameter	Description
Time1	Enter a valid time string. Assumed to be in the format specified by the next parameter. The default is the current time.
Format1	Enter a valid time format string. Describes the first parameter (time1). The default is time format 1 (HH:MM:SS).

Example

Here are some examples:

(Assume the current time is 03:05:09 pm.)

Function	Result	Explanation
Return(Hour( ))	3	Defaults to the current time and extracts 3.
Return(Hour (“9:50:20AM”, 2))	9	Reads the given time which is in format 2 and extracts 9.

See also Time Formats on page 80

ImageName

Use this function to get the name of a section. This name is returned.

Syntax

```
ImageName (Count, Startimage, Form, Group)
```

Parameter	Description
Count	Enter an index reference to locate a form before or after the specified form. The default is zero (0).
Startimage	Enter the name of a section from which to begin the search. The default is the current section.
Form	Enter the name of a form containing the requested section. The default is the current form.
Group	Enter the name of a group to contain the specified form.The default is the current group.

Keep in mind you can use an asterisk (*) as the object name to match parent objects. This lets you find objects without explicitly knowing the parent names.

NOTE: For optimal performance, avoid using wildcards (*) when searching for field, section, or form names.

The system returns the name of a section relative to another section on the same form. If no parameters are provided to this function, the current section's name is returned. The Count parameter tells the system to move a number of sections forwards or backwards (negative) from a located section before returning the section name.

If the starting section cannot be located or the Count parameter causes the system to move beyond the number of sections contained on the form, the system returns an empty string.

If there is more than one copy of a section on the located form, the name returned will contain the occurrence notation used by DAL functions to locate sections. For instance, a name like IMG\2 identifies the second copy of IMG on a particular form.

Example

Here are some examples:

(Assume the current section is named *IMG*.)

Function	Result	Explanation
ImageName( )	IMG	No parameters will result in returning the current section name.
ImageName(2, "IMG", "FormC")		Locate FORMC in the current group. Next, locate IMG on that form. Then, return the name of the section two positions beyond the located section.

See also

Name Functions on page 74

ImageRect

Use this procedure/function to retrieve the rectangular coordinates of a section in a form set (document).

Syntax

```
ImageRect (PrefixVariable, Section, Form, Group)
```

Parameter	Description
PrefixVariable	Enter the coordinates for the section.
Section	Enter the name of a section in the form set. The default is the current section.
Form	Enter the name of the form that contains the section. The default is the current form.
Group	Enter the name of the form group that contains the form and section. The default is the current form group.

This procedure gets the coordinates for the section and stores them in the defined variable names. If the prefix name variables do not exist in DAL, the system creates them. The system creates four internal variables: *prefix name.top*, *prefix name.left*, *prefix name.bottom*, and *prefix name.right*. If these variables exist, the system modifies them with the new coordinates.

Example

For these examples, assume the prefix name is *MyImage*, the current section is *Image25*, the form is *Input_form*, and the form group is *package1*. The coordinates are:

	Image25	Image50
top	25	125
left	50	150
bottom	100	200
right	200	200

Here are some examples:

Procedure	Result	Explanation

IMAGERECT (“MyImage”)	Internal variables equal:
	`MyImage.top=25`
	`MyImage.left=50`
	`MyImage.bottom=100`
	`MyImage.right=200`
		The procedure returns the coordinates for the current section (*Image25)* on the current form in the current form group
		If it does not exist, the procedure returns zero (0).

Procedure	Result	Explanation
IMAGERECT (“MyImage”, “Image50”)	Internal variables equal: `MyImage.top=125` `MyImage.left=150` `MyImage.bottom=200` `MyImage.right=200`	The procedure returns the coordinates for *Image50* on the current form in the current form group. If it does not exist, the procedure returns zero (0).
IMAGERECT (“m”, “MVF\2”, “XYZ”)	Internal variables equal: `m.top = 75` `m.left = 125` `m.bottom = 300` `m.right = 225`	Gets and stores the coordinates for the second occurrence of the section *MVF* on the form *XYZ* into the DAL target variables. If it does not exist, the procedure returns zero (0).

See also

Section Functions on page 77

SetImagePos on page 382

IncOvFlwSym

Use this procedure/function to increment an overflow symbol. This procedure provides DAL with the Documaker Server equivalent to the IncOvFlwSym rule, with the exception that it will only increment by one.

Syntax

```
IncOvFlwSym (Form, Symbol)
```

Parameter	Description
Form	Enter the name of the form that contains the fields on which overflow processing will occur.
Symbol	Enter the name you want to use as the overflow symbol.

The system optionally returns one (1) on success or zero (0) on failure.

This procedure increments the value contained in the specified overflow symbol.

Example

Here is an example:

```
rc = IncOvFlwSym ("CP0101NL", "Loc_Cnt")
```

In this example, the overflow symbol, *Loc_Cnt* is incremented and the DAL integer variable, # *rc*, is set to one (1) on success or zero (0) on failure.

Syntax

AddOvFlwSym on page 127

GetOvFlwSym on page 262

ResetOvFlwSym on page 361

Documaker Server Functions on page 58

INI

Use this function to get the value of an INI option from the currently loaded INI files.

If there is more than one occurrence of a control group and option in the various INI files the system uses, like the FSIUSER.INI and the FSISYS.INI files, this function uses the values in the first control group and option it finds that matches the criteria you enter. The system usually first loads the FSIUSER.INI file, which tells it to then load the FSISYS.INI file.

Syntax

```
INI (Group, Option, Default)
```

Parameter	Description
Group	Enter the name of the INI control group name (valid string) which contains the INI option string you want to retrieve.
Option	Enter the name of the INI option (valid string) which contains the INI string value you want to retrieve. If the control group and option do not contain a string, the system returns a null value.
Default	(Optional) The default string value to return from the function instead of the actual control group and option value.

The system retrieves the specified control group and option string. The system returns one (1) if no errors occur and zero (0) if errors occur.

Example

This example:

```
INI("UserInfo","File")
```

retrieves the name of the user information file, as stored in this control group:

```
< UserInfo >
   File =
```

See also

INI Functions on page 70

Using INI Options on page 8

GetINIBool on page 257

GetINIString on page 259

INLINELOGO

Use this procedure/function to cause a graphic (LOG) to be *in-lined* in the print stream. This means you do not have to store the graphic as a printer resource on the printer.

Syntax

```
InlineLogo (Graphic, Option, Section, Form, Group)
```

Parameter	Description
Graphic	Enter the name of the graphic to be in-lined in the print stream. Graphic names are assigned in Studio or Image Editor.
Option	This parameter sets the inline flag. You can choose from these options: One (1) equals On Zero (0) equals Off The default is one (1).
Section	Enter the name of a section that contains the specified graphic. The default is the current section.
Form	Enter the name of a form that contains the section. The default is the current form.
Group	Enter the name of a group to use to locate the specified object. The default is the current group.

The system optionally returns one (1) on success or zero (0) on failure.

Example

Here are some examples:

Procedure	Result	Explanation
InlineLogo(Log1”)	1 or 0	In-lines Log1 (on the current section, form, and group) into the print stream.
InlineLogo(“Log1”, 1,”IMH1\3”,”UpRate”)	1 or 0	In-lines Log1 (on the 3rd occurrence of the named section, IMH1, on the form, UpRate) into the print stream.

See also

ChangeLogo on page 163

DelLogo on page 214

HaveLogo on page 272

Logo on page 303

RenameLogo on page 359

Graphics Functions on page 71

INPUT

Use this function to create a window with a title and a prompt which asks the user to enter information.

Syntax

```
Input (Prompt, Title, Length, DefText)
```

Parameter	Description
Prompt	Enter a text string to assign as the prompt for the field. The default is Text.
Title	Enter a text string to assign as the title of the window. The default is Title.
Length	Enter the maximum input text length. The default is set by Windows.
DefText	Enter a text string to assign as the default input data.

The system returns the input results.

This function creates a window you can use to gather information from a user. The text entered through the window is returned as a string. If no text is assigned, or if the user closes the window without choosing Ok, the returned string will be empty.

Example

Here are some examples:

Function	Result	Explanation
NAME = Input (“Please enter your name:”, “Name Entry”); Return(Name)	Produces a window requesting input.	The name of the window is *Name Entry*. The user sees the prompt *Please enter your name:* If the user selects Cancel, NAME is an empty string. If the user selects Ok, NAME contains the text entered by the user.
Return(Input( ))	Produces a window requesting input.	This window will not have a title or a prompt. The user is merely presented with an input field into which data should be entered.
Return(Input (“Confirm this result”, , 30, “123.45”))	Produces a window requesting input.	This window will have the prompt *Confirm this result*. The input field accepts up to 30 characters and defaults to “123.45”. There will be no title.

See also Documaker Workstation Functions on page 59

INSERT

Use this function to insert a substring into a string at the position you specify. The result string is returned.

Syntax

```
Insert (String, Position, SubString)
```

Parameter	Description
String	Enter a valid string. The default is the value of the current field.
Position	Enter the position in the field to perform the insert. The default is the one (1), the first position.
SubString	Enter the string that you want to insert.

The system adds the substring to the string you specified in the first parameter at the indicated position. If the position indicated in the second parameter is greater than the length of the original string, the string is increased to the given length before the third parameter is inserted.

If no position is given in the second parameter the insertion begins at position one. If no value is provided for the third parameter (Substring), nothing is inserted.

Example

Here are some examples:

(Assume the current field contains the text *Your Name*.)

Function	Result	Explanation
Return(Insert (, , “Type ”))	Type Your Name	Defaults to the first position of the current field and inserts *Type*.
Return(Insert (, CFIND (, “ “), “ First”))	Your First Name	First the CFind function locates a blank space at position 5 within *Your Name*. Next, *First* is inserted at position 5.
Return(Insert (“Complete blank.”, 10, “every “))	Complete every blank.	Goes to position 10 and inserts *every*.
Return(Insert (“Complete blanks”, 17, “with black ink.”))	Complete blankswith black ink.	Increases the length of the field to 17 and appends *with black ink*.

See also

String Functions on page 78

CFind on page 162

INT

Use this function to return the integer portion of a number.

Syntax

```
INT (Number)
```

Parameter	Description
Number	Enter a valid numeric data type. The default is the integer value of the current field

The system returns the integer value of a number.

The decimal portion of the number is truncated. The number is not rounded up or down. The sign of the number is not changed.

Example

Here are some examples:

Function	Result	Explanation
INT(-101.99)	-101	Defaults to the current field.
$TEMP = 99.99 #RESULT = INT($TEMP)	99	After executing these statements, *$TEMP* will be 99.99 and *#RESULT* will be 99, without a decimal.
#RESULT = INT(10/4)	2	The parameter value will equate to 2.5 The INT function will truncate this result to 2. The function does not round.

See also

Mathematical Functions on page 72

IsPrintObject

Use this function during banner processing or in another print operation to determine if the section (image), form, or group is printable. This determination is based on the current print recipient and the recipient copy count.

Syntax

```
IsPrintObject (Section, Form, Group)
```

Parameter	Description
Section	Enter the name of the section you want to check. If you omit this parameter, the system uses the current section.
Form	Enter the name of the form you want to check. If you omit this parameter, the system uses the current form.
Group	Enter the name of the group you want to check. If you omit this parameter, the system uses the current group.

NOTE: You can use this function outside of a print operation to determine if a section is printable, but a true (1) result is not a guarantee the section will print during the next print operation.

Example

Here is an example:

```
IsPrintObject();
```

This example checks the current section on the current form in the current group and returns a one (1) if that section is printable or a zero (0) if it is not.

See also

Printer and Recipient Functions on page 76

IsXMLError

Use this function to check the list for error status.

Syntax

```
IsXMLError (%xXMLTree, SrchCriteria)
```

Parameter	Description
%xXMLTree	Enter a list type DAL variable that passes the XML tree handle.
SrchCriteria	Enter a string type DAL variable that passes the search criteria. The search criteria can be a node name, followed by up to five pairs of attribute names and values.

The system returns one (1) if no errors occur or zero (0) if errors occur.

See also XML Functions on page 89

JCenter

Use this function to center text within a specified length and return the result.

NOTE: To justify a display item, such as a field, on a fixed point use the JustField function. The JCenter function is for padding a text string so it will appear centered within a given string length.

Syntax

```
JCenter (String, Length)
```

Parameter	Description
String	Enter a valid string. The default is the value of the current field.
Length	Specify the desired length of output. The default is the length of the input string.

The system justifies the text characters of the string parameter within the specified length and returns the new string.

If the length specified in the Length parameter is longer than the string, the result will be increased to the given length before the system centers the string. If the length specified is less than the string, the length of the string is used.

For example, if the variable field has a length of 30, the DAL script says Return(JCenter (,10)), and you enter *ABC* in the variable field, the system will center ABC using a length of 10 instead of 30.

Example

Here are some examples:

(Assume the current field contains the text *Name* and can be up to 20 characters.)

Function	Result	Explanation
JCenter (, Size ( ))	" Name "	First the Size function determines that the maximum length of the field is 20. Then the JCenter function defaults to the current field and centers the text name within the given size of 20.
JCenter ("Complete blanks.", 5)	Complete blanks.	Ignores the specified length (5) because it is less than the given string.
JCenter ("Complete blanks.", 25)	" Complete blank. "	Increases the size of the input string to 25 and centers the text. The variable field length is not affected, so the text appears to be off center.

See also

JustField on page 291

String Functions on page 78

Size on page 390

JLEFT

Use this function to left justify text within a specified length and return the result.

Syntax

```
JLeft (String, Length)
```

Parameter	Description
String	Enter a valid string. The default is the value of the current field.
Length	Specify the desired length of output. The default is the length of the input string.

The system left justifies the text characters of the string parameter within the specified length and returns the new string.

If the length specified in the length parameter is longer than the string, the result will be increased to the given length before the justification. If the length specified is less than the string, the length of the string is used.

Example

Here are some examples:

(Assume the current field contains the text *Name* and can be up to 20 characters.)

Function	Result	Explanation
JLeft ("Heading", 20)	"Heading "	Left justifies the text within a length of 20 spaces.
JLeft (" Complete blanks. ", 5)	"Complete blanks. "	Ignores the specified length (5) because it is less than the given string.
JLeft (, Size ( ) & "X")	"Name X"	First the Size function determines that the maximum length of the field is 20. Then *X* is added to the end of the field. There are 15 spaces between the end of the word *Name* and the *X*.

See also

String Functions on page 78

@ on page 109

Size on page 390

JRight

Use this function to right justify text within a specified length and return the result.

Syntax

```
JRight (String, Length)
```

Parameter	Description
String	Enter a valid string. The default is the value of the current field.
Length	Specify the desired length of output. The default is the length of the input string.

The system justifies the text characters of the string parameter within the specified length and returns the new string.

If the length you specify in the Length parameter is longer than the string, the result is increased to the given length before the text is justified.

If the length specified is less than the string, the system uses the length of the string.

Example

Here are some examples:

(Assume the current field contains the text *Name* and can be up to 20 characters.)

Function	Result	Explanation
JRight ("Heading", 20)	" Heading"	Increases the size of the field to 20 and right justifies the text.
JRight (" Complete blanks. ", 5)	" Complete blanks. "	Ignores the specified length (5) because it is less than the given string.
JRight (, SIZE ( ) & "!")	"Name!"	First the Size function determines that the maximum length of the field is 20. Then the original text in the field is right justified and an exclamation point (!) is concatenated after *Name*.

NOTE: If you are aligning decimal numbers, be sure to use a fixed or non-proportional font, such as Courier.

See also

String Functions on page 78

@ on page 109

Size on page 390

JUSTFIELD

Use this procedure/function to justify (left, right, or center) a variable field content by modifying its field coordinates.

NOTE: To pad a text string so it will appear centered within a given string length, use the JCenter function. The JustField function is for justifying display items, such as fields, on a fixed point.

Syntax

```
JustField (Mode, Xcoordinate, Justification, Field, Section, Form,
Group)
```

Parameters	Description
Mode	Enter L (left), R (right), or C (center). The default is L.
Xcoordinate	Enter the X coordinate used to align the field. If Mode is R, this will be zero (0), the right-most position of the field. If Mode is C, this will be the center of the field. Here is an example: `"R", 5000` If the data is 12345, the character 5 will be positioned at 5000 FAP units.
Justification	Enter a character found in the data to use to align the field. The procedure aligns the field so the character you specify overlays the X coordinate. You must define the X-coordinate parameter when using the justification character. If you omit the X-coordinate the system runs as if the justification character was not specified. Here is an example: R,5000,"." If the data is 123.45, then the decimal point will be positioned at 5000 FAP units.
Field	Enter the name of the field. The default is the current field.
Section	Enter the name of the section that contains the field. The default is the current section.
Form	Enter the name of the form that contains the section and/or field. The default is the current form.
Group	Enter the name of the group that contains the form, section, and/or field. The default is the current group.

Example

This example centers the original address lines data in the section, QJUSTFIELD2, at 10,000 FAP units.

```
JustField("C",10000, ,"line 1", , "qjustfield2")
JustField("C",10000, ,"line 2", , "qjustfield2")
JustField("C",10000, ,"line 3", , "qjustfield2")
```

Here is an example:

line 1	Oracle Insurance
line 2	Atlanta, GA 30339-4000
line 3	404.439.5500

5,000 FAP units

line 1	Oracle Insurance
line 2	Atlanta, GA 30339-4000
line 3	404.439.5500

10,000 FAP units

This example justifies the original line data (left aligned at 5,000 FAP units) on the decimal point at 10,000 FAP units.

```
JustField("C",10000,".","line 1")
JustField("C",10000,".","line 2")
```

Here is an example:

line 1	5,000.00
line 2	12345.8888888

5,000 FAP units

line 1	5,000.00
line 2	12345.8888888

10,000 FAP units

See also

JCenter on page 288

Field Functions on page 61

KickToWIP

Use this function to send a transaction to WIP from the GenData program. This function lets you use DAL instead of the KickToWIP rule or the field properties Attributes required field flag.

Syntax

```
KickToWIP ( )
```

There are no parameters for this function.

Use the ShowWIPWarning option to suppress the Sent to Manual Batch warning messages:

```
< RunMode >
    ShowWIPWarning = No
```

Option	Description
ShowWIPWarning	Enter No to suppress warning messages included the error logs when using the KickToWIP DAL function. The default is Yes, which tells the system to include the messages in the error logs.

Example

Here is an example of how you would set your AFGJOB.JDT file:

```
<Base Form Set Rules>
;NoGenTrnTransactionProc;;
…
;WriteOutput;;
;WriteNaFile;;
;PostTransDAL;;KickToWIP( );
```

In this example, the PostTransDAL function sets the Manual batch flag before the NA, POL, and Receipt batch files are written. Here is an example of the section-level rules:

```
<Image Rules>
…
;PreImageDAL;;KickToWIP( )
```

In this example, the Manual batch flag is set if the section is triggered.

Here is an example of the field-level rules:

```
;0;0;area1;0;0;area1;0;0;;DAL;Call(“Chk_If_Kick”);N;N;N;N;919;6736;
12112;
```

In this example, when the Areal field is executed the system calls the DAL script named Chk_If_Kick. The DAL script checks for the presence of two conditions and if true, sets the Manual batch flag for the transaction.

Here is an example of the Chk_If_Kick DAL script:

```
BeginSub Chk_If_Kick

    If (CountRec(“1,Second_Address”) = 0) AND \

        (GetData(“1,Second_party, 45,1) = “X”) Then

            KickToWIP( )
```

```
    End

EndSub
```

NOTE: You must execute this DAL function before the ConvertWIP form set level rule is executed, if it is included in the AFGJOB.JDT file

See also Documaker Server Functions on page 58

LeapYear

Use this function to find out whether or not the specified year is a leap year.

Syntax

```
LeapYear (Year)
```

Parameter	Description
Year	Enter the year. You can enter either a two- or four-digit number. If you enter a two-digit number, the current century is added to create the year value. The default is the current year.

The system returns one (1) if the year is a leap year and zero (0) if it is not.

This function is most often used with the Year function. The Year function extracts the year number from a given date.

Example

Here are some examples:

(Assume the current date is 07/01/08.)

Function	Result	Explanation
LeapYear ( )	1	The parameter defaults to the current year (2008). Since 2008 is a leap year, one (1), which represents true, is the result.
LeapYear (07)	0	The year 2007 was not a leap year. Therefore, the result is zero (0), representing false.
LeapYear (Year ("2009/09/09", "34"))	0	First the Year function extracts the year number (2009) from the date, which is given in the date format "34". Then LeapYear determines that 2009 is not a leap year and returns zero (0.)

See also

Using INI Options on page 8

Date Formats on page 52

Year on page 448

Date Functions on page 51

LEFT

Use this function to return a specified number of left most characters.

Syntax

```
Left (String, Length)
```

Parameter	Description
String	Enter a valid string. The default is the value of the current field.
Length	Specify the desired length of output. The default is the length of the input string.

The system returns a string equivalent to the given length from the left portion of the string.

The input string is trimmed of leading and trailing spaces. If the length specified in the second parameter exceeds the length of the string, the result is increased to the given length.

Example

Here are some examples:

(Assume the current field contains the text *Your Name* and can be up to 20 characters.)

Function	Result	Explanation
Left ( )	Your Name	Defaults to the current field and returns the full length of the field.
Left (“Complete blanks.”, 5)	Compl	Default to position one (1) and returns the first five characters.
Left (“ final payment”, 13)	“final payment”	Trims the field of leading spaces and returns 13 characters.

See also

Right on page 363

String Functions on page 78

LEN

Use this function to return the length of the specified string. The length includes all characters, including leading and trailing spaces.

Syntax

```
LEN (String)
```

Parameter	Description
String	Enter a valid string. The default is the value of the current field.

This function is often confused with the Size function. The LEN function returns the length of the actual data contained in a text string, including leading and trailing spaces. The Size function returns the length of the defined data area for a section field.

Example

Here are some examples:

(Assume the current field contains the text *Your Name*.)

Function	Result	Explanation
LEN ( )	9	Defaults to the current field.
LEN (" Your Name ")	19	The result includes the leading and trailing spaces of the given field.
LEN ("Street Address")	14	Returns the length of the given string.
LEN (@("ThisField"))	8	Finds the variable field named ThisField on the current section and counts the length of the data. The data in this field is *Jane Doe*, so the number 8 is returned.

See also

String Functions on page 78

@ on page 109

Size on page 390

LISTINLIST

Use this function to search for the comma-delimited list specified by the second parameter for each character string in the comma-delimited list specified by the first parameter. If a match is found, the function returns the ordinal position (integer) of the first string in the second parameter that matches any of the strings in the first parameter. If no match is found, the function returns a zero (0).

Syntax

```
ListInList (StringList, ListString)
```

Parameter	Description
StringList	Enter the name of the list of character strings or enter the list of character strings you want to search for. Use commas to separate each character string entry you want to find. Keep in mind the system considers spaces when searching, so strings must match exactly.
ListString	Enter the name of the string list or the character string list to be searched. Use commas to separate each string entry you want to search for.

The function returns a number that indicates which string entry was found. For instance, if the third string entry was found, the function returns a three (3).

Example

Here is an example:

This function statement	Returns	Assuming
ListInList(@("e_codes"), "ABC,AB,DE,A,GFHI,ABCD")	1	Field *e_codes* contains: *ABC,A*.
ListInList(GetValue("e_codes"), "ABC,AB,DE,A,GFHI")	2	DAL variable, *e_codes*, contains: *AB,abcd*.
ListInList(?("e_codes"), "ABC,AB,DE,A,GFHI,ABCD")	3	XDB entry *e_codes* returns: *DE,a*.
ListInList(?("e_codes"), ?("t_codes"))	4	XDB entry *e_codes* returns *A*. The entry *t_codes* contains: *ABC,AB,DE,A,GFHI,ABCD*.
ListInList(?("e_codes"), "ABC,AB,DE,A,GFHI,ABCD")	0	XDB entry *e_codes* returns: *XYZ*.

If you omit the first parameter, you get the data from the current field. If you omit the second parameter, you receive this error message:

```
Wrong number of parameters
```

Here is another example. For this example assume the following parameters contain:

- GetValue(col_name1) results in the character string: AA, EE.
- DAL variable col_name1_codes contains the string: EEacb,XXEE,EE,AEEAC.
- GetValue(ca_codes) contains the string: Xxaab,YYEE, EE,AA,AeeAC.

This statement	Returns
#rc = ListInList(GetValue(col_name1), col_name1_codes)	3
#rc = ListInList(GetValue(col_name1), GetValue(col_name1_codes))	4

The return value for the above example returns a four (4) because two spaces exist between the comma and EE.

Keep in mind:

- The search is not case-sensitive. This means *A* will match *a*.
- Spaces are considered. This means the system will find no matches in the following examples:

```
ListInList("Steel,Wood", " Steel,Aluminum")
ListInList("Steel,Wood", "Steel ,Aluminum")
ListInList("Steel,Wood", "Aluminum,Steel ")
```

See also String Functions on page 78

LoadINIFile

Use this procedure/function to load an INI file into cache memory.

Syntax

```
LoadINIFile (Context, File)
```

Parameter	Description
Context	(Optional) A name (valid string) that will be associated to the set of INI control groups and options contained in the physical file.
File	Enter the name of the INI file to load. If you omit the extension, the system assumes it is *INI*. The system searches in the current directory, or uses a full path name if you specify one

This procedure returns success (1) if no error occurred during its execution, otherwise a failure (0) is returned.

If you specify a context name, that name can be used by other INI functions to reference the loaded set of INI control groups and options.

Example

Here are some examples:

Procedure	Result	Explanation
LoadINIFile (,"DALRun");	The INI control groups and options can now be referenced by executing modules.	The INI file is loaded into cache memory. Execution of this procedure assumes the file extension is *INI*.
LoadINIFile ("Run_process" , "DALRun.ini");	The INI control groups and options can now be referenced by executing modules. This set of INI control groups and options can now be referenced by other INI functions, using the tag *Run_process*.	The INI file is loaded into cache memory.

See also

INI Functions on page 70

Using INI Options on page 8

SaveINIFile on page 370

GetINIBool on page 257

GetINIString on page 259

PutINIBool on page 349

PutINIString on page 351

LoadLib

Use this procedure/function to load into cache memory a file which contains a library of DAL scripts.

Syntax

```
LoadLib (File)
```

Parameter	Description
File	Enter the name of the file which contains the DAL scripts. If you omit the path, the system looks for the file in DefLib. If you omit the extension, the system uses the one defined in the Ext option of the DAL control group in your INI file. You must include the File parameter.

This procedure loads a file which contains one or more DAL functions into cache memory. Each of these procedures and functions can be referenced as a named subroutine.

NOTE: You should only execute the LoadLib procedure once per library.

Example

Here is an example:

Procedure	Result	Explanation
LoadLib ("DB_Func")	The system loads the DB_Func file into cache memory.	Once loaded, you can reference the DAL scripts stored in memory as named subroutines.

See also

Miscellaneous Functions on page 73

Creating a DAL Script Library on page 5

LoadXMLList

Use this function to load an XML document and extract an XML tree.

Syntax

```
LoadXMLList (FileName)
```

Parameter	Description
FileName	Enter the name of the XML file you want to load.

The system returns the XML tree in a list type DAL variable.

Example

For an example, see the DAL script in Scenario 2 on page 90.

See also

XML Functions on page 89

DestroyList on page 216

Logo

Use this procedure/function to place a graphic file (LOG) at a specified position in the section.

Syntax

```
Logo (Graphic, Xcoordinate, Ycoordinate, Section, Form, Group)
```

Parameter	Description
Graphic	Enter a valid name for a graphic. Must be a variable field object.
Xcoordinate	Enter a valid X coordinate location.
Ycoordinate	Enter a valid Y coordinate location.
Section	Enter the name of the section name that contains the new graphic. The default is the current section.
Form	Enter the name of the form name that contains the section you specified. The default is the current form.
Group	Enter the name of the group that contains the specified section or form. The default is the current group.

The system optionally returns one (1) on success or zero (0) on failure.

This procedure uses FAP units (1 inch = 2400 FAP units). The top-left position of a page represents coordinate (0, 0). To place a graphic an inch from the top and an inch from the left of the page, the X and Y coordinates would be (2400, 2400).

If the location for a particular graphic can be described in relation to a field on the form, you can use the FieldX and FieldY functions to get the coordinates of that field.

This function does not redraw the section display. Use the Refresh procedure with the Logo procedure to view the changes.

Example

Here are some examples:

Procedure	Result	Explanation
Logo (“janedoe”, “7500”, “5500”);Refresh( )	1 or 0	Defaults to add the graphic on the current section at the location specified.
Logo(“Hancock”, FieldX(“MyField”), FieldY(“MyField”), “IMG”, “FORM”)Refresh( )	1 or 0	First locate the specified form in the current group. Next locate IMG on that form. Finally, add the graphic at the same location as the field, “MyField”.

See also

ChangeLogo on page 163

DelLogo on page 214

HaveLogo on page 272

InlineLogo on page 282

FieldX on page 238

FieldY on page 239

Refresh on page 357

RenameLogo on page 359

Graphics Functions on page 71

Lower

Use this function to convert all alphabetic characters to lowercase characters and return the result.

Syntax

```
Lower (String, Length)
```

Parameter	Description
String	Enter a valid string. The default is the value of the current field.
Length	Specify the desired length of output. The default is the length of the input string.

If the length you specify is longer than the string, the string is increased to the given length. If the specified length is less than the string, the length of the string is used. The string is not truncated.

Example

Here are some examples:

(Assume the current field contains the text *Your Name*.)

Function	Result	Explanation
Lower ( )	“your name”	Defaults to the current field
Lower (“Street Address”)	“street address”	Lowercases the given string
Lower (, 15)	“your name “	Lowercases the current field and increases the length to 15

See also

Upper on page 420

String Functions on page 78

MailWIP

Use this procedure/function to send the current work-in-process to another user via email.

Syntax

```
MailWIP (Address)
```

Parameter	Description
Address	Enter a valid email address. The system defaults to the email address window which lets the user select a valid recipient. This window appears if the email address is omitted or incorrect.

The system optionally returns one (1) on success or zero (0) on failure.

If the MailWIP procedure succeeds in sending the WIP via email, the status of the form set will be changed to *Transmitted* and no longer appear as normal WIP in the sender's list.

NOTE: If the WIP is already following a routing slip's workflow, the form set will be sent to the next recipient in the existing slip.

Example

Here are some examples:

Procedure	Result	Explanation
MailWIP( )	1 or 0	The default presents the user with the email system's Address window, which lets the user choose the destination.
MailWIP("TOM")	1 or 0	If TOM is a valid email address for the email system, the form set will be sent. Otherwise, the Address window appears and the user chooses the correct address.

See also WIP Functions on page 88

MajorVersion

Use this function to get the major version number of the system being executed.

Syntax

```
MajorVersion ( )
```

There are no parameters for this function.

Example

Here is an example:

Function	Result	Explanation
#MAJOR = MajorVersion ()	string	Returns the system's major version number.

See also

Miscellaneous Functions on page 73

MinorVersion on page 312

DAL Script Examples on page 35

MAX

Use this function to return the greatest decimal value from a group of fields which have names that begin with common characters.

Syntax

```
MAX (PartialName, Section, Form, Group)
```

Parameter	Description
PartialName	Enter a valid string. The string must be the common (prefix) portion of a set of field names that occur on the current section. The default is the current field.
Section	Enter the name of a section that contains the field named. The default is the current section.
Form	Enter the name of a form that contains the section and/or field named. The default is the current form.
Group	Enter the name of the form group that contains the form, section, and/or field named. The default is the current group.

The system calculates and returns the average of the values of all fields that begin with the specified partial name. An example of field names that have a common start are:

Myfield1

Myfield2

Myfield20

Each of these fields is included if you specify the partial name using any of the leading characters of *myfield*. The first field will be excluded if you enter *myfield2*, but will match the other two field names.

The maximum is calculated by comparing all those fields that have values and have names matching the criteria. If all the field values are negative, then the result will be the negative number nearest the value zero. Note that zero (0) is a valid field value. Fields which have never been given a value are excluded from the calculation.

NOTE: Include the PartialName parameter. Fields must have unique names in a section. Using the default will probably not give the expected result, unless you created the form and understand the naming conventions.

Example

This table is used by the examples. The table represents the layout of two forms in the same group. Both forms share two sections (IMG A and IMG B). Each section has fields of the same name as a field in the other section.

Field	Section	Form	Group	Value
MyField1	IMG A	FRM A	GRP	100.24
MyField2	IMG A	FRM A	GRP	200.16

Field	Section	Form	Group	Value
MyField1	IMG B	FRM A	GRP	98.60
MyField2	IMG B	FRM A	GRP	* no value yet *
MyField1	IMG A	FRM B	GRP	0.00
MyField2	IMG A	FRM B	GRP	* no value yet *
MyField1	IMG B	FRM B	GRP	70.77
MyField2	IMG B	FRM B	GRP	* no value yet *

Here are some examples:

(Assume the current field is MyField1, on the first section of the first form. Reference the previous table for field values.)

Function	Result	Explanation
MAX ( )	100.24	Without any other information, the function will assume the current field and section. There will only be one value included in the search.
MAX (“Myfield2”)	200.16	Again, there is only one field included in this result.
MAX(“MyField”)	200.16	In this example, the current section contains two fields that begin with the name “MyField”. The second field has the greatest value.
MAX(“MyField”, “IMG B”)	98.60	Although two fields on IMG B have a matching name, only one field actually has a value.
MAX(“MyField”, , “FRM A”)	200.16	No section is specified in this example, so the entire form is searched. Four fields match the name criteria, but only three have values.
MAX(“MyField”, “IMG B”, , “GRP”)	98.60	This example specifies a section and group, but no form. There are four fields that match the name criteria, but only two have values.
MAX(“MyField”, , , “GRP”)	200.16	This example names the group without a form or section. Eight fields meet the naming criteria, but only five fields actually have values.

See also

Field Functions on page 61

Field Formats on page 62

Locating Fields on page 64

MIN

Use this function to return the least decimal value from a group of fields which have names that begin with common characters.

Syntax

```
MIN (PartialName, Section, Form, Group)
```

Parameter	Description
PartialName	Enter a valid string. The string must be the common (prefix) portion of a set of field names. The default is the current field.
Section	Enter the name of a section that contains the field named. The default is the current section.
Form	Enter the name of a form that contains the section and/or field named. The default is the current form.
Group	Enter the name of the form group that contains the form, section, and/or field named. The default is the current group.

The system calculates and returns the average of the values of all fields that begin with the specified partial name. An example of field names that have a common start are:

Myfield1

Myfield2

Myfield20

Each of these fields is included if you specify the partial name using any of the leading characters of *myfield*. The first field will be excluded if you enter *myfield2*, but will match the other two field names.

The minimum is calculated by comparing all those fields that have values and match the naming criteria. If all the values are negative, then the result will be the negative number most distant the value of zero. Note that zero (0) is a valid field value. Fields which have never been given a value are excluded from the calculation.

NOTE: Include the PartialName parameter. Fields must have unique names within a section. Using the default will probably not give the expected result, unless you created the form and understand the naming conventions.

Example

This table is used by the examples. The table represents the layout of two forms in the same group. Both forms share two sections (IMG A and IMG B). Each section has fields of the same name as a field in the other section.

Field	Section	Form	Group	Value
MyField1	IMG A	FRM A	GRP	100.24
MyField2	IMG A	FRM A	GRP	200.16

Field	Section	Form	Group	Value
MyField1	IMG B	FRM A	GRP	98.60
MyField2	IMG B	FRM A	GRP	* no value yet *
MyField1	IMG A	FRM B	GRP	0.00
MyField2	IMG A	FRM B	GRP	* no value yet *
MyField1	IMG B	FRM B	GRP	70.77
MyField2	IMG B	FRM B	GRP	* no value yet *

Here are some examples:

(Assume the current field is MyField1, on the first section of the first form. Reference the previous table for field values.)

Function	Result	Explanation
MIN ( )	100.24	Without any other information, the function will assume the current field and section. There will only be one value included in the search.
MIN (“MyField2”)	200.16	Again, there is only one field included in this result.
MIN(“MyField”)	100.24	In this example, the current section contains two fields that begin with the name *MyField*. The first field has the least value.
MIN(“MyField”, “IMG B”)	98.60	Although two fields on IMG B have a matching name, only one field actually has a value.
MIN(“MyField”, , “FRM A”)	98.60	No section is specified in this example, so the entire form is searched. Four fields match the name criteria, but only three have values.
MIN(“MyField”, “IMG B”, , “GRP”)	70.77	This example specifies a section and group, but no form. There are four fields that match the name criteria, but only two have values.
MIN(“MyField”, , , “GRP”)	0.00	This example names the group without a form or section. Eight fields meet the naming criteria, but only five fields actually have values. The least of these five contains the value 0.00.

See also Field Functions on page 61

Field Formats on page 62

Locating Fields on page 64

MinorVersion

Use this function to get the minor version number of the system being executed.

Syntax

```
MinorVersion ( )
```

There are no parameters for this function.

Example

Here is an example:

Function	Result	Explanation
vers = MajorVersion() & '.' & MinorVersion()	a string	Returns the system's major and minor version number concatenated together with a period used as a separator.

See also

Miscellaneous Functions on page 73

MajorVersion on page 307

DAL Script Examples on page 35

Minute

Use this function to extract the number of minutes from a time.

Syntax

```
Minute (Time, Format)
```

Parameter	Description
Time	Enter a valid time string. The system assumes your entry is in the time format specified in the Format parameter. The default is the current time.
Format	Enter a valid time format string that describes the Time parameter. The default is time format 1 (HH:MM:SS).

Example

Here are some examples:

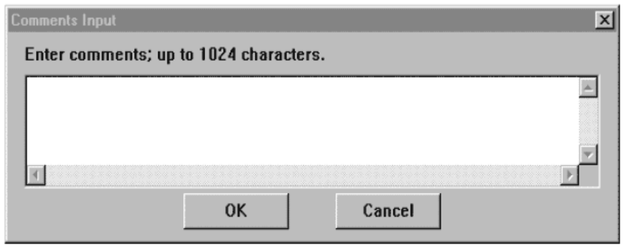
(Assume the current time is 03:05:09.)

Function	Result	Explanation
Minute( )	05	Defaults to the current time and extracts 05
Minute (“03:07:09”)	07	Reads the given time and extracts 07

See also Time Formats on page 80

MLEInput

Use this function to create a window with a title, prompt message, and a place for a user to enter multiple lines of text, such as the one shown here:



This function creates a window you can use to gather information from a user. The text entered through this window is returned as a string. If no text is assigned, or if the user closes the window by clicking on Cancel, the returned string will be empty.

Syntax

```
MLEInput (Prompt, Title, Length, DefText)
```

Parameter	Description
Prompt	Enter a text string to assign as the prompt for the field
Title	Enter a text string to assign as the title of the window.
Length	Enter the maximum input text length. THe default is 1024.
DefText	Enter a text string to assign as the default input data.

If the user presses ENTER to type on a new line, the system replaces the new line character with a *n* when it returns the text. You can leave the result like this, so you know where the line breaks are supposed to be, or you can send it to the MLETranslate function, which will translate the *n* into whatever characters you want.

NOTE: Multi-line variable fields cannot accept the data captured by the MLEInput function without the data first being translated. Before you assign the output from a MLEInput function to a multi-line variable field, you should do the following.

```
VALUE = MLETranslate (VALUE, "\n");
```

Where *VALUE* represents the text returned from the MLEInput statement. This will change all of the *n* occurrences to *n*, which is accepted by multi-line variable fields.

Example

Assume the user enters the following text (in quotes) into the window:

"line 1", Enter key, "line 3", "line 4", Entry key, and then "line 6"

Function	Results	Explanation
input_data = MLEInput (“Enter comments; up to 1024 characters.”, “Comments Input”); SetFld (input_data, variable”);	line 1\\n\\nline 3\\nline 4\\n\\nline 6	After you enter the information and click Ok, the DAL variable, 'input_data', contains the string in the result column. This example uses an A/N variable field.

input_data = MLEInput (“Enter your comments.”, “Comments Input”, , @(“ variable”));	1. Window:
	`line 1`
	`blank line 1`
	`line 3`
	`line 4`
	`blank line`
	`line 6`
	2. Input_data
	line 1\\nNow is the time\\nline 3\\nline 4 gray area\\n\\nline 6
		(Assume this DAL script is executed after the example above.)
		The window would contain the data under item 1.
		If you enter:
		- *Now is the time* in blank line 1
		- *gray area:* after 'line 4 '
		and click Ok. The DAL variable, 'input_data', will contain the string under item 2.
		This example uses an A/N variable field.

input_data = MLEInput (“Enter comments; up to 1024 characters.”, “Comments Input”);	Null string	Assume you clicked Ok or Cancel without entering any data. The system stores a null string in the variable.

input_data =MLETranslate (MLEInput (“Enter comments; up to 1024 characters.”, “Comments Input”), “\n”);
SetFld (input_data, “output”);
	1. DAL internal variable
	`line 1\n\nline 3\nline 4\n\nline 6`
	2. Multi-line variable field, *output*:
	`line 1`
	`blank line`
	`line 3`
	`line 4`
	`blank line`
	`line 6`
		After you enter the assumed information and click Ok, the DAL variable, 'input_data', contains the string shown in item 1.
		The data in the multi-line variable field, *output*, contains six lines, as shown in item 2.
		This example uses a multi-line variable field.

Function	Results	Explanation

input_data = MLETranslate (MLEInput ("Enter comments; up to 1024 characters.", "Comments Input", , @("output")), "\n");
 	
SetFld (input_data, "output1");
	1. Window
	`line 1`
	`blank line`
	`line 3`
	`line 4`
	`blank line`
	`line 6`
	
	2. Input_data
	line 1\nNow is the time\nline 3\nline 4 gray area\n\nline 6
	
	3. Multi-line variable *output1:*
	`line 1`
	`Now is the time`
	`line 3`
	`line 4 gray area`
	`blank line`
	`line 6`
		(Assume this DAL script is executed after the example above.)
		
		The window contains the data under item 1 after the DAL script is executed.
		
		Assuming you entered:
		- *Now is the time* in blank line 1
		- *gray area* after the data 'line 4 '
		and then clicked Ok, the DAL internal variable, 'input_data', contains the string shown in item 2.
		
		The multi-line variable field, *output1*, contains the data shown in item 3.
		
		This example uses a multi-line variable field.

See also MLETranslate on page 317

Documaker Workstation Functions on page 59

MLETRANSLATE

Use this function to translate the *n* characters in a data string created by the MLEInput function. This function translates those characters into whatever characters you want.

Syntax

```
MLETranslate (String, ReplaceChar)
```

Parameter	Description
String	Enter the text string returned from the MLEInput function.
ReplaceChar	Enter a text string to replace each set of \\n characters in a returned MLEInput data string.

The system returns the translated data string for display, storage, or both.

If the user presses ENTER to type on a new line, the system replaces the new line character with a *n* when it returns the text. You can leave the result like this, so you know where the line breaks are supposed to be, or, you can send it to the MLETranslate function, which will translate the *n* into whatever characters you want.

NOTE: Multi-line variable fields cannot accept the data captured by the MLEInput function without the data first being translated. Before you assign the output from a MLEInput function to a multi-line variable field, you should do the following.

```
VALUE = MLETranslate (VALUE, "\n");
```

Where *VALUE* represents the text returned from the MLEInput statement. This will change all occurrences of *n* to *n*, which is accepted by multi-line variable fields.

Example

Assume the user enters the following text (in double quotes) into the window.

“line 1", Enter key, “line 3", “line 4 “, Entry key, and then “line 6"

Function	Results	Explanation
input_data = MLETranslate (MLEInput (“Enter comments”, “Comments Input”), “*”); SetFld (input_data, variable”);	line 1**line 3*line 4 **line 6	After you enter the assumed information and click Ok, the DAL variable, 'input_data', contains the string in the result column. This example uses an A/N variable field.

Function	Results	Explanation
input_data = MLETranslate (MLEInput (“Enter comments; up to 1024 characters.”, “Comments Input”, , @(“ variable”)), “#”;	1. Multi-line edit window line 1**line 3*line 4 **line 6 2. Input_data line 1 Now is the time. #line 3*line 4 gray area**line 6	(Assume this DAL script is executed after the example above.) The window contains the data under item 1. Assuming you deleted the first two asterisks and entered *Now is the time.* followed by the entry key. Plus added *gray area* after line 4 and then clicked Ok. The DAL variable, 'input_data', would contain the data under item 2. This example uses an A/N variable field.
input_data = MLETranslate (MLEInput (“Enter comments; up to 1024 characters.”, “Comments Input”), “*”);	Null string	Assume you clicked Ok or Cancel without entering any data. The system stores a null string in the variable.

input_data =MLETranslate (MLEInput (“Enter comments; up to 1024 characters.”, “Comments Input”), “\n”);
SetFld (input_data, “output”);
	DAL internal variable
	line 1 \n\nline 3 \nline 4 \n\nline 6 \n
	Multi-line variable field
	`line 1`
	`blank line`
	`line 3`
	`line 4`
	`blank line`
	`line 6`
		After entering the assumed information and clicking Ok, the DAL variable, 'input_data', contains the data shown in item 1.
		The data in the multi-line variable, *output*, would contain six lines as shown in item 2.
		This example uses a multi-line variable field

Function	Results	Explanation

input_data = MLETranslate (MLEInput ("Enter comments; up to 1024 characters.", "Comments Input", , @("output")), "\n");
 	
SetFld (input_data, "output1");
	1. Window
	line 1
	blank line
	line 3
	line 4
	blank line
	line 6
	
	2. Input_data
	
	line 1 \nNow is the time. \nline 3 \nline 4 gray area \n\nline 6 \n
	
	3. Multi-line variable *output1*
	
	`line 1`
	`Now is the time.`
	`line 3`
	`line 4 gray area`
	`blank line`
	`line 6`
		(Assume this DAL script is executed after the example above.)
		
		After executing the script, the window contains the data shown in item 1.
		
		Assuming you entered:
		
		Now is the time. for blank line 1 area, and added *gray area*, after the data 'line 4 ' and then clicked Ok.
		
		The DAL internal variable, 'input_data', would contain the data string shown in item 2.
		
		The multi-line variable field, 'output1', would contain the data shown in item 3.
		
		This example uses a multi-line variable field.

See also

MLEInput on page 314

Documaker Workstation Functions on page 59

MOD

Use this function to return the remainder from modular arithmetic.

Syntax

```
MOD (Numerator, Denominator)
```

Parameter	Description
Numerator	Enter the value you want used as the numerator.
Denominator	Enter the value you want used as the denominator.

The system returns the integer remainder from an integer division.

NOTE: If you enter zero (0) as either the numerator or denominator, the system returns zero. Decimal or string input parameters are converted to integer values prior to the calculation.

Example

Assume you have the following entry in the SETRCPTBL.DAT file for the form trigger being processed. Also assume there are 30 records in the extract file that match the search mask.

```
;RP10;CIS;qa_f1550;;Customer(1);;1,M;25;0;1;;DALTrigger;F1550;
```

Here is an example:

```
BeginSub F1550
    #rec = CountRec("1,F1550,31,Data")
    #remaining = MOD(#rec, TriggerRecsPerOvFlw( ))
    While(#remaining > 0)
*       write additional records
        Write_fm( )
        #mod -= 1
    Wend
    Return(#rec)
EndSub
```

In this example, the MOD function returns the integer remainder of 5. If no extract records matched the search mask, the system would have returned zero (0).

See also

Mathematical Functions on page 72

MONTH

Use this function to determine the number of the month in a given date and return the number.

Syntax

```
Month (Date, Format, Locale)
```

Parameter	Description
Date	Enter a valid date string. The system assumes your entry is in the date format specified in the Format parameter. The default is the current date.
Format	Enter a valid date format string that describes the Date parameter. The default is date format 1.
Locale	(Optional) Enter the locale code. If you omit this parameter, the system checks the Locale INI option. If the Locale INI option offers no value, the system defaults to USD (United States/English).

The system determines the month portion of the given date based on the format you specify. This function is often used with the MonthName function.

Example

Here are some examples:

(Assume the current date is 07/01/09.)

Function	Result	Explanation
Month ( )	7	The parameter defaults to the current date.
Month ("09/138", "I")	5	The given date (09/138) in the date format I is the equivalent of May 18, 2009. Therefore the number of the month (5) is returned.
datestring= DateAdd(, , ,3); Month(datestring)	10	First the DateAdd function defaults to the current date and adds three months. The resulting date of October 1, 2009 is returned to the target variable *datestring*. The Month function then returns the number of the month of October (10).

See also

Date Functions on page 51

Locales on page 55

Date Formats on page 52

Locales on page 55

DateAdd on page 184

MonthName on page 322

MonthName

Use this function to find the name of the month in a given date and return that name.

Syntax

```
MonthName (Month, Locale)
```

Parameter	Description
Month	Enter a valid month value. For example, enter one (1) for January or 12 for December. The default is the current month.
Locale	(Optional) Enter the locale code. If you omit this parameter, the system checks the Locale INI option. If the Locale INI option offers no value, the system defaults to USD (United States/English).

This function is most often used with the Month function. The Month function extracts the month number from a given date.

Example

Here are some examples:

(Assume the current date is 07/01/09.)

Function	Result	Explanation
Return(MonthName())	July	Defaults to the current month.
Return(MonthName (11))	November	Returns November, which corresponds to the given parameter (11).
Return(MonthName (Month ("09/138", "I")))	May	First the Month function determines that the month number for the given date is 5. (09/138 is equivalent to May 18, 2009) Then MonthName returns the corresponding month name of May.

See also

Date Functions on page 51

Locales on page 55

Month on page 321

MSG

Use this procedure/function to create a message window with an Ok button. This procedure does not return a value.

Syntax

```
MSG (MsgLine1, MsgLine2, MsgLine3, Title)
```

Parameter	Description
MsgLine1	Enter the first line of the message.
MsgLine2	Enter the second line of the message.
MsgLine3	Enter the third line of the message.
Title	Enter a title for the message window.

This procedure provides the user with information. The message window is created as a standard message window.

This procedure displays a message each time the script executes. Therefore, use this procedure *only* in scripts that execute *once* during entry. Do not use the MSG procedure for scripts that execute each time a user tabs to a new field.

NOTE: A DAL script executes once during entry. A DAL calc executes each time the user tabs to a new field. You make the DAL script or DAL calc designation in the Properties window.

This is also useful in Documaker Workstation when debugging scripts.

Example

Here are some examples:

Procedure	Result	Explanation
MSG ("Sample Line 1", "Sample Line 2", "Sample Line 3", "Sample Message")		"Sample Message" is the title of the message. "Sample Line 1" "Sample Line 2" "Sample Line 3" is the message to the user.
MSG ("Don't forget to inform the customer about the luxury tax.")		The message appears without a title.

See also Documaker Workstation Functions on page 59

NL

Use this function to retrieve a string that contains a new line character sequence. This is useful when you are creating output text messages that contain line breaks.

NOTE: On Windows, this function returns a carriage return/line feed pair. On UNIX, it returns a line feed. The function works in both Documaker Server and Workstation.

Syntax

```
NL ( )
```

There are no parameters for this function.

Example

This example shows how you can use this function with the Print_It function:

```
Print_It("This is line one." & NL() & "This is line two.")
```

In this example, two lines are output to the command line during Documaker Server processing. Without this function, you would have to include two Print_It statements.

```
This is line one.
This is line two.
```

This example shows how you can create multi-line text area messages:

```
data = ?("cus_name") & NL() & ?("state") & ",  " & ?("zip")
SetFld(data, "cus_ss")
```

In this example, two lines are stored in a multi-line text area on separate lines. Without this function, you would have to define the multi-line text area, a fixed-size font, and the script would have calculated the number of spaces to pad to the first line to make sure the line wrapped properly.

```
John A. Smith
CA,  81234-4444
```

You can also use the NL function when you are creating comment strings you want inserted into a print stream using the AddComment procedure.

See also

String Functions on page 78

NUM

Use this function to return the numeric value of a field. On numeric formatted fields, this function operates the same as the @ function, however, NUM automatically converts a non-numeric field into its numeric content.

Syntax

```
NUM (Field, Section, Form, Group)
```

Parameter	Description
Field	Enter the name of a section field.The default is the current field.
Section	Enter the name of a section that contains the field you specified. The default is the current section.
Form	Enter the name of a form that contains the section and/or field you specified. The default is the current form.
Group	Enter the name of the form group that contains the form, section, and/or field you specified. The default is the current group.

The system uses the parameters provided to search for one field on a section and return that field's data as a number. The field does not have to be defined as a numeric data type.

Example

Here are some examples:

(Assume the current field value is ABC1234.23XYZ and is named MyField. Also, assume that a second occurrence of MyField appears on the form, MyForm, and contains the value *automobile*.)

Function	Result	Explanation
NUM( )	1234.23	Returns the value in the current field as a number. Notice that any non-numeric value is removed before returning the value.
NUM (“MyField”)	1234.23	Returns the value in the named field, located on the current section.
NUM(“MyField\ 2”, , “MyForm”)	0	Since the second occurrence of MyField on this form does not contain any numeric values, the result is zero (0).

See also

Field Functions on page 61

Field Formats on page 62

Locating Fields on page 64

@ on page 109

NUMERIC

Use this function to test if a string contains a valid numeric value. The system returns one (1) if the string is a valid number and zero (0) if not.

Syntax

```
Numeric (String)
```

Parameter	Description
String	Enter a valid string. The default is the value of the current field.

The system returns true or false depending on whether the string parameter contains a valid numeric value.

Leading or trailing spaces are removed before the string is evaluated. A numeric value contains only numbers, a sign (leading or trailing), and a single decimal point.

Example

Here are some examples:

(Assume the current field value is -101.564)

Function	Result	Explanation
Numeric ( )	1	Defaults to the current field and determines a true statement, such as if the field contains a valid numeric value.
Numeric (“123T456”)	0	Determines a false statement, such as if the field does not contain a valid numeric value.
IF Numeric (“4633392”) result = “Yes”; ELSE result = “No”; END Return(Result)	“YES”	The specified value is numeric therefore the variable result will be assigned Yes.

See also

Mathematical Functions on page 72

NumText

Use this function to convert a numeric value into a series of descriptive words.

Syntax

```
NumText (Number, DollarWord, CentWord, DeciMode)
```

Parameter	Description
Number	Enter an amount. The default is the value of the current field.
DollarWord	Enter the word you want the system to use to describe the main unit of currency. The default is: `"dollars and"`
CentWord	Enter the word you want the system to use to describe the secondary unit of currency. The default is: `"cents"`
DeciMode	Choose from these options: 1 - numeric decimal amount 2 - spell decimal amount 3 - suppress zero, numeric decimal amount 4 - suppress zero, spell decimal amount The default is one (1).

The system returns the written word equivalent of a numeric value.

The system attempts to remove formatting information from the parameter number. If the value after deformatting is not a valid number, the function returns an empty result.

This function is basically designed to produce the text that might appear on a bank check. The default type strings are *dollars and* and *cents*. When the default descriptions are used, this function uses the singular word *dollar* or *cent* when the associated value is 1, otherwise it uses the plural text. Alternate descriptions provided as parameters are not changed for any value amount.

The optional decimode parameter is an integer value from 1 to 4. This parameter includes or suppresses the zero (0) decimal value. You can also use this parameter to specify if the decimal amount should be presented as a number or spelled out.

NOTE: This function only supports two decimal places. Additional places are truncated without rounding.

Example

Here are some examples (assume the current field value is 1641.56):

Function	Result	Explanation
NumText ( )	One thousand six hundred forty-one dollars and 56 cents	Defaults to dollars and cents and numeric decimal result.
NumText(, , , 2)	One thousand six hundred forty-one dollars and fifty-six cents	Decimal mode 2 spells the decimal amount.

Function	Result	Explanation
NumText(12.00, , ,3)	Twelve dollars	A decimal mode of 3 suppresses the zero decimal.
NumText(34.55,”meters and”,”centimeters”,3)	Thirty four meters and 55 centimeters	Demonstrates substituting alternate references.
NumText(1.00,,,)	One dollar	
NumText(1.01,,,)	One dollar and one cent	
NumText(1.00,”meters and”,”centimeters”,3)	One meters and	
NumText(1.01,”meters and”,”centimeters”,3)	One meters and one centimeters	

If you include Dollarword and Centword and the number does not contain a decimal, the *exact* content you specify in Dollarword is printed and the system does not distinguish the number from being singular or plural. The Dollarword and Centword are printed exactly as specified. Notice the difference in the default format (dollars and cents) in the last two examples.

See also String Functions on page 78

FrenchNumText on page 248

PAD

Use this function to add trailing spaces or characters and return the result.

Syntax

```
PAD (String, Length, Char)
```

Parameter	Description
String	Enter a valid string. The default is the value of the current field.
Length	Specify the desired length of output. The default is the length of value in the String parameter.
Char	Enter a valid string that contains the pad characters you want to use. The default is the space character.

The system returns the string created by padding parameter 1 with the characters from parameter 3.

If the length specified in parameter 2 is longer than the string, the result is increased to the integer length you specified. If the specified integer length is less than the string, the length of the string is maintained.

The string is not truncated. All leading and trailing spaces are removed from the input string before the PAD function.

Example

Here are some examples:

(Assume the current field contains the text *Last Name Only*.)

Function	Result	Explanation
PAD("may ", 9)	"may "	Pad the result string to a length of 9. The pad character defaults to the space character.
PAD (, 20, !)	"Last Name Only!!!!!!"	Defaults to the current field, and adds the pad character (!) until the length reaches 20.
PAD ( )	"Last Name Only"	Defaults to the current field; No length was specified; therefore the field remains the same.
PAD ("Ten dollars ", 15, *)	"Ten dollars****"	Adds the pad character (*) to the end of the specified parameter until the length reaches 15. Notice that the trailing spaces were first removed and then padded with the new character.

See also

String Functions on page 78

PageImage

Use this function to return the name of a section on a given page number within the form set or form. If you include the name of a recipient as a parameter, the system will filter the sections by that name. Once you have a section name, you can use other DAL functions to query the section, to insert a new section, or to delete the section.

Syntax

```
PageImage (Page, Recipient, Form, Group)
```

Parameter	Description
Page	Include this parameter to indicate the specific page where you want to locate a section. If you omit this parameter, a section from page one is located. Depending upon the remaining parameters, this page will be the page within the entire form set, or within a given form.
Recipient	Include this parameter to filter the sections located by that recipient. If you omit this parameter, the name of the first section on the requested page is returned.
Form	Include this parameter if you want the system to first locate the specified form and then use the Page parameter to find the specified page within that form. If you omit this parameter, the Page parameter is based on a page located by starting at the first page of the form set or group (if the Group parameter is specified).
Group	Include this parameter to tell the system to first locate a specific group. If you also include the Form parameter, the system will find that form in that group. If you omit the group but include the form, the system looks for that form in the current group – which is identified by the current field or section executing the script. If you include the group but omit the form, the system uses the Page parameter to return that page in the specified group.

The name returned by this function also includes the *occurrence* value if the section occurs more than once. For instance, if a section named, *MySection*, is located on the given page, but this is the second occurrence of the section within the named form, the name returned will be *MySection\2*.

See also PageInfo on page 331

Name Functions on page 74

PAGEINFO

Use this function to get information about the page of a form you specify. This information includes height, width, and orientation.

Syntax

```
PageInfo (Prefix, Page, Recipient, Section, Form, Group)
```

Parameter	Description:
Prefix	This parameter identifies a prefix for creating the variable names to contain the page information.
Page	(Optional) This parameter determines the relative page number that should be examined, once the starting page is located by examining the remaining parameters. The default is the first page located.
Recipient	(Optional) This parameter names a specific recipient that must be used on a section of the page located. If you omit this parameter, the function matches the first page identified by the remaining search criteria.
Section	(Optional) This parameter names a section that should be found to identify the page.
Form	(Optional) This parameter names a form that contains the page to be found.
Group	(Optional) This parameter names a group that must contain the page to be found.

The Section, Form, and Group parameters are optional and when used will work together to locate the starting page for the search. Here are some examples:

- If you name a section, but no form or group, the assumption is the section is on the current form.
- If you name a form, without a group, the assumption is the form must be within the current group of forms.
- If you name a section and a group, but no form, the assumption is the section can occur on any form within that group.
- If you omit the section, form, and group parameters the search starts from the beginning of the document set.

Once the requested page is located, the system assigns the page information to DAL variables using the Prefix parameter. If these variables do not exist in DAL, the system creates them for you. The system creates four internal variables: *prefix*.height, *prefix*.width, *prefix*.landscape, and *prefix*.paper. If these variables exist, the system modifies them with the new information.

For example, a call like this will create four variables.

```
PageInfo("MYPAGE");
```

Variable	Description
MYPAGE.Height	Contains the height of the page in FAP units (2400 DPI).
MYPAGE.Width	Contains the width of the page in FAP units (2400 DPI).

Variable	Description
MYPAGE.Landscape	Contains one (1) if the page is landscape, otherwise zero (0).
MYPAGE.Paper	Contains a value that corresponds to a paper size table entry.

Note that for landscape pages, the height and width values reflect the rotation of the width and height. For instance, non-landscape letter documents return a height of 26400 and a width of 20400. Landscape letter documents return a height of 20400 and a width of 26400.

The page size (height and width) is determined by finding the first section on a page with the required recipient. If no recipient is specified, the first section on the page is used. The form pages within a document do not have to be the same size. Also note that if the first section on a page is a custom size, the width and height will reflect the *best* values.

Generally when a section is a custom size, the actual page size is found in the form definition. If, however, the form size (height or width) is smaller than the corresponding section size, then the larger of the values is returned.

Also remember since page size is determined by the first section designated for a given recipient, it is possible for the *same* page to have a different size for different recipients.

The PageInfo function returns a value if used in an expression that requires it. The possible return values are zero (0) if the requested page could not be found, or non-zero if the page is found.

Possible reasons for a page not to be located include:

- The page number is outside the range of pages for the given search criteria. For instance, you ask for page three of a form that only has two pages.
- The recipient cannot be located within the document search criteria.
- The section, form, or group (or combination thereof) cannot be located within the specified document.

See also

PageInfo on page 331

Page Functions on page 75

PAGINATEFORM

Use this function/procedure to apply section origins and re-pagination the form if necessary. During this re-pagination, the function will create or delete pages as needed.

NOTE: The AddImage and DelImage DAL functions include a parameter (Paginate) which you can use to force re-pagination after the affected section has been manipulated.

Syntax

```
PaginateForm (Form, Group)
```

Parameter	Description
Form	(Optional) If you omit this parameter, the current form controlling the active script is paginated. If you include the name of a form, that form is located and paginated. You can include the occurrence indicator (a backslash followed by a number, such as BIZ\3) to indicate a specific occurrence of the form to find and paginate. If you do not specify an occurrence with the name, the first occurrence of the form is paginated.
Group	(Optional) This parameter identifies the Key2 or GroupName2-level parent that contains the form. This is sometimes referred to as the *line of business* that contains the form If you omit the Group parameter, the system tries to locate the named form within the current group that is controlling the execution of the script.

You can call PaginateForm as a function or procedure. As a function, it returns a one (1) if the requested form is located or a zero (0) if it could not be located.

Note that if the form is found and paginated, there may not be any visible change to the document. The form layout is determined when you design the form and by the application of section origin rules.

See also

AddImage on page 122

DelImage on page 212

Page Functions on page 75

ParseListCount

Use this function to count the indexed components within the formatted text.

NOTE: Use the ParseListCount and ParseListItem functions when accepting tokenized (comma or semicolon-delimited) data, such as data from a spreadsheet program or other application. These are sometimes referred to as CSV (comma separated value) files.

Syntax

```
ParseListCount (String, Separator)
```

Parameter	Description
String	Enter the formatted string you want the system to search and parse.
Separator	Enter the list of character separators used within the formatted text parameter. If you omit this parameter, the system uses semicolons and commas.

The system returns the number of formatted items found within the String parameter. If the String parameter text starts with delimiter characters, those characters are skipped.

If you do not have at least a space character between delimiters, this will not be identified as a separate index item.

NOTE: You can use the ParseListItem function to return the text components parsed from the formatted text.

Example

For these examples, assume xString = “A,B;C”

```
value = ParseListCount(xString)
```

The value is *3*.

```
value = ParseListCount(xString,”;”)
```

The value is *2*. In this example the parameter overrides and assigns only a semicolon as a valid separator. Therefore, there are two items within this string.

For these examples, assume xString = “;A;,B,;C”

```
value = ParseListCount(xString)
```

The value is *3*. If the formatted string starts with separator characters, these characters are skipped. Note that adjacent separators are treated as a single separation.

For these examples, assume xString = “; ,A; ,B;”

```
value = ParseListCount(xString)
```

The value is *4*. Note the intervening character – a space - between some of the separator characters.

```
value = ParseListCount(xString,”;”)
```

The value is *2*. This overrides and assigns only a semicolon as the format separator, therefore there are only two components. Also note that although there are three separators, the first one that starts the string and the final one that ends the string are also ignored.

See also ParseListItem on page 336

String Functions on page 78

ParseListItem

Use this function to return indexed components from the formatted text.

NOTE: Use the ParseListCount and ParseListItem functions when accepting tokenized (comma or semicolon-delimited) data, such as data from a spreadsheet program or other application. These are sometimes referred to as CSV (comma separated value) files.

Syntax

```
ParseListItem (String, Item, Separator)
```

Parameter	Description
String	Enter a formatted string to search and parse.
Item	Enter the number of the item you want from within that formatted string. If you omit this parameter, the first item parsed from the formatted text is returned.
Separator	Enter a list of character separators used within the formatted text parameter. If you omit this parameter, the semicolons and commas are used.

The return value is a string of text. If the formatted text contains leading or trailing spaces on items formatted within it, they are not removed. You can use the Trim function on the returned text if you do not want the spaces.

If the first parameter text starts with delimiter characters, they will be skipped. Because the function will return spaces, you know when you have exceeded the number of items formatted within the string when you get an empty string returned.

NOTE: If you do not have at least a space character between delimiters, this will not be identified as a separate index item.

Example

Here are some examples. Assume xString = “A,B;C”

```
value = ParseListItem(xString)
```

The value is *A*.

```
value = ParseListItem(xString,3)
```

The value is *C* because the default separators include both commas and semicolons.

```
value = ParseListItem(xString,1,”;”)
```

The value is *A,B*. Note in this example the third parameter overrides and assigns only the semicolon as a valid separator. Therefore, the first item includes all text up to the first semicolon.

For these examples, assume xString = “;A;,B,;C”

```
value = ParseListItem(xString)
```

The value is *A*. Note that if the formatted string starts with separator characters they are skipped.

```
value = ParseListItem(xString,2)
```

The value is *B*. Note again how adjacent separators without intervening characters (or space) are skipped. Therefore the semicolon and comma (;,) between the A and B are treated as a single separation.

```
value = ParseListItem(xString,3)
```

The value is *C*. Note again how adjacent separators without intervening characters (or space) are skipped. Therefore the semicolon and comma (;,) between the *A* and *B* are treated as a single separation and the semicolon and comma (;,) between the *B* and *C* are also treated as a single separation.

```
value = ParseListItem(xString,3,",")
```

The value is *;C*. Note the third parameter overrides and assigns only the comma as a valid separator. Therefore the third index item includes all text following the second comma to the end of the string (because no other separators were encountered).

For these examples, assume xString = "; ,A; ,B;"

```
value = ParseListItem(xString)
```

The value is a space. Note that there is at least one intervening character — a space — between the first set of separator characters.

```
value = ParseListItem(xString,2)
```

The value is *A*.

```
value = ParseListItem(xString,3)
```

The value is a space.

```
value = ParseListItem(xString,4)
```

The value is *B*.

```
value = ParseListItem(xString,5)
```

The value is an empty string because this index item exceeds the list of items provided.

See also ParseListCount on page 334

String Functions on page 78

PathCreate

Use this function to create the parameter subdirectory path if it does not already exist. The function assumes all of the text you pass in is a path and does not remove any of it before it tries to verify or create the path.

The function creates multiple subdirectories as necessary in an attempt to satisfy the request.

NOTE: The PathCreate and PathExist functions let you create paths and verify that paths exist. These are useful, for instance, if you are trying to create printed output and organize that output into subdirectories on disk. You can do this using one of the print callback methods that support a DAL script.

Syntax

```
PathCreate (Path)
```

Parameter	Description
Path	Enter the full path you want the system to verify or create.

The system returns zero (0) if it cannot create the path requested. Anything else means the path now exists, but is not an indication that it had to be created.

NOTE: This function is not valid on the z/OS operating system.

See also

PathExist on page 339

File and Path Functions on page 68

PathExist

Use this function to take the parameter path you provide and check for its existence. This function does not create subdirectories.

NOTE: The PathCreate and PathExist functions let you create paths and verify that paths exist. These are useful, for instance, if you are trying to create printed output and organize that output into subdirectories on disk. You can do this using one of the print callback methods that support a DAL script.

Syntax

```
PathExist (Path)
```

Parameter	Description
Path	Enter the full path you want the system to verify.

The system returns zero (0) if the path is invalid. Anything else indicates the path you provided exists.

NOTE: This function merely checks for the existence of the path you specified. Provided the path does exist, this is not an indication that the process will be able to access or create files within that path.

See also

PathCreate on page 338

File and Path Functions on page 68

POW

Use this function to raise a number to an exponential power.

Syntax

```
POW (Base, Exponent)
```

Parameter	Description
Base	Enter the base number, positive or negative, to be raised to an exponential power. The default is 1.00.
Exponent	Enter the exponent (power) to which the base number will be raised. The default is zero (0).

The system returns a one (1) on success or a zero (0) on failure.

This function handles calculations such as those needed to figure annuities and interest rates. Using the function, a decimal number is returned from a base number that has been raised to an exponential power. Values can contain up to 14 digits.

The function handles both positive and negative integer or decimal values for the base number and exponent.

Example

Here is an example:

Function	Result
POW (2, 3)	8
POW (2, -3)	0.125
POW (34.5, 3.14)	67414.289005316

See also Mathematical Functions on page 72

PRINT

Use this procedure/function to print the entire document. Optionally this procedure returns one (1) on success or zero (0) on failure.

Syntax

```
Print ( )
```

There are no parameters for this procedure.

This procedure performs a similar action to choosing print from the menu. The user is shown the Print window from which he or she can choose printer options.

Example

Here is an example:

Procedure	Result	Explanation
Print ( )	1 or 0 (zero)	Print the current form set.

See also WIP Functions on page 88

Print_It

Use this procedure to print a string to the console.

Syntax

```
Print_It (Text)
```

Parameter	Description
Text	Enter the string you want the system to print to the console.

NOTE: This is useful when debugging scripts in Documaker Server.

Example

Here is an example:

Procedure	Result	Explanation
If (HaveGVM('Company')) Print_It (GVM('Company'))) End	a string	The content of the GVM variable *Company* is printed to the console.

See also

DAL Script Examples on page 35

Miscellaneous Functions on page 73

PRINTERCLASS

Use this function to find out the type of print stream the system is generating.

Syntax

```
PrinterClass ( )
```

There are no parameters for this function.

Example

Here are some examples. Assume these INI options exist:

```
< Printer >
     PrtType        = AFP
< PrtType:AFP >
    PrintViewOnly  = Yes
    OnDemandScript = OnDemand
```

Function	Result	Explanation
type = PrinterClass ( )	a string	The DAL target variable, *type* will contain *AFP*.
If (PrinterClass() = 'PrtType:AFP') Then AddComment(AppIdxRec()) End	a string	If the print type is *AFP* then execute following statement.

See also

AddComment on page 117

DAL Script Examples on page 35

Printer and Recipient Functions on page 76

PrinterGroup

Use this function to retrieve the group name that is being used to generate the print stream. This name is stored in the INI file.

Syntax

```
PrinterGroup ( )
```

There are no parameters for this function.

Example

Here are some examples. Assume these INI options exist:

```
< Printer >
     PrtType        = AFP
< PrtType:AFP >
     PrintViewOnly = Yes
     OnDemandScript= OnDemand
```

Function	Result	Explanation
G_name = PrinterGroup()	PrtType:AFP	Retrieves the printer group name.
ScriptName = GetINIString (, PrinterGroup(), 'OnDemandScript')	OnDemand	Contains the name of the DAL script you want to execute.

See also

GetINIString on page 259

DAL Script Examples on page 35

Printer and Recipient Functions on page 76

PRINTERID

Use this function to return the active printer ID assigned during a Documaker Server processing run. The printer ID is a string of text associated with the current batch output and normally determined via INI option during a batch run. The IDs are associated from the PrinterInfo control group with each batch printer definition.

You can use this ID, for instance, when naming print file. For example, you might want all the files from one printer ID in a separate location or have the names prefixed in a certain manner.

Syntax

```
PrinterID ( )
```

There are no parameters for this function.

NOTE: The printer ID is only valid during a batch print operation and calling the function at other times returns the last value assigned or an empty string.

See also Printer and Recipient Functions on page 76

PrinterOutputSize

Use this function to get the approximate size of the current print output file during a batch print operation.

Syntax

```
PrinterOutputSize ( )
```

There are no parameters for this function.

This function is only available during Documaker batch process operations, such as GenPrint, and only returns a non-zero value if a print stream is actively being built and written to a physical file on disk.

Note: When printing through the Windows GDI device, there is no physical file and therefore the value returned is unreliable and may be zero.

See also Printer and Recipient Functions on page 76

PutFormAttrib

Use this function to save the named attribute and information to a form within your document set. You can add new attributes via this function or update an attribute on a form you specify.

NOTE: Adding or changing a form attribute only affects the current document set. You cannot update the contents of a FORM.DAT or FOR file from a DAL script. Once changed, the attribute will stay with your form even if saved to WIP or archived.

Syntax

```
PutFormAttrib (Name, Data, Form, Group)
```

Parameter	Description
Name	Enter the name of the form attribute (metadata).
Data	Enter the value associated with the form attribute (metadata). The default is an empty string.
Form	Enter name of a form to retrieve data from. The default is the current form.
Group	Enter name of the group that contains the specified form. The default is the current group.

If you omit both the Form and Group parameters, the system chooses the current form, based on where the script executes. During Entry (via the Workstation or the plug-in) this will be the form that contains the DAL script. During Documaker Server processing, the first logical form found within the document set is the current form, unless the script is executed from a section or field rule.

If you include the Form parameter, but omit the Group parameter, the system looks for the form within the current group of forms, as defined by where the script executes. During Entry (via the Workstation or the plug-in) this is the group that contains the form where the script executes. During Documaker Server processing, the first logical group found within the document set is the current group, unless the script is executed from a section or field rule.

If you omit the Form parameter but include the Group parameter, the system locates the first form within the group you specified.

If the function is successful in adding the attribute to a form, it returns a one (1). If the function is not successful, it returns a zero (0). A failure typically means that based on the form and group name parameters, the function could not locate the form.

Example

In this example assume the form 1111 has this metadata:

Name	Value
Offer	Good until cancelled
Codes	R4,79, ZW

Here is an example:

```
xx=PutFormAttrib("Restriction", "Must be 18 or older", "1111")
```

After execution, the form contains the following:

Name	Value
Offer	Good until cancelled
Codes	R4,79, ZW
Restriction	Must be 18 or older

Keep in mind...

- The name of a user-defined attribute must follow the naming convention used for Documaker objects. This means the name cannot include semicolons (;), backslashes (\ ), equals signs (=), or two pipe symbols in sequence (||). You can use underscores (_), hyphens and dashes (-), and periods or full stops (.).
- You cannot use a pipe symbol (|) as the first character in a name or value.
- The value size cannot exceed 1000 characters for each value.
- The names *Category* and *Key3* are reserved. Avoid using these names.

See also

GetFormAttrib on page 255

Have Functions on page 69

PutINIBool

Use this procedure/function to store a Boolean value in an INI control group and option Boolean variable.

Syntax

```
PutINIBool (Context, Group, Option, Default)
```

Parameter	Description
Context	(Optional) Enter the name (valid string) associated with a set of INI control groups and options loaded into cache memory.
Group	Enter the name of the control group which contains the INI option Boolean variable.
Option	Enter the name of the option in which the INI Boolean variable will be stored. If the control group and option does not exist, the system creates them.
Default	(Optional) Enter the default Boolean value to store into the control group and option Boolean variable. The default is zero (0).

The system returns one (1) if no error occurred during execution and zero (0) if there was an error.

This procedure stores a Boolean value in the specified control group and option Boolean variable.

If you omit context name and the control group and option does not exist in any of the INI files, the procedure stores the Boolean value in the FSIUSER.INI file.

If there are multiple control groups and options with the same name, the procedure stores the Boolean value in the first INI control group and option variable equal to the specified control group and option name.

If a context name is present, the procedure only stores the Boolean value in the control group and option variable associated with the context name.

Example

Assume an INI file, *TEST1.INI,* was loaded with the context name, *MVF*. The TEST1.INI file contains this control group and option:

```
< Control >
    LogEnabled = 1
```

In addition, the FSIUSER.INI file contains this control group and option:

```
< Control >
    LogEnabled = 0
```

Plus, the FSISYS.INI file contains this control group and option:

```
< Control >
    LogEnabled = 1
```

Based on this scenario, the following table shows and explains several possible results.

Procedure	Result	Explanation
rc = PutINIBool (,"control", "LogEnabled");	The variable *bool_value* in the FSIUSER.INI file now contains a zero (0). The return code *rc* is set to one (1).	The procedure scanned the loaded INI control groups and options. It found the specified control group and option in the FSIUSER.INI first. The FSIUSER.INI set is searched first, followed by the FSISYS.INI set and then any other loaded sets, in order.
rc = PutINIBool ("MVF", "control", "LogEnabled");	The variable *bool_value* in the TEST1.INI file now contains a zero (0). The return code *rc* is set to one (1).	The procedure scans only the control group and option set associated with the context name *MVF*.
rc = PutINIBool ("MVF", "control", "LogEnabled", 1);	The variable *bool_value* in the TEST1.INI file now contains a one (1). If *Control* and *LogEnabled* are not found, the system creates a control group and option and sets the Boolean variable *LogEnabled* to one (1).	The procedure scans only the control group and option set associated with the context name *MVF*.

See also

INI Functions on page 70

Using INI Options on page 8

PUTINISTRING

Use this procedure/function to store a string value in a specified INI control group and option string variable.

Syntax

```
PutINIString (Context, Group, Option, Default)
```

Parameter	Description
Context	(Optional) Enter the name associated with a set of INI control groups and options loaded into cache memory.
Group	Enter the name of the control group name which contains the INI option string variable.
Option	Enter the name of the option into which you want the INI string variable stored. If the control group and option does not exist, the system creates them.
Default	(Optional) Enter the default string value you want to store in the control group and option string variable. The default is *Null*.

The system returns one (1) if no error occurred during execution and zero (0) if there was an error.

This procedure stores a string value into the specified control group and option string variable. If the context name is not present and the control group and option does not exist in any of the INI sets, the procedure stores the string variable into the FSIUSER.INI file.

If there are multiple control groups and options of the same name, the procedure stores the string value in the first INI control group and option variable equal to the specified control group and option name.

If a context name is present, the procedure only stores the string value in the control group and option variable associated with the context name.

Example

Let's assume that an INI file, *TEST1.INI,* was loaded with the context name, *MVF*. The TEST1.INI file contains this control group and option:

```
< Control >
    title = MVF's string
```

In addition, the FSIUSER.INI file contains this control group and option:

```
< Control >
    Title = Bob's string
```

Plus, the FSISYS.INI file contains this control group and option:

```
< Control >
    Title = fap entry
```

Based on this scenario, the following table shows and explains several possible results.

Procedure	Result	Explanation
rc = PutINIString (,"Control", "Title");	The string variable *Title* in the FSIUSER.INI file now contains *Bob's string*. The return code *rc* is set to one (1).	The procedure scanned the loaded INI control groups and options. It found the specified control group and option in the FSIUSER.INI first. The FSIUSER.INI set is searched first, followed by the FSISYS.INI set and then any other loaded sets, in order.
rc = PutINIString ("MVF", "Control", "Title");	The string variable *Title* in the TEST1.INI file now contains *MVF's string*. The return code *rc* will be set to one (1).	The procedure scans only the control group and option set associated with the context name *MVF*.
rc = PutINIString ("MVF", "Control", "Title", "New string");	The string variable *Title* in the TEST1.INI file now contains *New string*. If *Control* and *Title* are not found, the system creates them and sets the string variable *Title* to *New string*.	The procedure scans only the control group and option set associated with the context name *MVF*.

Example INI Functions on page 70

Using INI Options on page 8

RecipBatch

Use this function to get the name of the recipient batch file being processed. This function is only applicable to batch banner processing or comment record processing with the GenPrint program.

Syntax

```
RecipBatch ( )
```

There are no parameters for this function.

Example

Here is an example. Assume the recipient batch file entitled *Batch1* is being processed.

Function	Result	Explanation
rb = RecipBatch();	Batch1	Returns the name of the recipient batch being processed.

See also

RecipCopyCount on page 354

RecipName on page 356

Printer and Recipient Functions on page 76

RecipCopyCount

Use this function to count the number of recipient copies for specified sections and return that number.

Syntax

```
RecipCopyCount (Recip, Section, Form, Group)
```

Parameter	Description
Recip	(Optional) Enter the names of the recipients you want included in the count.
Section	Enter the names of the sections you want the function to look through.
Form	(Optional) Enter the names of the forms you want the function to look through.
Group	(Optional) Enter the names of the groups you want the function to look through.

If a recipient has a zero copy count, it is omitted from the total. For instance, if there are three recipients, all with a zero copy count, zero (0) is returned.

NOTE: The recipient list this function uses is the same one that generates the POLFile. The list is not re-generated from the POLFile, therefore if any changes occurred in the POLFile, those changes would not be represented in the internal list.

Example

Here is an example:

```
RecipCopyCount(Recip,Section,Form,Group)
[ReqType:i_Check]
    function=atcw32->ATCLogTransaction
    function=atcw32->ATCLoadAttachment
    function=dprw32->DPRSetConfig
    function=atcw32->ATCUnloadAttachment
    function=dprw32->DPRCheck
```

See also

RecipBatch on page 353

RecipName on page 356

Have Functions on page 69

RecipientName

Use this function to return from the FORM.DAT file the recipient name related to the specified section, form, or group.

You can use this function along with the HaveRecip function in DAL scripts to place a sequence number on each page of each recipient batch.

Syntax

```
RecipientName (Count, Section, Form, Group)
```

Parameter	Description
Count	An indexed reference to locate a recipient in the FORM.DAT file. The default is the first recipient in the FORM.DAT file.
Section	Enter the name of a section that contains the recipient. The default is the current section.
Form	Enter the name of a form that contains the recipient. The default is the current form.
Group	Enter the name of the form group that contains the recipient. The default is the current group.

If you omit the parameters, the system uses the first recipient it finds in the FORM.DAT file for the section, form, or group.

If the section, form, or group can not be located or the Count parameter causes the system to move beyond the last recipient in the FORM.DAT file for the section, form, or group, an empty string is returned.

See also

HaveRecip on page 274

Name Functions on page 74

RecipName

Use this function to get the name of the recipient batch record for the transaction currently being printed. This function is only applicable to batch banner processing or comment record processing with the GenPrint program.

Syntax

```
RecipName ( )
```

There are no parameters for this function.

Example

Here is an example. Assume the transactions for the Insured batch are being processed.

Function	Result	Explanation
rb = RecipName();	Insured	Returns the name of the recipient batch being processed.

See also

RecipCopyCount on page 354

RecipBatch on page 353

Printer and Recipient Functions on page 76

REFRESH

Use this procedure to refresh or repaint the screen.

Syntax

```
Refresh ( )
```

There are no parameters for this procedure.

Use this procedure with the AppendTxm, AppendText, DelLogo, Logo, and ChangeLogo procedures. The result from these procedures may not immediately display. Use the Refresh procedure to repaint the screen and display the text or graphic (LOG).

NOTE: This procedure is valid only in Documaker Workstation scripts.

Example

Here is an example:

Procedure	Result	Explanation
Refresh ( )	Repaints the screen.	New graphics or text now appears.

See also

Documaker Workstation Functions on page 59

AppendText on page 132

AppendTxm on page 134

DelLogo on page 214

Logo on page 303

ChangeLogo on page 163

RemoveAttachVAR

Use this function to remove an attachment variable. You can use this function when creating print comments using Documaker Bridge.

Syntax

```
RemoveAttachVAR (Name, DSIqueue)
```

Parameter	Description
Name	Enter the name of the attachment variable.
DSIqueue	(Optional) Enter one (1) for input or two (2) for output. The default is one (1).

The system returns one (1) if the variable was found and zero (0) if it was not found.

See also

Docupresentment Functions on page 60

AddAttachVAR on page 114

GetAttachVAR on page 252

RenameLogo

Use this procedure/function to rename a graphic (LOG).

Syntax

```
RenameLogo (Graphic, NewName, Section, Form, Group)
```

Parameter	Description
Graphic	Enter the name of the graphic you want to rename. Graphic names are assigned in Studio or Image Editor.
NewName	Enter the new name for the graphic.
Section	Enter the name of the section that contains the specified graphic. The default is the current section.
Form	Enter the name of the form that contains the section. The default is the current form.
Group	Enter the name of the group to use to locate the specified object. The default is the current group.

The system returns one (1) on success or zero (0) on failure.

This procedure renames the graphic you specify. The Logo procedure, which adds a graphic on the fly, names the new graphic using the name you specify.

If you want a more generic name so you can address the graphic again without knowing the file associated with it, use this procedure *after* you use the Logo procedure.

You must specify both the Graphic and NewName parameters.

Example

Here are some examples:

Procedure	Result	Explanation
RenameLogo(“Log1”, “Jane Doe”)	1 or 0	Renames Log1 (on the current section, form, and group) to Jane Doe.
RenameLogo(“Log1”,”Jane Doe”,”IMH1\3”, “UpRate”)	1 or 0	Renames Log1 (on the 3rd occurrence of the named section, IMH1, on the form, UpRate) to Jane Doe.

See also

ChangeLogo on page 163

DelLogo on page 214

HaveLogo on page 272

InlineLogo on page 282

Logo on page 303

Name Functions on page 74

ResetFld

Use this procedure/function to delete the data from a variable field, including multi-line variable fields. This procedure works even if no data was entered into the field.

Syntax

```
ResetFld (Field, Section, Form, Group)
```

Parameter	Description
Field	Enter the name of the field you want to reset. Enclose the field name in quotation marks. Here is an example: `"FIELD01"`
Section	Enter the name of the section that contains the field name. The default is the current section.
Form	Enter the name of a form that contains the section or field name or both. The default is the current form.
Group	Enter the name of the form group that contains the form, section, and/or field name. The default is the current group.

Example

Here are some examples:

Procedure	Result	Explanation
ResetFld ("ACCUM_TOT")	1 or 0	Clears the data from the ACCUM_TOT field.
ResetFld ("YEARTODATE")	1 or 0	Clears the data from the YEARTODATE field.
ResetFld("TOTAL_PREM", "BOAT PREM")	1 or 0	Clears the data in the field, TOTAL_PREM, in the section, BOAT PREM.

See also

Field Functions on page 61

ResetOvFlwSym

Use this procedure/function to reset the value in an overflow symbol to zero.

Syntax

```
ResetOvFlwSym (Form, Symbol)
```

Parameter	Description
Form	Enter the name of the form that contains the fields on which overflow processing will occur.
Symbol	Enter the name you want to use as the overflow symbol.

The system optionally returns one (1) on success or zero (0) on failure. This procedure stores a zero (0) in the specified overflow symbol.

Note: This procedure provides DAL with the functionality included in Documaker Server's ResetOvFlw and ResetOvSym rules.

Example

Here is an example:

```
#reset_rc = ResetOvFlwSym ("CP0101NL", "Loc_Cnt")
```

In this example, the overflow symbol, *Loc_Cnt*, is set to zero and the DAL integer variable, *#reset_rc*, is set to a one (1) on success and zero (0) on failure.

Syntax

AddOvFlwSym on page 127

GetOvFlwSym on page 262

IncOvFlwSym on page 280

Documaker Server Functions on page 58

RETAIN

Use this procedure to identify DAL variables that should not be cleared between the processing of transactions.

Syntax

```
Retain (Variable)
```

Parameter	Description
Variable	Enter the names of the DAL variables (as a quoted string) you want to retain during the processing of transactions.

Keep in mind that certain features rely upon DAL variables living forever. This procedure lets you identify the DAL variables you do not want cleared during the processing of transactions.

This procedure is not required unless you have the FlushDALSymbols option set to Yes, as shown here:

```
< RunMode >
    FlushDALSymbols = Yes
```

The Retain procedure works in both the Documaker and Documaker Workstation environments and is necessary when you want certain variables to live for the entire session.

NOTE: Declaring a variable to be retained does not affect the value you assign to the variable. The Retain procedure does not protect that variable's value from being changed in subsequent scripts that are executed.

Once declared as retained, a variable cannot be later removed from the list.

Example

Here is an example:

```
$total_amt = Sum("$prem_");
Retain ("$total_amt");
```

In this example, the DAL variable *$total_amt* will survive transaction boundaries and can be referenced in any subsequent transaction DAL script.

See also

Using INI Options on page 8

Miscellaneous Functions on page 73

Right

Use this function to return a specified number of right most characters.

Syntax

```
Right (String, Integer)
```

Parameter	Description
String	Enter a valid string. The default is the value of the current field text.
Integer	Enter the desired length for the output. The default is the length of the String parameter.

If the length you specify in the integer parameter is longer than the string, the system pads the result with spaces to reach the requested length. The input string is first trimmed of leading and trailing spaces before the output is determined.

Example

Here are some examples:

(Assume the current field contains the text *Your Name*.)

Function	Result	Explanation
Right ( )	“Your Name”	Defaults to the current field; No length was specified; therefore the field remains the same.
Right (“ est text”, 9)	“test text”	Takes the nine right most characters from the specified field and returns the result.
Right (“Complete Street Address”, 14)	“Street Address”	Takes the 14 right most characters from the specified field and returns the result.

See also

String Functions on page 78

Left on page 296

RootName

Use this function to extract and return the root name, or the original part of the name, of a string you specify. This function strips off the *#nnn* portion of a field name to get the root field name.

NOTE: Documaker requires that all fields on a section be uniquely named. Studio and Image Editor force a unique name if a field is duplicated. Appending *#002* or *#003*, for example, to the end of the field name creates unique names. In some cases you may want to use the name of a field to supply the name of a data dictionary symbol to use to fill that field. If each unique instance of a field is to use the same name, this can present a problem.

Syntax

```
RootName (Field)
```

Parameter	Description
Field	Enter the name of the field for which you want the system to return the root portion of that name.

Example

Here are some examples:

```
RootName("Street address #002")
```

This returns *Street address*.

```
MYFIELDNAME = "Comment #003"
RootName(MYFIELDNAME)
```

This returns *Comment*.

```
RootName(FieldName())
```

This returns the root name of the current field.

See also Name Functions on page 74

ROUND

Use this function to round a number to the nearest specified decimal point and return the result.

Syntax

```
Round (Number, Places)
```

Parameter	Description
Number	Enter a valid numeric value with decimals. The default is the value of the current field.
Places	Enter the number of decimal places you want. The default is two (2).

The system returns the string value of a decimal number rounded to the number of places specified.

The sign of the number is not changed. Decimal numbers maintain up to 14 digits of precision. The Round function returns the value with or without trailing zeros requested. If you use the result the Round function returns in a mathematical equation or to represent a decimal parameter, the string is implicitly converted as needed.

Example

Here are some examples:

(Assume the current field value is 23.5473)

Function	Result	Explanation
Round ( )	23.55	Defaults to the current field and to two decimal places.
Round (, 3)	23.547	Defaults to the current field and uses three decimal places.
Round (101.999, 0)	102	Rounds the given value to zero decimal places.
Round (101.999, 4)	101.9990	Rounds the given value to four decimal places.

NOTE: When using the result of the Round function to assign a section field value, make sure the numeric field is defined *without* a format. If the field has a format, it may override the text provided by this function.

See also

String Functions on page 78

ROUTEWIP

Use this procedure/function to send all the work contained in WIP to all the recipients specified in a routing slip.

Syntax

```
RouteWIP (Slip)
```

Parameter	Description
Slip	Enter the name of a routing slip. The default is to let the system display a window that lets the user select a routing slip

The system optionally returns one (1) on success or zero (0) on failure.

If the WIP is already following a routing slip's workflow, the form set is sent to the next recipient in the existing slip.

Example

Here are some examples:

Procedure	Result	Explanation
RouteWIP( )	1 or 0	Displays for the user the Routing Slip Selection window.
RouteWIP ("manager")	1 or 0	This specifies the routing slip named, *manager*. If successful, the WIP is sent to the first recipient in the list. If the slip name is invalid, the user can choose another slip.

See also

WIP Functions on page 88

RPErrorMsg

Use this procedure to write an error message into Documaker Server's error file (ERRFILE.DAT). In addition, it increments the Documaker Server error count as necessary.

Syntax

```
RPErrorMsg (Message)
```

Parameter	Description
Message	(Optional) Enter the message you want the system to display. The message can consist of static text, DAL functions, and procedures. The default is a blank string.

Example

Here is an example:

```
RPErrorMsg ( )
```

This example would cause the following to be written into your ERRFILE.DAT file:

```
Error : In DAL Script <.\deflib\iso_create.dal> at line <23>: (Text
omitted)
```

Here is another example:

```
RPErrorMsg ("Failed to Open the INFO table in iso_create.")
```

This example would cause the following to be written into your ERRFILE.DAT file:

```
Error : In DAL Script <.\deflib\iso_create.dal> at line <18>: Failed
to Open the INFO table in iso_create.
```

Here is another example:

```
RPErrorMsg (Time() & " " & Date() & " variable =  " & table_name)
```

This example would cause the following to be written into your ERRFILE.DAT file:

```
Error : In DAL Script <.\deflib\iso_create.dal> at line <26>:
11:51:02 08/06/2003 variable = INFO
```

See also

RPLogMsg on page 368

RPWarningMsg on page 369

Documaker Server Functions on page 58

RPLogMsg

Use this procedure to write a message into Documaker Server's log file (LOGFILE.DAT).

Syntax

```
RPLogMsg (Message)
```

Parameter	Description
Message	(Optional) Enter the message you want the system to display. The message can consist of static text, DAL functions, and procedures. The default is a blank string.

Example

Here are some examples:

```
RPLogMsg ( )
```

This example would cause the following to be written into your LOGFILE.DAT file:

```
Message : In DAL Script <.\deflib\iso_create.dal> at line <23>: (Text
omitted)
```

Here is another example:

```
RPLogMsg ("Failed to Open the INFO table in iso_create.")
```

This example would cause the following to be written into your LOGFILE.DAT file:

```
Message : In DAL Script <.\deflib\iso_create.dal> at line <18>:
Failed to Open the INFO table in iso_create.
```

Here is another example:

```
RPLogMsg (Time() & " " & Date() & " variable =  " & table_name)
```

This example would cause the following to be written into your LOGFILE.DAT file:

```
Message : In DAL Script <.\deflib\iso_create.dal> at line <26>:
11:51:02 08/06/2003 variable = INFO
```

See also

RPErrorMsg on page 367

RPWarningMsg on page 369

Documaker Server Functions on page 58

RPWarningMsg

Use this procedure to write a warning message into the Documaker Server error file (ERRFILE.DAT). In addition, it increments the Documaker Server warning count as necessary.

Syntax

```
RPWarningMsg (Message)
```

Parameter	Description
Message	(Optional) Enter the message you want the system to display. The message can consist of static text, DAL functions, and procedures. The default is a blank string.

Example

Here are some examples:

```
RPWarningMsg ( )
```

This example would cause the following to be written into your ERRFILE.DAT file:

```
Warning : In DAL Script <.\deflib\iso_create.dal> at line <23>: (Text
omitted)
```

Here is another example:

```
RPWarningMsg ("Failed to Open the INFO table in iso_create.")
```

This example would cause the following to be written into your ERRFILE.DAT file:

```
Warning : In DAL Script <.\deflib\iso_create.dal> at line <18>:
Failed to Open the INFO table in iso_create.
```

Here is another example:

```
RPWarningMsg (Time() & " " & Date() & " variable =  " & table_name)
```

This example would cause the following to be written into your ERRFILE.DAT file:

```
Warning : In DAL Script <.\deflib\iso_create.dal> at line <26>:
11:51:02 08/06/2003 variable = INFO
```

See also

RPErrorMsg on page 367

RPLogMsg on page 368

Documaker Server Functions on page 58

SaveINIFile

Use this procedure/function to save a set of INI control groups and options that were loaded into cache memory.

Syntax

```
SaveINIFile (Context, File)
```

Parameter	Description
Context	(Optional) A name (valid string) associated with a set of INI control groups and options loaded into cache memory.
File	Enter the name of the file in which you want to store the specified set of INI control groups and options. If you omit the file extension, the system uses *INI*. If you omit the path, the system stores the file in the current directory.

The system optionally returns one (1) on success or zero (0) on failure.

If a context name is associated with the execution of this procedure, that set of INI control groups and options will be stored in the specified physical file name.

Example

Here are some examples:

Procedure	Result	Explanation
SaveINIFile (,"DALRun");	The set of INI control groups and options are saved in a file.	The INI control groups and options are saved to the specified file. Execution of this procedure assumes that the file extension is *INI*.
SaveINIFile ("Run_process", "DALRun.ini");	The set INI control groups and options referenced by the context name, *Run_process*, are saved in a file.	The INI control groups and options are saved to the specified file.

See also

INI Functions on page 70

Using INI Options on page 8

SaveWIP

Use this procedure/function to save the WIP record being processing. Optionally, this procedure returns a one (1) on success or a zero (0) on failure. This procedure is needed in the DAL script called by the Documaker Workstation function, AFEBatchDALProcess, if you change any data in the WIP record being processed.

Syntax

```
SaveWIP ( )
```

There are no parameters for this procedure.

Example

Here is a sample DAL script.

```
desc_field   = WIPFld(”DESC”);
mod_data     = desc_field & ” - 04/03/03”;
rc_setwipfld = SetWIPFld("DESC", mod_data);
rc_savewip   = SaveWIP( );
```

This script appends the text, – *04/03/03*, to the content of the DESC field in each WIP record in the WIP.DBF file.

See also

Executing a DAL Script from a Menu on page 7

SECOND

Use this function to extract the number of seconds in a time.

Syntax

```
Second (Time, Format)
```

Parameter	Description
Time	Enter a valid time string. The system assumes your entry is in the time format specified in the Format parameter. The default is the current time.
Format	Enter a valid time format string that describes the Time parameter. The default is time format 1 (HH:MM:SS).

Example

Here are some examples:

(Assume the current time is 03:05:09.)

Function	Result	Explanation
Second( )	09	Defaults to the current time and extracts 09.
Second (“09:20:20”)	20	Reads the given time and extracts 20.

See also Time Functions on page 80

SetDeviceName

Use this procedure to set a new output device file name which will be used the next time the output device is opened, assuming nothing overrides the name prior to that. You can use this function when splitting recipient batches into multiple print stream files.

Syntax

```
SetDeviceName (Device)
```

Parameter	Description
Device	Enter the new output device file name.

Here is an example of script logic from a post-transaction banner DAL script:

```
IF TotalSheets() > 16000
    #COUNTER += 1
    CurFile = DeviceName()
    Drive = FileDrive(CurFile)
    Path = FilePath(CurFile)
    Ext = FileExt(CurFile)
    RecipBatch = RecipBatch()
    NewFile = FullFileName(Drive,Path,RecipBatch & #COUNTER,Ext)
    SetDeviceName(NewFile)
    BreakBatch()
END
```

NOTE: See FileDrive, FileExt, FileName, FilePath, and FullFileName for information on using DAL functions to manipulate file names.

Keep in mind...

- The print drivers supported are: PCL5, PCL6, PST, MET, AFP, PDF, HTML, and RTF.
- These print drivers *are not* supported: EPT, MDR, and GDI.
- All platforms are supported, but note that while UniqueString is supported on z/OS, z/OS does not support PDF or long file names.
- Both multi-step and single-step processing are supported.

The only DAL function actually involved in splitting the print stream is BreakBatch. The others make it easier to implement this functionality. For example, since you need to name the new print stream, you use the SetDeviceName procedure. To find the name of the current device, you use the DeviceName function. If you need to create unique file names, you can use the UniqueString function.

NOTE: While you can call all of these DAL functions in Documaker Server or Entry, the BreakBatch and SetDeviceName functions are not applicable in Entry since it does not use the batch printing engine. The other functions, DeviceName and UniqueString, are applicable to both Entry and Documaker Server.

See also
[Printer and Recipient Functions on page 76]
[BreakBatch on page 158]
[DeviceName on page 217]
[UniqueString on page 421]

SetEdit

Use this procedure/function to determine which field should be the next active field during normal entry. Normal entry refers to tabbing from field to field. If a user mouse clicks a particular field or pages between sections, the field selected by the SetEdit procedure is ignored. This procedure optionally returns one (1) on success or zero (0) on failure.

Syntax

```
SetEdit (Field, Count, Section, Form, Group)
```

Parameter	Description
Field	Enter the name of a field. The default is the current field.
Count	Enter a positive or negative number used to move beyond the field you specified. The default is zero (0).
Section	Enter the name of a section that contains the field named. The default is the current section.
Form	Enter the name of a form that contains the section and/or field named. The default is the current form.
Group	Enter the name of the form group that contains the form, section, and/or field named. The default is the current group.

This procedure first locates the specified field. If you omit the Field parameter, the system uses the current field.

You can use a positive or negative number for the count parameter. A positive count moves forward from the located field. A negative count moves backward from the located field. Forward and backward refer to the order in which the field appears in the section's edit list, not necessarily its physical position on the section. Do not include fields designated as display only in the count.

This procedure sets the next edit field each time the script executes. Therefore, use this procedure *only* in scripts that execute once during entry. Do not use the SetEdit procedure for scripts that execute each time a user tabs to a new field.

NOTE: A DAL script executes once during entry. A DAL calc executes each time the user tabs to a new field. You make the DAL script or DAL calc designation in the Properties window.

This procedure returns one (1) if it finds the specified field. Otherwise, it returns zero (0).

NOTE: The navigation logic you enter on the Navigation tab of the field's Properties window overrides this procedure.

Here are some examples:

Assume the section has three fields named FIRST, SECOND, and THIRD. The fields occur in that order.

Procedure	Result	Explanation
SetEdit(”THIRD”)	1	Locates the field named *THIRD* on the current section. If found and if editable, that field will be the next field to receive focus.
SetEdit(“THIRD”.-2)	1	Locates the field named *THIRD* on the current section. If found, moves two fields prior to *THIRD*--to the field named *FIRST*.
SetEdit(“MyField”,,, “FRM”)	1 or 0	Locates the form named *FRM* in the current form group. Then locates *MyField* on that form. If found, focus changes to that form and field.

See also Documaker Workstation Functions on page 59

Field Formats on page 62

Locating Fields on page 64

SETFLD

Use this procedure/function to assign a value to a section field. Normally, this procedure is used to assign values to display only fields or to assign default values to fields which have not yet been edited.

Syntax

```
SetFld (String, Field, Section, Form, Group)
```

Parameter	Description
String	Enter a value appropriate for the field you are assigning. THe default is an empty string.
Field	Enter the name of a field. The default is the current field.
Section	Enter the name of a section that contains the field named. The default is the current section.
Form	Enter the name of a form that contains the section and/or field named. The default is the current form.
Group	Enter the name of the form group that contains the form, section, and/or field named. The default is the current group.

The system returns one (1) if successful or zero (0) if the field cannot be changed or does not exist.

This procedure attempts to change the field's text each time the script executes. Therefore, use the SetFld procedure with discretion. Do not use the SetFld procedure if the script should not execute *each time* a user highlights a new field.

NOTE: A DAL script executes once during entry. A DAL calc executes each time the user highlights a new field. You make the DAL script or DAL calc designation in the Properties window.

If you are using the SetFld procedure in a batch system execution and you are trying to set a field other than the one which initiated the rule, you must load the FAP file. To do so, add the CheckImageLoaded rule to the sections to which you plan to assign fields.

Trailing spaces are deleted from the string to be stored. If you need the spaces, use a hard space (ALT + 0160). See the Rules Reference for more information about this rule.

Example

Here are some examples:

(Assume the section has three fields (First, Second, Third). The value of First is 123.)

Procedure	Result	Explanation
SetFld("N/A", "SECOND")	1	Assume this script is associated with the field named First. When the user tabs from this field or highlights another field, the value of the field, Second is changed to *N/A*.

Procedure	Result	Explanation
IF (!SetFld(101, “MyField”)) MSG(“Field” & “MyField”, “not assigned!”) END	0	The IF statement determines whether or not the field MyField can be assigned the value “101”. If not (meaning a field by that name does not exist or failed to accept the data), the message “Field MyField not assigned!” appears.
SetFld(@( ), “MyField”, , “FRM”)	1 or 0	This statement attempts to assign the value of the current field to MyField, located on the specified FRM. Since a section name was not given, the field may occur on any section on that form.

See also Field Functions on page 61

Field Formats on page 62

Locating Fields on page 64

SetFont

Use this function to change the font on a field. For instance, you can use SetFont on non-multi-line text fields or bar code fields. You cannot use the SetFont function to reformat a text area.

Syntax

```
SetFont (FontID, Field, Section, Form, Group)
```

Parameter	Description
FontID	Enter the font ID of the font to which you want to change. A font ID of less than one (1) causes the function to fail.
Field	(Optional) Enter the name of a field that identifies a multi-line text area. This is the field that receives the appended text. The default is the current field.
Section	(Optional) Enter the name of the section that contains the field you specified. The default is the current section.
Form	(Optional) Enter the name of the form that contains the section and/or field you specified. The default is the current form.
Group	(Optional) Enter the name of the form group that contains the form, section, and/or field you specified. The default is the current group.

The system returns one (1) on success or zero (0) on failure.

The system applies the font change to the first field that matches the criteria.

See also Field Functions on page 61

SetFormDesc

Use this function to change the description of a form.

Syntax

```
SetFormDesc (NewDescription, Form, Group)
```

Parameter	Description
NewDescription	Enter the new description. The text you enter replaces any existing form description.
Form	Enter the name of the form for which you want to change its description. The default is the current form.
Group	Enter the name of the group which contains the form you specified in the Form parameter. The default is the current group.

The system returns one (1) if the form was found and the description was assigned. Otherwise, it returns zero (0) to indicate that no form was found based upon the parameters you provided.

Example

Here is an example:

```
SetFormDesc("Cover Page",Form1,Group2)
```

See also

Name Functions on page 74

SetGVM

Use this procedure/function to update the contents of a GVM variable. You can also use this procedure to create a GVM variable.

Syntax

```
SetGVM (Name, Data, Instance, Type, Size)
```

Parameter	Description
Name	Enter a string which contains the name of the GVM variable.
Data	Enter the data you want to store in the GVM variable.
Instance	Enter the instance number of the GVM variable. The default is one (1)
Type	Indicate the type of GVM variable to create. You can choose from these options: C - Character array S - Short L - Long F - Float D - Double Q - Long double
Size	Enter the number of bytes to reserve when creating a GVM variable. This parameter is not used if the GVM already exists.

The system returns a one (1) if successful or a zero (0) if not.

NOTE: You can use this function to set a reserved GVM value, but be aware of how that reserved GVM is used. Some reserved GVM values should not be modified, such as NA_OFFSET and POL_OFFSET. Additionally, keep in mind that reserved GVM values may be changed by subsequent rule processing.

Example

Here are some examples:

Procedure	Result	Explanation
If (HaveGVM('Company')) then; SetGVM('Company', 'My Company') End	1 or 0	If the variable exist; then set the GVM, *Company*, to the string *My Company*.
If (HaveGVM('My Variable') = 0) then; SetGVM('My Variable','My Data',, 'C', 50) End	1 or 0	If the GVM variable, *My Variable*, does not exist; then create one that is a character array with a size of 50 plus store *My Data* in it.

See also

HaveGVM on page 270

DAL Script Examples on page 35

Documaker Server Functions on page 58

SetImagePos

Use this procedure/function to reposition a section on a page.

Syntax

```
SetImagePos (PrefixName, Section, Form, Group)
```

Parameter	Description
PrefixName	A prefix name to be associated with the coordinates returned by the procedure.
Section	Enter the name of a section in the form set. The default is the current section.
Form	Enter the name of a form in the form set that contains the section. The default is the current form.
Group	Enter the name of the form group that contains the form and section you specified. The default is the current group.

This procedure repositions a section at the coordinates you specify in the PrefixName parameter: *prefix name.top*, *prefix name.left*.

NOTE: The section remains the same size.

This procedure retrieves these variables and sets the section's top coordinate to *prefix name.top* and its left coordinate to *prefix name.left*. This procedure returns a bad variable error message if the *prefix name.top* or *prefix name.left* variables are not defined as DAL internal variables.

Example

For this example, assume the current section is *Image25*, the form is *Input_form*, and the form group is *Package1*. The coordinates are:

	Image25	For internal variables	Image50
Top	25	125	95
Left	50	150	90

NOTE: The the Bottom-Right coordinate is automatically calculated from the new Top-Left coordinate by adding the section height and width, which are not changed by this DAL function.

Procedure	Result	Explanation

SetImagePos ("MyImage")	New coordinates for the current section, *Image25*, will be:
	`Myimage.top = 125`
	`Myimage.left = 150`
	`Myimage.bottom = 200`
	`Myimage.right = 200`
		Sets the coordinates for the current section to the internal DAL variables: Myimage.top, Myimage.left, Myimage.bottom, and Myimage.right.

Procedure	Result	Explanation

SetImagePos (“MyImage”, “Image50”)	New coordinates for the section, *Image50*, will be:
	`Myimage.top = 125`
	`Myimage.left = 150`
	`Myimage.bottom = 200`
	`Myimage.right = 200`
		Sets the coordinates for the section, Image50, to the internal DAL variables: Myimage.top, Myimage.left, Myimage.bottom, and Myimage.right.

SetImagePos (“m”, “MVF\2”, “XYZ”)	The section is reposition to:
	`m.top = top`
	`m.left = left`
	`m.bottom = coordinate`
	`m.right = right`
		The second occurrence of the section MVF on the form XYZ is repositioned using the DAL target variables.

```
IF (ImageRect (“MyRect”, “MyImage”))
     MyRect.Top += 2400;
     SetImagePos(“MyRect”,”MyImage”);
END;
```

This script takes the coordinates of the section named MyImage and sets them to the variables *MyRect.Top*, *MyRect.Left*, *MyRect.Bottom,* and *MyRect.Right*. Next, it increases *MyRect.Top* by 2400 FAP units then moves MyImage one inch (2400 FAP units) lower on the page.

See also

Section Functions on page 77

ImageRect on page 278

SETLINK

Use this function to update a hyperlink setting in a variable field, a graphic, or a text label.

Syntax

```
SetLink (Target, Parms, ObjectName, Section, Form, Key2, ObjectType)
```

Parameter	Description
Target	Enter the name of the target object (the HREF value). If the target object has a hyperlink type of *internal* or *target*, enter the name of the target object. If the target object has a hyperlink type of *external*, this parameter should contain a hypertext reference, such as: `www.oracle.com` and the Parms parameter should contain additional parameters to an HREF type link. Make sure this parameter contains valid HTML syntax.
Parms	(Optional) Enter any link parameters (HREF parameters), such as a target frame or mouseover behavior. Here is an example: `"target="new"` Make sure this parameter contains valid HTML syntax.
ObjectName	Enter the name of the variable field, graphic, or text label that contains the hyperlink. The system updates the first object found that matches your entry for this parameter.
Section	(Optional) Enter the name of the section.
Form	(Optional) Enter the name of the form.
Key2	(Optional). Enter the name of the Key2 group.
ObjectType	Enter the type of object, such as (variable) Field, Graphic, or Text (label). The default is Field.

Keep in mind...

- The object (variable field, graphic, or text label) referenced by SetLink must have an initial hyperlink setting.
- You must make sure the Target and Parms parameters contain valid HTML syntax.

Example

Here is an example:

```
SETLINK("http://www.oracle.com", "target=new", "Section2256",
"FormQ1331TPG", , , "Text")
```

See also

Field Functions on page 61

SetLogo

This function is obsolete and is no longer supported. Use the ChangeLogo function instead.

See also ChangeLogo on page 163

Graphics Functions on page 71

SetProtect

Use this procedure/function to protect a specified field so it cannot be altered or to unprotect a field so that it can be edited.

Syntax

```
SetProtect (Mode, Field, Section, Form, Group)
```

Parameter	Description
Mode	Enter a non-zero value to specify field protection mode. Enter zero (0) to leave the field unprotected. The default is one (1), which protects the field.
Field	Enter the name of the field. The default is the current field.
Section	Enter the name of a section that contains the field you specified. The default is the current section.
Form	Enter the name of a form that contains the section and/or field named. The default is the current form.
Group	Enter the name of the form group that contains the form, section, and/or field named. The default is the current group.

The system returns zero (0) if the field you specified could not be changed or does not exist in the section. The system returns one (1) if the field was successfully protected.

Example

Here are some examples:

(Assume the section has fields named *First* and *Second*. Assume *First* contains *Y*.)

Procedure	Result	Explanation
IF (@( ) ="Y") SetProtect(1, "SECOND"); END	1	Tests the value of the current field (First). Since it contains the letter Y, *Second* is protected. If you call SetProtect as a procedure, a one (1) is returned, indicating the field was successfully protected.
IF (@( ) ="Y") SetProtect (0, "SECOND"); END	1	Unprotects *Second* based on the same criteria.

See also

Field Functions on page 61

Field Formats on page 62

Locating Fields on page 64

SETRECIP

Use this procedure/function to assign the recipient copy count for a particular section, form, or group.

Syntax

```
SetRecip (Recipient, Count, Section, Form, Group)
```

Parameter	Description
Recipient	Enter a valid recipient name for the sections you want to change.
Count	Enter the total number of copies of the designated sections that you want this recipient to receive. The default is zero (0).
Section	Enter the name of the section you want to locate. The default is the current section.
Form	Enter the name of the form that contains the section you specified. The default is the current section.
Group	Enter the name of the group that contains the section or form you specified. The default is the current section.

The system optionally returns one (1) on success or zero (0) on failure.

This procedure lets you specify how many copies of a section should print for a designated recipient. Setting the copy count to zero (0) for a recipient means the section will not print for that recipient.

Unlike other procedures, this one does not strictly apply the hierarchy rules for the section, form, and group, so you can specify a form without naming a section and the procedure will assign the copy count to all sections on that form designated for that recipient. Likewise, if you only specify the group parameter, without section or form, all the sections in that group will receive the new copy count for the designated recipient.

NOTE: This procedure cannot add a new recipient to a section. Images are predefined for specific recipients. This procedure can only change the copy count of known recipients for any particular section.

Example

Here are some examples:

Procedure	Result	Explanation
SetRecip (“Insured”, 2)	1 or 0	Defaults to the current section. If this section includes Insured as a recipient, that copy count will be assigned 2.
SetRecip(“HOME OFFICE”, 1, , “FORM”)	1 or 0	Locate FORM in the current group. Assign any section that specifies HOME OFFICE as a recipient the new copy count of one (1).

See also Section Functions on page 77

SetRequiredFld

Use this function to change the required option of a field to Required or Not Required.

Syntax

```
SetRequiredFld (Required, Field, Section, Form, Group)
```

Parameter	Description
Required	Enter Yes if you want to make the field required. Enter No if you want to make the field optional.
Field	(Optional) Enter the name of the field. The default is the current field.
Section	(Optional) Enter the name of the section. The default is the current section.
Form	(Optional) Enter the name of the form. The default is the current form.
Group	(Optional) Enter the name of the group. The default is the current group.

Example

Here are some examples:

```
SetRequiredFld ("Yes", "Myfield", :MyImage", "Myform", "MyGroup");
SetRequiredFld ("Yes", "Myfield", :MyImage", "Myform", );
SetRequiredFld ("Yes", "Myfield", :MyImage", );
SetRequiredFld ("Yes", "Myfield",);
SetRequiredFld ("Yes", );
```

If you include the Section parameter, but omit the field parameter, the system uses the first field on that section. If you omit the Section and Field parameters, but include the Form, the system looks for the first field on the first section of the form you specified, and so on.

See also

Field Functions on page 61

Field Formats on page 62

SetWIPFld

Use this procedure/function to set WIP fields from DAL to the record in memory.

Syntax

```
SetWIPFld (Field, Data)
```

Parameter	Description
Field	Enter the name of the variable field.
Data	Enter the data you want to store in the field.

NOTE: You cannot change the FormsetID field which is used to associate WIP records with data files.

The system returns one (1) if successful or zero (0) if the field cannot be changed or does not exist.

Example

Here are some examples:

Procedure	Result	Explanation
SetWIPFld ("DESC", "My Description")	1 or 0	Assigns to the WIP description field a new description.
SetWIPFld("DESC")	1 or 0	Clears the WIP description field.

See also

WIP Functions on page 88

SIZE

Use this function to return the defined length of a specified field.

Syntax

```
Size (Field, Section, Form, Group)
```

Parameter	Description
Field	Enter the name of the variable field. The default is the current field.
Section	Enter the name of the section that contains the field you specified. The default is the current section.
Form	Enter the name of a form that contains the section and/or field you specified. The default is the current form.
Group	Enter the name of the form group that contains the form, section, and/or field you specified. The default is the current group.

The system returns the length of the defined data area for the specified field.

The Size function is often confused with the LEN function. The LEN function returns the length of the actual data contained in a field or DAL variable.

Example

Here are some examples:

(Assume the current field contains the text *Your Name* and its defined length is 15.)

Function	Result	Explanation
Size ( )	15	Returns the defined length of the current field.
Size (“Myfield”, , “FRM”)	field size or zero	This example will look for MyField on the form, FRM. It may occur on any section. If the field is located, its size will be returned, otherwise the result is zero (0). Generally, you can assume that a zero result means the field is not defined, since it is unlikely that a field of zero length would be legitimate.

See also

Field Formats on page 62

Locating Fields on page 64

LEN on page 297

Field Functions on page 61

SLIPAPPEND

Use this procedure/function to add an email address to the end of the routing slip associated with the form set.

Syntax

```
SlipAppend (Address, Mode)
```

Parameter	Description
Address	Enter an email address.
Mode	Choose from these options: 0 - linear recipient 1 - CC recipient The default is zero (0).

The system optionally returns one (1) on success or zero (0) on failure.

This procedure only works with scripts associated with routing slips. This procedure lets you, via scripts, direct workflow during the routing process. Do not use this procedure in a typical field script situation.

The address name is appended to the end of the current routing slip. If the mode parameter is not zero (0), the new entry is appended as a carbon-copy (CC) recipient. For example, assume the following routing slip is defined:

CC	Recipient
	@MyScript
	EDJ

If the script executes the statements, SlipAppend("TOM",1); SlipAppend("CAR"), the slip will be adjusted to look as follows:

CC	Recipient
*	@MyScript EDJ
	EDJ
X	TOM
	CAR

Example

Here are some examples:

Procedure	Result	Explanation
SlipAppend ("TOM")	1 or 0	The email address is appended to the end of the current routing slip. The defaults is a linear recipient.
SlipAppend("TOM", 1)	1 or 0	Appends the email address as a CC recipient.

See also

WIP Functions on page 88

SlipInsert

Use this procedure/function to insert another email address on a routing slip associated with the form set.

Syntax

```
SlipInsert (Address, Mode)
```

Parameter	Description
Address	Enter an email address.
Mode	Choose from these options: 0 - linear recipient 1 - CC recipient The default is zero (0).

The system optionally returns one (1) on success or zero (0) on failure.

This procedure only works with scripts associated with routing slips. This procedure lets you, via scripts, direct workflow during the routing process. It should not be used in a typical field script situation.

The address name is inserted immediately after the script reference in the routing slip. If two SlipInsert statements are executed in order, the second email address appears before the one inserted by the former statement. Think of this as last in, first out.

For example, assume the following routing slip is defined:

CC	Recipient
	@MyScript EDJ

If the script executes the statements, SlipInsert("TOM",1); SlipInsert("CAR"), the slip will be adjusted to look as follows.

CC	Recipient
*	@MyScript CAR
X	TOM EDJ

The asterisk (*) indicates the script has already been executed. If the mode parameter is not zero (0), the new entry is appended as a carbon-copy (CC) recipient.

Example

Here are some examples:

Procedure	Result	Explanation
SlipInsert ("TOM")	1 or 0	The email address will be inserted immediately after the script reference. The default is a linear recipient.
SlipInsert("TOM", 1)	1 or 0	Inserts the email address as a CC recipient.

See also WIP Functions on page 88

SpanField

Use this function/procedure to move a field horizontally and then resize it to span the distance between two other fields you specify. This function sets the span field's contents to be enough of a fill character to span the distance.

This function only moves the field horizontally. It will not move the other two fields. The section designer must ensure vertical alignment between the fields.

NOTE: If you use this function with resources created prior to version 11.0, which had separate FAP and DDT files, this procedure automatically loads the section (FAP or compiled FAP) if it is not already loaded.

Syntax

```
SpanField (SpanField, LeftField, RightField, Section, Form, Group)
```

Parameter	Description
SpanField	Use this parameter to specify the filler character you want the system to use to span the distance between the end of the left field text and the beginning of the right field text. If either field is empty, the left coordinate of the field is used. The system only uses the first character of the text contained in the field you specify as the filler character. In addition to the filler character, the field you specify also determines the font ID to be used for calculating the number of characters required to fill the width of the field. If there is fractional space remaining in the width, the filler character is duplicated. The extra white space will be placed to the left of the span field, so that the spanned field will is placed against the right-most field. The default is a period (.).
LeftField	Enter the name of the field on the left of the area you want to span.
RightField	Enter the name of the field to the right of the area you want to span.
Section	(Optional) Enter the name of a section that contains the fields you specified. The default is the current section.
Form	(Optional) Enter the name of a form that contains the section and/or field you specified. The default is the current form.
Group	(Optional) Enter the name of the form group that contains the form, section, or fields you specified. The default is the current group.

The SpanField parameter is always the first parameter, but you can specify the LeftField and RightField parameters in any order. The system automatically determines which of the two fields is to the right or left of the span field.

NOTE: If you are using the SpanField function in Documaker Server processing, the JustFld rule may be useful to right justify the right-most field to make sure the maximum distance is spanned. If you use the Move_It rule, or other rules that support right justification by padding the data with spaces, the results will be incorrect. The SpanField function calculates the width of a field based upon the entire contents and does not remove space, or any other white space or characters in the fields.

Example

Here is an example:

Assume LeftField contains ABCDEFG, RightField contains $123.45, and SpanField contains a dash (-).

```
SpanField("SPANFIELD", "LEFTFIELD", "RIGHTFIELD")
```

Yields: ABCDEFG———$123.45

The horizontal location of the span field is adjusted to make sure it is positioned against the right edge of the left field, and then expanded with enough of the fill character to fill the gap between the left and right fields. The section designer is responsible for vertical alignment.

See also

Field Functions on page 61

SRCHDATA

Use this function to retrieve data from an XML or flat extract file.

NOTE: The SrchData function, released in version 11.1 and included in version 11.0, patch 32, lets you include spaces in the search criteria, whereas the older GetData function does not. Here is an example:

```
SrchData("11,HEADERREC,21(A,B, ,D)", 40, 20)
SrchData("'!/XML/Form[@form="PP 03 02"]/@form", 1,10)
```

Note the space between *A,B, ,D* and *PP 03 02*. The ability to include spaces in search criteria is important when you are using XML XPaths.

The SrchData function does not format the data it returns.

Syntax

```
SrchData (SearchCriteria, Offset, Length, Occurrence)
```

Parameter	Description
SearchCriteria	Enter the criteria you want the system to use to look for the data in the extract file.
Offset	For XML extract files, enter the offset into the data where the desired data starts. For flat files, enter the offset into the record where the data starts. The default is zero (0).
Length	Enter the number of characters to return. The default is zero (0).
Occurrence	This parameter is not valid for XML extract files. This parameter lets you specify which occurrence of the data to return. Entering one (1) or zero (0) returns the first occurrence of the data. The default is the first occurrence.

Use this function during Documaker Server processing, after the rule which loads the extract file has been run.

Example

Here are some examples:

In this example, the SrchData function finds the extract record designated by *11,HEADEREC* and returns the data at offset 40 for a length of 20:

```
SrchData ("11,HEADERREC", 40, 20)
```

This example shows how to use an occurrence variable to get the Nth iteration of the data. In this example, the SrchData function finds the second extract record occurrence designated by search criteria *11,ADDRESS*, and returns the data starting at offset 40 for a length of 20.

Entering a one (1) or zero (0) will return the first occurrence of the data.

```
SrchData ("11,ADDRESS", 40, 17, 2)
```

Here is an example that gets data from an XML extract file. The SrchData function checks to see if the specified XML extract record equals 2549, if it does, the function returns the string: *equal* concatenated with the value from another XML extract record. If not, it returns the string: *not equal* concatenated with a value from a different XML extract record.

```
value = SrchData ("!Diamond/Data/Client/Accounts/Account/
        Policy/PolicyImages/Policy/premium_fullterm", 1, 7)
If Trim (SrchData ("!Diamond/Data/Client/Accounts/Account/
       Policy/PolicyImages/Policy/premium_fullterm", 1, 4) = "2549"
  Then
       Return ("equal - " & SrchData ("!/descendant::Personalauto/
           child::Vehicle[**vehovfsym**]/vehicle_num", 1,2)
  Else
       Return ("not equal - " & value)
End
```

See also GetData on page 253

Documaker Server Functions on page 58

STR

Use this function to return the string value of a field. The @ function automatically converts a numeric format field into its number value. The STR function does not convert field data in any way and returns the value as it appears in the field.

NOTE: To consider case in the comparison, use the STRCompare function.

Syntax

```
STR (Field, Section, Form, Group)
```

Parameter	Description
Field	Enter the name the field. The default is the current field.
Section	Enter the name of a section that contains the field named. The default is the current section.
Form	Enter the name of a form that contains the section and/or field named. The default is the current form.
Group	Enter the name of the form group that contains the form, section, and/or field named. The default is the current group.

The system uses the parameters you provide to search for one field on a section and return that field's data value as formatted. The field can have any format type.

Example

Here are some examples:

(Assume the current field value is $1,234.23 and is named MyField. Also, assume that a second occurrence of MyField appears on the form, MyForm, and contains the value *automobile*.)

Function	Result	Explanation
STR( )	$1,234.23	Returns the string value in the current field. Notice that the formatting of the field is not removed.
STR ("MyField")	$1,234.23	Returns the string value of the named field, located on the current section.
STR("MyField\2 ", , "MyForm")	automobile	The second occurrence of MyField already contained a string value.

See also

STRCompare on page 398

Field Functions on page 61

Field Formats on page 62

Locating Fields on page 64

@ on page 109

STRCompare

Use this function to compare two strings with case a consideration. In normal DAL string expressions, strings are compared in a case-insensitive manner. For example, the system would normally evaluate the following strings to be equal:

```
ABC        abc
```

If, however, you use the STRCompare function, the system considers case and judges these strings to not be equal.

NOTE: The best way to use this function is to test for equality. For instance, use this function to test two strings and compare for a zero (0) value being returned to indicate the strings are equal or a non-zero value to indicate they are unequal.

You can use this function to determine if one string is greater or less than the other, but the result can be confusing if the strings contain mixed case or have different lengths.

Syntax

```
STRCompare (String1, String2, #Count)
```

Parameter	Description
String1	Enter the text for the first string you want to compare. The default is an empty string.
String2	Enter the text for the second string you want to compare. The default is an empty string.
#Count	(Optional) Enter the number of characters to compare. If you enter a value greater than zero, the system compares that number of characters. If you enter zero (0) or less, the system compares all characters. If you enter a value greater than the length of either string, the system pads the strings with blank characters to match the number of characters you specified. The default is -1 which indicates that all characters will be compared.

If String1 and String2 compare as equal, the system returns a zero (0).

The system returns a negative one (-1) if String1 is less than String2.

The system returns a one (1) if String1 is greater than String2.

Example

Assume String1 is *ABCDEF* and String2 is *ABCdef* in these examples:

This example	Returns
#RTN = STRCompare(string1 , string2)	-1
#RTN = STRCompare(string2 , string1)	1
#RTN = STRCompare(string1 , string2 , 3)	0

See also STR on page 397

String Functions on page 78

SUB

Use this function to return a substring from a string at a specified position.

Syntax

```
SUB (String, Position, Length)
```

Parameter	Description
String	Enter a valid string. The default is the current field.
Position	Enter the position where sub should begin. The default is one (1).
Length	Enter the length to retrieve from the text. The default is the length of what remains of the String parameter value, beginning at the position indicated by the Position parameter.

The system returns a portion of the first specified parameter starting at the specified position for the length given.

If you omit the Position parameter, the system defaults to the first character of the string. If the specified position is greater than the length of the string, the system returns an empty result.

If you omit the Length parameter, the remainder of the string following the specified position is included.

Example

Here are some examples:

(Assume the current field contains the text *Your Name*.)

Function	Result	Explanation
SUB (, , 5)	"Your "	Defaults to position one of the current field and returns the first five characters.
SUB ( )	"Your Name"	Defaults to the current field; No length was specified, so the field remains the same.
SUB ("Complete Street Address", 10, 6)	"Street"	Goes to position 10 of the specified field and returns six characters.

See also String Functions on page 78

SUM

Use this function to return the decimal sum of a group of fields which have names that begin with common characters.

Syntax

```
SUM (PartialName, Section, Form, Group)
```

Parameter	Description
PartialName	Enter a valid string. The string must be the common (prefix) portion of a set of field names that occur on the current section. The default is the current field.
Section	Enter the name of a section that contains the field you specified. The default is the current section.
Form	Enter the name of a form that contains the section and/or field you specified. The default is the current form.
Group	Enter the name of the form group that contains the form, section, and/or field you specified. The default is the current group.

The system calculates and returns the accumulated values of all fields that begin with the specified partial name.

An example of field names that have a common start are:

Myfield1

Myfield2

Myfield20

Each of these fields will be included if the partial name is specified using any of the leading characters of *myfield*. The first field will be excluded if you enter *myfield2*, but will match the other two field names.

NOTE: Include the *Partialname* parameter. Fields must have unique names within a section. Using the default will probably not give the expected result, unless you created the form and understand the naming conventions.

Example

Here are some examples:

This table is used by the examples. The table represents the layout of two forms in the same group. Both forms share two sections (IMG A and IMG B). Each section has fields of the same name as a field in the other section.

Field	Section	Form	Group	Value
MyField1	IMG A	FRM A	GRP	100.24
MyField2	IMG A	FRM A	GRP	200.16
MyField1	IMG B	FRM A	GRP	98.60

Field	Section	Form	Group	Value
MyField2	IMG B	FRM A	GRP	* no value yet *
MyField1	IMG A	FRM B	GRP	0.00
MyField2	IMG A	FRM B	GRP	* no value yet *
MyField1	IMG B	FRM B	GRP	70.77
MyField2	IMG B	FRM B	GRP	* no value yet *

(Assume the current field is MyField1, on the first section of the first form. Reference the previous table for field values.)

Function	Result	Explanation
SUM ( )	100.24	Without any other information, the function assumes the current field and section. There will only be one value included in the sum.
SUM ("Myfield2")	200.16	Again, there is only one field included in this result.
SUM("MyField")	300.40	In this example, the current section contains two fields that begin with the name "MyField". The equation is as follows: (100.24 + 200.16).
SUM("MyField", "IMG B")	98.60	Although two fields on IMG B have a matching name, only one field actually has a value.
SUM("MyField", , "FRM A")	399.00	No section is specified in this example, so the entire form is searched. Four fields match the name criteria, but only three have values: (100.24 + 200.16 + 98.60).
SUM("MyField", "IMG B", , "GRP")	169.37	This example specifies a section and group, but no form. There are four fields that match the name criteria, but only two have values: (98.60 + 70.77).
SUM("MyField", , , "GRP")	469.77	This example names the group without a form or section. Eight fields meet the naming criteria, but only five fields actually have values: (100.24 + 200.16 + 98.60 + 0.00 + 70.77).

See also

Mathematical Functions on page 72

Field Formats on page 62

Locating Fields on page 64

SuppressBanner

Use this procedure/function to suppress the printing of the banner page. This is useful when you are doing batch banner processing you need to combine several transactions within the same transaction banner pages.

NOTE: For information about processing banner pages, see the Documaker Administration Guide.

Syntax

```
SuppressBanner ( )
```

There are no parameters for this procedure.

Example

Here is an example.

Procedure	Result	Explanation
SuppressBanner();		Suppresses the current banner from printing.

You can use this procedure when you want to combine several transactions inside one set of banner pages, based on a flag the DAL script checks.

See also

Printer and Recipient Functions on page 76

DelBlankPages on page 208

AddBlankPages on page 115

Table

Use this procedure/function to look up and return a value from a standard table.

Syntax

```
Table (RetCode, Key, Table, File)
```

Parameter	Description
RetCode	Enter a return code value designated by the letters *K* and *D*. For example: K - key code D - code description K + D - key code and code description D + K - code description and key code The default is the value of the current field table return value.
Key	Enter the table key code. The default is the value of the current field text.
Table	Enter the name of the table you want to search. Note that this parameter is case sensitive. The default is the current field table.
File	Enter the name of the file that contains the table you specified in the Table parameter. Note that this parameter is not case sensitive. The default is the value of current field table file name, or the current section table file name.

This procedure makes sure a given value (Key) is an entry in the specified table (Table). This procedure returns the string value identified in the RetCode parameter.

The table name in the Table parameter and file name in the File parameter must conform to the naming conventions used for naming tables in Studio or Image Editor. If the Key parameter does not occur within the named table, the return string is empty.

You can include one of these INI options to specify that entry table files will use the old or new format. (*Do not* include both options.)

```
< Tables >
    OldFormatOnly = Yes
    NewFormatOnly = Yes
```

For instance, if you are doing a lot of entry table lookups from the DAL code, your tables are located on a network drive, and the tables are a mix of both old and new format tables, performance can be affected because the system has to check the format of each table.

If, however, you can use one of these new options to tell the system that all tables are in the same format, it can omit that query and performance improves.

Specify *only* the option that applies. If you omit both options, the system first checks to see if the table is in the new format. If not, then it checks to see if the table is in the old format.

Keep in mind that if you include one of these options, all of your tables must be in that format. For instance if you set the OldFormatOnly option to Yes, all of your tables must be in the old format. If you later decide to convert your tables to the new format, you must remove this option and, to get the same performance gain, include the NewFormatOnly option.

Example

Here are some examples:

Procedure	Result	Explanation
Table (“D”, “GA”, “STATCOD”, “table1”)	Georgia	Verifies that a table named STATCOD is contained in the file named *table1*. Then returns the description (Georgia) for the key code *GA*.

See also

Documaker Workstation Functions on page 59

TIME

Use this function to build a time from a given time, or the current time.

Syntax

```
Time (Format, Hour, Minutes, Seconds)
```

Parameter	Description
Format	Enter a time format string. The default is time format 1 (HH:MM:SS).
Hour	Enter a number to indicate the hour. The default is the current hour.
Minutes	Enter a number to indicate the minute. The default is the current minute.
Seconds	Enter a number to indicate the second. The default is the current second.

The system returns a time string that contains a formatted time value.

If you omit one of the Hour, Minute, or Seconds parameters, the system uses the appropriate value from the current time.

Example

Here are some examples:

(Assume the current time is 07:07:32 am.)

Function	Result	Explanation
Time( )	07:07:32	No parameters entered. It defaults to the current time in format 1.
Time(2,13,30,5)	01:30:05 PM	Format 2 selected; time displays in 12-hour format using these values.

See also

Time Formats on page 80

TIME2TIME

Use this function to convert a time from one format to another.

Syntax

```
Time2Time (OldTime, OldFormat, NewFormat)
```

Parameter	Description
OldTime	Enter a valid time string. The system assumes your entry is in the time format specified in the OldFormat parameter. The default is the current time.
OldFormat	Enter a valid time format that describes the OldTime parameter. The default is time format 1 (HH:MM:SS).
NewFormat	Enter a valid time format that describes the format you want the OldTime converted to. The default is time format 1 (HH:MM:SS).

Example

Here is an example:

(Assume *T1* is 01:30:05 pm.)

Function	Result	Explanation
Time2Time(“T1”, “2”, “1”)	13:30:05	Takes the time in T1 (which is in format 2) and converts it to format 1.

See also

Time Formats on page 80

TimeAdd

Use this function to add time to a given time and return the new time. The resulting time is returned in the same format.

Syntax

```
TimeAdd (Time, Format, Seconds, Minutes, Hours)
```

Parameter	Description
Time	Enter a valid time string. The system assumes your entry is in the time format specified in the Format parameter. The default is the current time.
Format	Enter a valid time format that describes the Time parameter. The default is time format 1 (HH:MM:SS).
Seconds	Enter the number of seconds to be added. The default is zero (0).
Minutes	Enter the number of minutes to be added. The default is zero (0).
Hours	Enter the number of hours to be added. The default is zero (0).

Example

Here is an example:

(Assume the current time is 1:20:03 pm.)

Function	Result	Explanation
TimeAdd(, , “10”, “20”, “3”)	4:40:13	Defaults to the current time and adds 3 hours, 20 minutes, and 10 seconds. Returns the result in the same format.

See also

Time Formats on page 80

TimeZone

Use this function to return the system's time zone setting or to make sure a time zone is valid.

Syntax

```
TimeZone (TimeZone)
```

Parameter	Description
TimeZone	(Optional) If you include a time zone string, the system makes sure that string is valid. If it is invalid, the system returns an empty string. The default is to return the system's current time zone setting.

Example

Here are some examples:

This example returns the system time zone, such as *America/New_York*:

```
T1 = TimeZone()
```

This example checks to see if a time zone string, such as *Europe/London*, is valid:

```
T1 = 'Europe/London'
T2 = TimeZone(T1)
if (T2 = '') then
Print_It(T1 & 'is not a valid time zone string')
else
Print_It(T1 & 'is a valid time zone string')
end
```

See also

TimeZone2TimeZone on page 410

Time Functions on page 80

Using the Time Zone Functions on page 81

ICU Time Zones on page 82

TIMEZONE2TIMEZONE

Use this function to convert date and time values from one geographic region into date and time values that are local to another geographic region. The function will also adjust for daylight savings time as needed.

Syntax

```
TimeZone2TimeZone (PrefixName, TimeZone, NewTimeZone)
```

Parameter	Description
PrefixName	Enter the prefix name associated with variables that will be used to hold date and time settings. Here are some examples: PrefixName.day PrefixName.month PrefixName.year PrefixName.hour PrefixName.minutes PrefixName.seconds
TimeZone	(Optional) Enter the time zone used for the PrefixName variables. If you enter an invalid time zone string, the system returns a value of zero (0) and sets variables associated with the PrefixName to zero (0). The default is to return the system's current time zone setting.
NewTimeZone	(Optional). Enter the time zone by which you want to adjust the values in the PrefixName variables. If you enter an invalid time zone string, the system returns a value of zero (0) and sets variables associated with the PrefixName to zero (0). The default is to return the system's current time zone setting.

If you define these variables, the system uses the PrefixName and time you specified and converts that time to the equivalent time in the location you specified via the NewTimeZone parameter.

If you do not define these variables, the system creates these variables based on the PrefixName you entered and assigns values into these variables based on the current date and time.

If there are no errors, the system returns a non-zero value.

Example

Here are some examples:

This example creates date and time variables using *tz* as a prefix (tz.day, tz.month, tz.year, tz.hour, tz.minute, tz.second) and stores the current date and time values based on the system's time zone:

```
TimeZone2TimeZone('tz', ,)
Print_It('Date:' & Date(, tz.day, tz.month, tz.year))
Print_It('Time:' & Time(, tz.hour, tz.minute, tz.second))
```

This example converts date and time variables (tz.xxxx) that use the system's time zone into GMT date and time:

```
TimeZone2TimeZone('tz', , 'GMT')
Print_It('GMT Date:' & Date(, tz.day, tz.month, tz.year))
Print_It('GMT Time:' & Time(, tz.hour, tz.minute, tz.second))
```

This example converts a current America/New_York date and time into an Australia/Melbourne date and time:

```
tz.day = ''
tz.month = ''
tz.year = ''
tz.hour = ''
tz.minute = ''
tz.second = ''
if (TimeZone2TimeZone('tz', 'America/New_York', 'Australia/
Melbourne')) then
Print_It('Australia/Melbourne Date:' & Date(, tz.day, tz.month,
tz.year))
Print_It('Australia/Melbourne Time:' & Time(, tz.hour, tz.minute,
tz.second))
else
Print_it('Error calling TimeZone2TimeZone')
end
```

See also

TimeZone on page 409

Time Functions on page 80

Using the Time Zone Functions on page 81

ICU Time Zones on page 82

TOTALPAGES

Use this function to return the number of pages that will print for a given recipient or for all recipients. A page is considered any *side* of paper that has a printable section for a recipient. A duplex sheet with front and back sections counts as two pages.

Syntax

```
TotalPages (Recipient)
```

Parameter	Description
Recipient	(Optional) If you include the Recipient parameter, the count only reflects the pages that print for that recipient. If you omit the Recipient parameter, the count includes all recipients.

The count considers copy-counts and reflects the total number of printed sides that will be referenced. A section may be empty (containing no text or discernible print objects) and still be designated to print. So, the count does not necessarily mean the pages will contain any real text.

Example

For example, assume you have a one-page document that has two recipients. Recipient1 gets one copy, while Recipient2 gets two copies.

With this command:

```
TotalPages(“Recipient1”)
```

The system returns one (1) as the page count

With this command:

```
TotalPages(“Recipient2”)
```

The system returns two (2) as the page count, since the one-page document will be printed twice. if you omit the Recipient parameter, the system returns three (3) as the page count.

NOTE: The count reflects when the function is called. The system cannot predict whether banner pages will be created or whether additional formatting or data entry will add or remove pages. Make sure you do not call this function until all page items have been created and formatted.

See also

TotalSheets on page 413

Documaker Workstation Functions on page 59

TOTALSHEETS

Use this function to return the total number of sheets of paper that will print for a recipient. A sheet is considered a physical piece of paper that may have print on one or both sides. Therefore a duplex sheet with a front and back sections will count as one sheet.

NOTE: Although the TotalSheets function does take duplex options into consideration, it has no knowledge of whether you will actually print to a printer that supports duplex commands. The count reflects what the document defines, not what the printer will support

Syntax

```
TotalSheets (Recipient)
```

Parameter	Description
Recipient	(Optional) If you include the Recipient parameter, the count only reflects the sheets that print for that recipient. If you omit the Recipient parameter, the count includes all recipients.

The count takes into consideration recipient copy counts and duplex options. A section may be empty (containing no text or discernible print objects) and still be designated to print. So, the count does not necessarily mean that the sheets will contain any real text.

Example

For example, assume you have a two-page document that is duplexed (prints front and back). Recipient1 gets one copy, while Recipient2 gets two copies.

With this command:

```
TotalSheets(Recipient1)
```

The system returns one (1) as the sheet count.

With this command:

```
TotalSheets(Recipient2)
```

The system returns two (2) as the sheet count, since the two-page document will be printed twice. if you omit the Recipient parameter, the system returns three (3) as the sheet count.

NOTE: The count reflects when the function is called. The system cannot predict whether banner pages will be created or whether additional formatting or data entry will add or remove pages. Make sure you do not call this function until all page items have been created and formatted.

See also

TotalPages on page 412

Documaker Workstation Functions on page 59

TRIGGERFORM

Use this function/procedure to add a form to your form set and still have the form's section triggers evaluated.

For instance, you can use this function when users add forms to a policy and want endorsements to print in a sequence unique to each transaction. The RunTriggers rule evaluates each form level trigger and adds the form to the set using the order specified by the forms list in the MRL.

If you use an import file rule, the system adds the forms in the order specified by the associated input. If you have a situation where you want to control the form addition, but the data is not an import style file, you may want to use this function to do the form additions.

This function adds a form to the form set and evaluates the form's section triggers. Section trigger and recipient processing are then performed as if the form had been triggered by the RunTriggers rule.

For each triggered form, the form's section triggers are evaluated. If a section trigger returns a zero (0) count, the system removes the section from the form. If the section trigger returns a multiple count, the system inserts additional sections into the form until the section count matches the trigger count.

If the TriggerForm function is executed before the standard section processing rule (RULStandardImageProc), the standard field rules are also be executed on the inserted sections.

Here's an example. Assume these entries are in your AFGJOB.JDT file:

```
;PreTransDAL;;TriggerForm("Myltr1");
;RunTriggers;;
;PreTransDAL;;TriggerForm("Myltr2");
;PreTransDAL;;TriggerForm("Myltr3","Myltr1");
;PreTransDAL;;TriggerForm("Myltr4",,"LB2");
```

This form	Is added
Myltr1	To the beginning of the form set (before any forms triggered by the RunTriggers rule)
Myltr2	After any forms triggered by the RunTriggers rule
Myltr3	After the first occurrence of Myltr1
Myltr4	To the group named LB2

NOTE: You can also use the AddForm procedure to add a form to a form set. The difference is TriggerForm also tells the system to evaluate the form's section triggers, where AddForm does not.

Syntax

```
TriggerForm (Form, Insert, Group)
```

Parameter	Description
Form	Enter the name of a form in the specified group.
Insert	Enter the name of a form *after* which the new form should be inserted. The default is to append after the last form in the group.
Group	Enter the name of a group to contain the specified form. The default is the current group.

The system optionally returns one (1) on success or zero (0) on failure.

This procedure adds a copy of the specified form to the document set. The form named must be a valid form in the specified group. You cannot add a form defined for one group into another group. You can specify the name of the form you want to insert using the occurrence indicator.

If you include the Group parameter, make sure it references a group defined in the Application Definition (BDF) file.

Use this procedure with resources created using Documaker Studio and processed via the GenData program. If you use this procedure outside of GenData, such as with GenPrint or the entry system, or if you use it with legacy resources, the TriggerForm function will not cause the system to evaluate section triggers. In that scenario, you simply get the functionality of the AddForm procedure.

Example

Here are some examples:

Procedure	Result	Explanation
TriggerForm("FormA")	1 or 0	Add FormA after the last form in the current group and evaluate FormA's section triggers.
TriggerForm("FormA", "FormA\1", GRP")	1 or 0	Add FormA after the first occurrence of that form within the named group and evaluate FormA's section triggers.

See also

AddForm on page 119

AddForm_Propagate on page 120

CopyForm on page 174

DupForm on page 228

WIP Functions on page 88

TriggerFormName

If you are using DAL scripts during Documaker Server SetRecip trigger processing, use this function to return the form name of the current SetRecipTb entry being processed.

Syntax

```
TriggerFormName ( )
```

There are no parameters for this function.

Example

Here is an example:

Assume your SETRECIPTB.DAT file has the following entries and a loaded DAL library file contains the DAL sub-routine function, *ILDSChk*. The forms are triggered if the conditions in the DAL script are met.

Here is an example of the SETRECIPTB.DAT file:

```
…
;Docu;CP;ILDS498;S004H;XLC;Agent(1);;0;0;0;1;;DALTrigger;ILDSChk;
;Docu;CP;ILDS598;S004L;XLC;Agent(1);;0;0;0;1;;DALTrigger;ILDSChk;
…
```

Here is an example of the DAL library file:

```
***  If driver's age, insured state, and form name are the specified
***  conditions then trigger the form.

BeginSub ILDSChk

trig_f_name  = TriggerFormName()
If trig_f_name = "ILDS498" AND \
        ?("driver_age") <= 25 AND \
        ?("insure_st")  = "CA" Then
             Return(1)

   ElseIf trig_f_name = "ILDS598" AND \
        ?("driver_age") > 25
        ?("insure_st")  = "FL" Then
             Return(1)

         Else
             Return(0)
End

EndSub
```

See also

TriggerImageName on page 417

TriggerRecsPerOvFlw on page 418

Documaker Server Functions on page 58

TriggerImageName

If you are using DAL scripts during Documaker Server SetRecip trigger processing, use this function to return the section (FAP file) name of the current SetRecipTb entry being processed.

Syntax

```
TriggerImageName ( )
```

There are no parameters for the function.

Example

Here is an example:

Assume your SETRECIPTB.DAT file has the following entries and a loaded DAL library file contains the DAL sub-routine function, *ILDSChk*. The forms are triggered if the conditions in the DAL script are met.

Here is an example of the SETRECIPTB.DAT file:

```
…
;Docu;CP;ILDS498;S004H;XLC;Agent(1);;0;0;0;1;;DALTrigger;ILDSChk;
;Docu;CP;ILDS598;S004L;XLC;Agent(1);;0;0;0;1;;DALTrigger;ILDSChk;
…
```

Here is an example of the DAL library file:

```
***  If driver's age, insured state, and section name are the
specified
***  conditions then trigger the section.

BeginSub ILDSChk

trig_f_name  = TriggerImageName()
If trig_f_name = "S004H" AND \
        ?("driver_age") <= 25 AND \
        ?("insure_st")  = "CA" Then
             Return(1)

   ElseIf trig_f_name = "S00L" AND \
        ?("driver_age") > 25
        ?("insure_st")  = "FL" Then
             Return(1)

         Else
             Return(0)
End

EndSub
```

See also

TriggerFormName on page 416

TriggerRecsPerOvFlw on page 418

Documaker Server Functions on page 58

TRIGGERRECSPEROVFLW

Use this function to retrieve the number of records per overflow section value which is stored in the SETRCPTBL.DAT entry being processed. Depending on the current trigger, this integer value can be the overflow record count for a form or section.

NOTE: This is only applicable in Documaker Server processing during DAL trigger processing.

Syntax

```
TriggerRecsPerOvFlw ( )
```

There are no parameters for this function.

Example

Assume you have the following entry in the SETRCPTBL.DAT file for the form trigger being processed. Also assume there are 30 records in the extract file that match the search mask.

```
;RP10;CIS;qa_f1550;;;Customer(1);;1,M;25;0;1;;DALTrigger;FEATURE155
0;
```

Here is an example:

```
BeginSub Feature1550
    #rec = CountRec("1,Feature1550,31,Data")
    #remaining = MOD(#rec, TriggerRecsPerOvFlw( ))
    While(#remaining > 0)
*       write addition records
        Write_fm( )
    #mod -= 1
    Wend
    Return(#rec)
EndSub
```

In this example, the TriggerRecsPerOvFlw function, returns a records per overflow section value of 25, which is used in the MOD function.

See also

MOD on page 320

TriggerFormName on page 416

TriggerImageName on page 417

Documaker Server Functions on page 58

TRIM

Use this function to remove leading and/or trailing spaces from a given string. The integer parameter determines whether spaces on the left, right, or both ends are to be removed. The resulting string is returned.

Syntax

```
Trim (String, Integer)
```

Parameter	Description
String	Enter a valid string. The default is the value of the current field.
Integer	Choose from these options: 0 - remove trailing spaces 1 - remove leading spaces 2 - remove leading and trailing spaces The default is two (2).

The system removes leading and trailing spaces from the string specified in parameter one. The Integer parameter determines which spaces are removed.

Example

Here are some examples:

(Assume the current field contains the text "   Your Name")

Function	Result	Explanation
Trim (" Value ")	"Value"	Defaults to trim leading and trailing spaces.
Trim (" Value ", 0)	" Value"	Removes trailing spaces.
Trim( )	"Your Name"	Use current field and remove leading and trailing spaces. See the note below.

NOTE: During field entry, the system automatically removes trailing spaces from values entered by the user. Only variables assigned during DAL scripts are likely to have trailing spaces.

See also

String Functions on page 78

Upper

Use this function to convert all characters to uppercase and return the result.

Syntax

```
Upper (String, Length)
```

Parameter	Description
String	Enter a valid string. The default is the value of the current field.
Length	Enter the length of the output. The default is the length of the current field.

If the length specified in the Length parameter is longer than the string, the result is the length you specified. If the specified length is less than the string, the length of the string is used. The system does not truncate the string.

Example

Here are some examples:

(Assume the current field contains the text *Your Name*.)

Function	Result	Explanation
Upper ( )	"YOUR NAME"	Defaults to the current field.
Upper (, 15)	"YOUR NAME "	Defaults to the current field and increases the length of the field to 15.
Upper ("Street Address")	"STREET ADDRESS"	Uppercases the specified string.

See also

String Functions on page 78

Lower on page 305

UniqueString

Use this function to return a 45-character globally unique string.

Syntax

```
UniqueString ( )
```

There are no parameters for this function.

Keep in mind...

- These print drivers are supported: PCL5, PCL6, PST, MET, AFP, PDF, HTML, and RTF.
- These print drivers *are not* supported: EPT, MDR, and GDI.
- All platforms are supported, but note that while UniqueString is supported on z/OS, z/OS does not support PDF or long file names, so the PDF example does not apply to z/OS.
- Both multi- and single-step processing are supported.

The only DAL function actually involved in splitting the print stream is BreakBatch. The others make it easier to implement this functionality. For example, since you need to name the new print stream, you use the SetDeviceName procedure. To find the name of the current device, you use the DeviceName function. If you need to create unique file names, you can use the UniqueString function.

NOTE: While you can call all of these DAL functions in Documaker Server or Entry, the BreakBatch and SetDeviceName functions are not applicable in Entry since it does not use the batch printing engine. DeviceName and UniqueString are applicable to both Entry and Documaker Server.

Example

Here is an example:

```
DataPath = GetINIString(,"Data","DataPath")
Drive = FileDrive(DataPath)
Path = FilePath(DataPath)
UniqueID = UniqueString()
Outputname = FullFileName(Drive,Path,UniqueID,".PDF")
SetDeviceName(Outputname)
```

See also

Miscellaneous Functions on page 73

BreakBatch on page 158

DeviceName on page 217

SetDeviceName on page 373

UserID

Use this function to return the user ID used to log on to the Entry module.

Syntax

```
UserID ( )
```

There are no parameters for this function.

This function is only useful if the system is set up to require user IDs.

Example

Here are some examples:

(Assume the current user is TOMJ.)

Function	Result	Explanation
result = UserID( )	TOMJ	Identifies the current user ID as TOMJ.
SetFld (UserID ( ), "MyField")	TOMJ	First UserID determines that the current user ID is TOMJ, then the field named *MyField* is assigned the value TOMJ by the SetFld procedure.

See also

WIP Functions on page 88

UserLvl on page 423

UserLvl

Use this function to get the currently logged in user's access rights level. The value returned is in the range 0-9. Zero represents the highest level and nine represents the lowest level. Access rights levels are specific to each system implementation.

Syntax

```
UserLvl ( )
```

There are no parameters for this function.

This function is only useful if the system is set up to require user IDs and user rights.

Example

Here is an example:

(Assume the current user is TOMJ with an access rights level of 7.)

Function	Result	Explanation
#result=UserLvl ( )	7	Determines that TOMJ's user rights are 7 and returns a 7.

IF (UserLvl( ) !=0)
MSG(USERID( ),
"Remember to get a supervisor to approve this transaction.");
END;
	TOMJ
	Remember to get a supervisor to approve this transaction.
		First UserLvl determines that TOMJ's rights level does not equal zero (0). Then the MSG procedure creates a window and displays the given message along with the current user ID (TOMJ) returned by the UserID function.

See also

WIP Functions on page 88

UserID on page 422

WEEKDAY

Use this function to determine the day of the week in a given date and return the value as a number.

Syntax

```
WeekDay (Date, Format, Locale)
```

Parameter	Description
Date	Enter a valid date string. The system assumes your entry is in the format specified by the Format parameter. The default is the current date.
Format	Enter a valid date format that describes the format used by your entry in the Date parameter. The default is date format 1 (MM/DD/YY).
Locale	(Optional) Enter the locale code. If you omit this parameter, the system checks the Locale INI option. If the Locale INI option offers no value, the system defaults to USD (United States/English).

The system returns the number of the day of the week, from 1 to 7, as shown here:

Number	Day of the week
1	Sunday
2	Monday
3	Tuesday
4	Wednesday
5	Thursday
6	Friday
7	Saturday

WeekDay is most often used with the DayName function. The DayName function extracts the name of the day of the week from a given date.

Example

Here are some examples:

(Assume the current date is Wednesday, July 5, 2009.)

Function	Result	Explanation
WeekDay ( )	4	Defaults to the current date.
Datestring = DateAdd(, , 1); WeekDay(datestring)	5	First the DateAdd function adds one day to the current date, resulting in a date of Thursday, July 6, 2009. Then WeekDay returns 5, which corresponds to Thursday.

See also

Date Functions on page 51

Locales on page 55

Using INI Options on page 8

Date Formats on page 52

DateAdd on page 184

DayName on page 189

WhatForm

Use this function to return the name of the form that includes the item you searched for. Having the name of the form lets you manipulate that object using other DAL functions, which may require its name.

Syntax

```
WhatForm (Field, Section, Form, Group)
```

Parameter	Description
Field	Enter the name of the field. The default is the current field.
Section	Enter the name of the section. The default is the current section.
Form	Enter the name of the form. The default is the current form.
Group	Enter the name of a group to contain the specified form. The default is the current group.

If nothing matches your criteria, the system returns a blank.

Keep in mind you can use an asterisk (*) as the object name to match parent objects. This lets you find objects without explicitly knowing the parent names.

NOTE: For optimal performance, minimize using wildcards (*) when searching for field, section, or form names.

Example

Here is an example:

Function	Result	Explanation
form = WhatForm("Total Field\3", , ,"*");	The name of the form or 0	Attempts to locate the third occurrence of a field in a form set and returns the name of the form that contains that field.

See also

WhatGroup on page 427

WhatImage on page 428

Name Functions on page 74

WHATGROUP

Use this function to return the name of the group that includes the item you searched for. Having the name of the form lets you manipulate that object using other DAL functions, which may require its name.

Syntax

```
WhatGroup (Field, Section, Form, Group)
```

Parameter	Description
Field	Enter the name of the field. The default is the current field.
Section	Enter the name of the section. The default is the current section.
Form	Enter the name of the form. The default is the current form.
Group	Enter the name of a group to contain the specified form. The default is the current group.

If nothing matches your criteria, the system returns a blank.

Keep in mind you can use an asterisk (*) as the object name to match parent objects. This lets you find objects without explicitly knowing the parent names.

NOTE: For optimal performance, minimize the use of wildcards (*) when searching for field, section, or form names.

Example

Here is an example:

Function	Result	Explanation
group = WhatGroup(, , "MyForm", "*");	The name of the form or 0	Attempts to locate the group name that contains a specific form.

See also

WhatForm on page 426

WhatImage on page 428

Name Functions on page 74

WhatImage

Use this function to return the name of the section that includes the item you searched for. Having the name of the form lets you manipulate that object using other DAL functions, which may require its name.

Syntax

```
WhatImage (Field, Section, Form, Group)
```

Parameter	Description
Field	Enter the name of the field. The default is the current field.
Section	Enter the name of the section. The default is the current section.
Form	Enter the name of the form. The default is the current form.
Group	Enter the name of a group to contain the specified form. The default is the current group.

If nothing matches your criteria, the system returns a blank.

Keep in mind you can use an asterisk (*) as the object name to match parent objects. This lets you find objects without explicitly knowing the parent names.

NOTE: For optimal performance, minimize the use of wildcards (*) when searching for field, section, or form names.

Example Here is an example:

Function	Result	Explanation
section = WhatSection("Total Field\12", , ,"*");	The name of the form or 0	Attempts to locate the twelfth occurrence of a field in a form set and returns the name of the section that contains that field.

See also WhatForm on page 426

WhatGroup on page 427

Name Functions on page 74

WIPExit

Use this procedure/function to close work-in-process.

Syntax

```
WIPExit (SaveFlag)
```

Parameter	Description
SaveFlag	Enter a positive number, such as one (1), to save and close the current form set. Enter zero (0) to close the form set without saving and exit WIP. The default is to save and close the current form set.

This procedure generates a message that tells the system to close the current form set.

Although control returns to the script after calling this procedure, the only statement that should be executed afterwards is a RETURN statement.

Example

Here are some examples:

Procedure	Result	Explanation
WIPExit(1)	Exits WIP and saves your work.	Work is saved with a valid positive flag.
WIPExit(0)	Exits WIP but does not save your work.	Work is not saved with a flag of zero.

See also WIP Functions on page 88

WIPFLD

Use this function to return the value of a database field from the current WIP record.

Syntax

```
WIPFld (WIPfield)
```

Parameter	Description
WIPfield	Enter the name of the field in the WIP record.

The system returns the value of an identified field within the current WIP record.

WIP records are only defined within the Entry system and are implementation specific. If a request is made for a field that is not part of the WIP record definition, the system returns an empty string.

Example

Here are some examples:

(Assume the current WIP record has a field named OrigUser which contains the string *David Harris*.)

Function	Result	Explanation
result = WIPFld (“OrigUser”)	David Harris	Determines that the current WIP record named OrigUser has the value David Harris and returns that value.
IF (WIPFld ('StatusCode') !='W') SetFld(“N/A”); END		If the current WIP record does not contain a StatusCode field that is equal to *W* the SetFld statement executes.

See also

WIP Functions on page 88

WIPKEY1

Use this function to return the value of the Key1 field from the current WIP record.

Syntax

```
WIPKey1 ( )
```

There are no parameters for this function.

The system returns the value of the Key1 field within the current WIP record known as the *Company* field in the insurance market. WIP records are only defined within the Entry module and are specific for each implementation.

This is a short-cut method for WIPFld("KEY1"), which would return the same value.

Example

Here are some examples:

(Assume the current WIP record contains a Key1 field with the value *Oracle*.)

Function	Result	Explanation
result = WIPKey1( )	Oracle	Determines the value contained in the WIP Key1 field and returns that value.
IF WIPKey1 ( ) = "Oracle" SetFld("N/A"); END	1 N/A	Determines that the Key1 field contains the value *Oracle*, then executes the SetFld procedure and places *N/A* in the current field. Also returns one (1) to indicate that the SetFld procedure was successful.

See also

WIP Functions on page 88

WIPFld on page 430

WIPKey2 on page 432

WIPKeyID on page 433

SetFld on page 377

WIPKEY2

Use this function to return the value of the Key2 field from the current WIP record.

Syntax

```
WIPKey2 ( )
```

There are no parameters for this function.

The system returns the description of the Key2 field in the current WIP record, known as the *Line of Business* field in the insurance market. WIP records are only defined within the entry system and are implementation specific.

This is a short-cut method for WIPFld("KEY2"), which would return the same value.

Example

Here are some examples:

(Assume the current WIP record contains a Key2 field with the value "Fire Insurance".)

Function	Result	Explanation
result = WIPKey2( )	Fire Insurance	Determines the value contained in the WIP Key2 field and returns that value.
IF WIPKey2 ( ) = "Oracle" SetFld("N/A"); END	Nothing	Determines that the Key2 field does not contain the value "Oracle"; therefore the SetFld procedure does not execute.

See also

WIP Functions on page 88

WIPFld on page 430

WIPKey1 on page 431

WIPKeyID on page 433

SetFld on page 377

WIPKeyID

Use this function to replace the value of the KeyID field from the current WIP record.

Syntax

```
WIPKeyID ( )
```

There are no parameters for this function.

The system returns the value of the KeyID field in the current WIP record, known as the *Policy Number* field in the insurance market. WIP records are only defined in the Documaker and are implementation specific.

This is a short-cut method for the WIPFld("KEYID") function, which would return the same value.

Example

Here are some examples:

(Assume the current WIP record contains a KeyID field with the value "1300".)

Function	Result	Explanation
result = WIPKeyID( )	1300	Determines the value contained in the WIP KeyID field and returns that value.
IF LEFT(WIPKeyID ( ), 3) > 100 SETFLD("N/A"); END	1 N/A	Finds the KeyID field value. Then determines that the three left most characters in the KeyID field are greater than 100. Executes the SetFld procedure and places "N/A" in the current field. Also returns one (1) to indicate that the SetFld procedure was successful.

See also

WIP Functions on page 88

WIPFld on page 430

WIPKey1 on page 431

WIPKey2 on page 432

SetFld on page 377

XMLAttrName

Use this function to return the name of the current attribute pointed to by the XMLFirstAttrib and XMLNextAttrib functions.

Syntax

```
XMLAttrName (%XMLTree)
```

Parameter	Description
%XMLTree	Enter a list type DAL variable that passes the XML tree handle.

The system returns the name of the current attribute pointed to by the XMLFirstAttrib and XMLNextAttrib functions.

Example

This example returns the second attribute name of the first form in the list.

```
aStr="Attribute not found!";
%xXMLTree=LoadXMLList("test.xml");
%XMLTree=XMLFind(%xXMLTree,"Forms","Form");
#rc=XMLFirst(%XMLTree);
#rc=XMLFirstAttrib(%XMLTree);
#rc=XMLNextAttrib(%XMLTree);
if #rc > 0
aStr=XMLAttrName(%XMLTree);
end
#rt=DestroyList(%xXMLTree);
return(aStr);
```

See also

XML Functions on page 89

XMLFirstAttrib on page 438

XMLNextAttrib on page 443

XMLAttrValue

Use this function to return the value of the current attribute pointed to by the XMLFirstAttrib and XMLNextAttrib functions. This function is similar to the XMLAttrName function.

Syntax

```
XMLAttrValue (%XMLTree)
```

Parameter	Description
%XMLTree	Enter a list type DAL variable that passes the XML tree handle.

The system returns the value of the current attribute pointed to by the XMLFirstAttrib and XMLNextAttrib functions.

Example

This example returns the second attribute name of the first form in the list.

```
aStr="Attribute not found!";
%xXMLTree=LoadXMLList("test.xml");
%XMLTree=XMLFind(%xXMLTree,"Forms","Form");
#rc=XMLFirst(%XMLTree);
#rc=XMLFirstAttrib(%XMLTree);
#rc=XMLNextAttrib(%XMLTree);
if #rc > 0
aStr=XMLAttrValue(%XMLTree);
end
#rt=DestroyList(%xXMLTree);
return(aStr);
```

See also

XML Functions on page 89

XMLAttrName on page 434

XMLFirstAttrib on page 438

XMLNextAttrib on page 443

XMLFIND

Use this function to locate the XML path from the extracted XML tree and return a list of matched elements to either a:

- List type DAL variable, or a
- Matched text to a string type DAL variable

The result depends on the search request.

Syntax

```
XMLFind (%xXMLTree, SrchNode, XPath)
```

Parameter	Description
%xXMLTree	A list type DAL variable which is passed from either the XMLFileExtract rule or the LoadXMLList function. You can use the predefined %extract variable as a parameter here, as discussed in scenario 1.
SrchNode	A string type DAL variable that passes a node name from which the search starts. If you omit this parameter, the search starts from the root of the XML tree.
XPath	A string type DAL variable that passes the XML location. If you omit the second parameter, the search starts from the root of the XML tree.

The system returns a list type or a string type DAL variable.

Example

This example returns text from the last element in the list.

```
aStr="Text not found!";
%xXMLTree=LoadXMLList("test.xml");
%XMLTree=XMLFind(%xXMLTree,"Forms","Form[text()]");
#rc=XMLFirst(%XMLTree);
loop:
    if #rc=0
        goto endloop:
    end
    aStr=XMLGetCurName(%XMLTree);
    #rc=XMLNext(%XMLTree);
    goto loop:
endloop:
#rc=DestroyList(%xXMLTree);
return(aStr);
```

See also

XML Functions on page 89

Scenario 1 on page 90

Scenario 2 on page 90

XMLFirst

Use this function to set the current pointer to the first element in the specified list.

Syntax

```
XMLFirst (%XMLTree)
```

Parameter	Description
%XMLTree	A list type DAL variable. This variable can be either an XML tree or a list of extracted elements.

The system returns one (1) for success or zero (0) for failure.

Example

This example returns text from the last element in the list.

```
aStr="Text not found!";
%xXMLTree=LoadXMLList("test.xml");
%XMLTree=XMLFind(%xXMLTree,"Forms","Form[text()]");
#rc=XMLFirst(%XMLTree);
loop:
    if #rc=0
        goto endloop:
    end
    aStr=XMLGetCurName(%XMLTree);
    #rc=XMLNext(%XMLTree);
    goto loop:
endloop:
#rc=DestroyList(%xXMLTree);
return(aStr);
```

See also XML Functions on page 89

XMLFirstAttrib

Use this function to set the current pointer to the first element in the list you specify.

Syntax

```
XMLFirstAttrib (%XMLTree)
```

Parameter	Description
%XMLTree	Enter a list type DAL variable. You can enter either an XML tree or a list of extracted elements.

This function sets the attribute pointer to the first attribute for the current element in the element list or to the first attribute element in the attribute list.

If you input an element list, use these functions to retrieve the attribute name and value:

- XMLAttrName
- XMLAttrValue

If you input an attribute list, use these functions to retrieve attribute name and value:

- XMLNthAttrName
- XMLNthAttrValue

The system returns one (1) for success or zero (0) for failure.

Example

This example returns text from the last element in the list.

```
aStr="Text not found!";
%xXMLTree=LoadXMLList("test.xml");
%XMLTree=XMLFind(%xXMLTree,"Forms","Form[text()]");
#rc=XMLFirst(%XMLTree);
loop:
    if #rc=0
        goto endloop:
    end
    aStr=XMLGetCurName(%XMLTree);
    #rc=XMLNext(%XMLTree);
    goto loop:
endloop:
#rc=DestroyList(%xXMLTree);
return(aStr);
```

See also

XML Functions on page 89

XMLAttrName on page 434

XMLAttrValue on page 435

XMLNthAttrName on page 445

XMLNthAttrValue on page 446

XMLFirstText

Use this function to set the current text to be the first text element in the XML search list and then retrieve that text.

Syntax

```
XMLFirstText (List)
```

Parameter	Description
List	Enter the name of the list.

Example

Here is an example:

```
Mystring = XMLFirstText(List)
```

See also

XML Functions on page 89

XMLGetCurName

Use this function to get the name from the current element. This function is similar to the XMLGetCurText function.

Syntax

```
XMLGetCurName (%XMLTree)
```

Parameter	Description
%XMLTree	A list type DAL variable. This variable can be either an XML tree or a list of extracted elements.

The system returns the element name from the current element. The return value is a string type DAL variable.

Example

This example returns text from the last element in the list.

```
aStr="Text not found!";
%xXMLTree=LoadXMLList("test.xml");
%XMLTree=XMLFind(%xXMLTree,"Forms","Form[text()]");
#rc=XMLFirst(%XMLTree);
loop:
    if #rc=0
        goto endloop:
    end
    aStr=XMLGetCurName(%XMLTree);
    #rc=XMLNext(%XMLTree);
    goto loop:
endloop:
#rc=DestroyList(%xXMLTree);
return(aStr);
```

See also

XML Functions on page 89

XMLFirst on page 437

XMLGetCurText on page 441

XMLGetCurText

Use this function to get the text from the current element. This function is similar to the XMLGetCurName function.

Syntax

```
XMLGetCurText (%XMLTree)
```

Parameter	Description
%XMLTree	A list type DAL variable. This variable can be either an XML tree or a list of extracted elements.

The system returns the text from the current element. The return value is a string type DAL variable.

Example

This example returns text from the last element in the list.

```
aStr="Text not found!";
%xXMLTree=LoadXMLList("test.xml");
%XMLTree=XMLFind(%xXMLTree,"Forms","Form[text()]");
#rc=XMLFirst(%XMLTree);
loop:
    if #rc=0
        goto endloop:
    end
    aStr=XMLGetCurText(%XMLTree);
    #rc=XMLNext(%XMLTree);
    goto loop:
endloop:
#rc=DestroyList(%xXMLTree);
return(aStr);
```

See also

XML Functions on page 89

XMLFirst on page 437

XMLGetCurName on page 440

XMLNext

Use this function to set the current pointer to the next node or element in the specified list. This function is similar to the XMLFirst function.

Syntax

```
XMLNext (%XMLTree)
```

Parameter	Description
%XMLTree	A list type DAL variable. This variable can be either an XML tree or a list of extracted elements.

The system sets the current pointer to the next node or element in the list you specified list and returns one (1) for success or zero (0) for failure.

Example

This example returns text from the last element in the list.

```
aStr="Text not found!";
%xXMLTree=LoadXMLList("test.xml");
%XMLTree=XMLFind(%xXMLTree,"Forms","Form[text()]");
#rc=XMLFirst(%XMLTree);
loop:
    if #rc=0
        goto endloop:
    end
    aStr=XMLGetCurName(%XMLTree);
    #rc=XMLNext(%XMLTree);
    goto loop:
endloop:
#rc=DestroyList(%xXMLTree);
return(aStr);
```

See also

XML Functions on page 89

XMLFirst on page 437

XMLNEXTATTRIB

Use this function to set the current pointer to the next element in the list you specify. This function is similar to the XMLFirstAttrib function.

Syntax

```
XMLNextAttrib (%XMLTree)
```

Parameter	Description
%XMLTree	A list type DAL variable. This variable can be either an XML tree or a list of extracted elements.

This function sets the current attribute pointer to the next attribute for the current element in the list or to the next attribute element in the attribute list.

If you input an element list, use these functions to retrieve the attribute name and value:

- XMLAttrName
- XMLAttrValue

If you input an attribute list, use these functions to retrieve attribute name and value:

- XMLNthAttrName
- XMLNthAttrValue

The system returns one (1) for success or zero (0) for failure.

Example

This example returns text from the last element in the list.

```
aStr="Text not found!";
%xXMLTree=LoadXMLList("test.xml");
%XMLTree=XMLFind(%xXMLTree,"Forms","Form[text()]");
#rc=XMLFirst(%XMLTree);
loop:
    if #rc=0
        goto endloop:
    end
    aStr=XMLGetCurName(%XMLTree);
    #rc=XMLNext(%XMLTree);
    goto loop:
endloop:
#rc=DestroyList(%xXMLTree);
return(aStr);
```

See also

XML Functions on page 89

XMLAttrName on page 434

XMLAttrValue on page 435

XMLNthAttrName on page 445

XMLNthAttrValue on page 446

XMLNextText

Use this function to retrieve the next text element in the XML search list.

Syntax

```
XMLNextText (List)
```

Parameter	Description
List	Enter the name of the list.

Example

Here is an example:

```
Mystring = XMLNextText(List);
```

See also

XML Functions on page 89

XMLNthAttrName

Use this function to return the nth attribute name, as indicated by an index number you specify.

Syntax

```
XMLNthAttrValue (%XMLTree,#Index)
```

Parameter	Description
%XMLTree	A list type DAL variable that passes a name list.
#Index	A integer type DAL variable that passes an index number.

The system returns the nth attribute name indicated by the index number.

Example

In this example, the XMLFind function returns a list of attributes and the XMLNthAttrName function returns the name of the first attribute in the list.

```
aStr="Attribute not found!";
%xXMLTree=LoadXMLList("test.xml");
%XMLTree=XMLFind(%xXMLTree,"Forms","Form/@*");
aStr=XMLNthAttrName(%XMLTree, 1);
end
#rt=DestroyList(%xXMLTree);
return(aStr);
```

See also

XML Functions on page 89

XMLFind on page 436

XMLNthAttrValue

Use this function to return the nth attribute value, as indicated by an index number you specify. This function is similar to the XMLNthAttrName function.

Syntax

```
XMLNthAttrValue (%XMLTree,#Index)
```

Parameter	Description
%XMLTree	A list type DAL variable that passes a name list.
#Index	A integer type DAL variable that passes an index number.

The system returns the nth attribute value indicated by the index number.

Example

In this example, the XMLFind function returns a list of attributes and the XMLNthAttrValue function returns the name of the first attribute in the list.

```
aStr="Attribute not found!";
%xXMLTree=LoadXMLList("test.xml");
%XMLTree=XMLFind(%xXMLTree,"Forms","Form/@*");
aStr=XMLNthAttrValue(%XMLTree, 1);
end
#rt=DestroyList(%xXMLTree);
return(aStr);
```

See also

XML Functions on page 89

XMLNthAttrName on page 445

XMLNthText

Use this function to return the nth text value, as indicated by an index number you specify.

Syntax

```
XMLNthText (%XMLTree,#Index)
```

Parameter	Description
%XMLTree	A list type DAL variable that passes a name list.
#Index	A integer type DAL variable that passes an index number.

The system returns the nth text value indicated by the index number.

Example

In this example, the LoadXMLList function returns a text list and the XMLNthText function gets the first text.

```
AStr="Text not found";
%xXMLTree=LoadXMLList("test.xml");
%XMLTree=XMLFind(%xXMLTree,"Forms","Form/text()");
aStr=XMLNthtext(%XMLTree, 1);
#rt=DestroyList(%xXMLTree);
return(aStr);
```

See also

XML Functions on page 89

LoadXMLList on page 302

YEAR

Use this function to determine the number of the year in a given date and returns the value as a four-digit number.

Syntax

```
Year (Date, Format, Locale)
```

Parameter	Description
Date	Enter a valid date string. The system assumes your entry is in the date format specified by the Format parameter. The default is the current date.
Format	Enter a valid date format that describes your entry in the Date parameter. The default is date format 1, (MM/DD/YY).
Locale	(Optional) Enter the locale code. If you omit this parameter, the system checks the Locale INI option. If the Locale INI option offers no value, the system defaults to USD (United States/English).

The system determines the year portion of the given date based on the format you specified in the Format parameter.

Example

Here are some examples:

(Assume the current date is 07/01/09.)

Function	Result	Explanation
Year ( )	2009	Defaults to the current date and returns a four-digit year.
Year (“2-5-09”, “1-2”)	2009	Returns a four-digit year for the given date.

See also

Date Functions on page 51

Locales on page 55

Using INI Options on page 8

Date Formats on page 52

YearDay on page 449

YearDay

Use this function to determine the number of days from the beginning of the year (counting consecutively from January 1) to a given date and return the value as a number.

Syntax

```
YearDay (Date, Format, Locale)
```

Parameter	Description
Date	Enter a valid date string. The system assumes your entry is in the date format specified by the Format parameter. The default is the current date.
Format	Enter a valid date format that describes your entry in the Date parameter. The default is date format 1, (MM/DD/YY).
Locale	(Optional) Enter the locale code. If you omit this parameter, the system checks the Locale INI option. If the Locale INI option offers no value, the system defaults to USD (United States/English).

The system determines the day of the year portion of the given date based on the format you specified in the Format parameter.

Example

Here are some examples:

(Assume the current date is 07/01/09.)

Function	Result	Explanation
YearDay ( )	182	Defaults to the current date and returns the day of the year (counting consecutively from January 1).
YearDay (“7-1-08”)	183	Returns the day of the year (counting consecutively from January 1) for the given date. (Since 2008 is a leap year the number is one greater.)

See also

Date Functions on page 51

Locales on page 55

Date Formats on page 52

Year on page 448

Chapter 3

Keyword Reference

This chapter contains a reference, in alphabetical order, of all the keywords you can use in your DAL scripts.

See the Keyword Table on page 452 for a list of the keywords. See Grammar and Syntax on page 14 for more information on using DAL.

Keyword Table

This table lists each keyword and provides a description of the keyword. Click on the function name to jump to a discussion of that function.

Keyword	Description
And	Include AND to perform a logical conjunction on two Boolean expressions.
BeginSub	Include a BeginSub statement at the beginning of each subroutine in a DAL subroutine library.
Break	Use a Break statement to exit a While..Wend statement block.
Continue	Use a Continue statement to restart a While...Wend statement loop.
Else	Include an Else statement if you want to pass control to the statement that follows this keyword if the logical expression is false.
ElseIf	If the first logical expression is false, the first ELSEIF logical expression is evaluated.
End	Include an End statement to end an IF, ELSEIF, or ELSE statement
EndSub	Include a EndSub statement to end each subroutine in a DAL subroutine library.
Goto	Include a Goto statement to move to a specific location within a calculation.
If...End	Use IF statements to execute commands based on the occurrence of a given condition.
Or	Include OR to perform a logical disjunction on two Boolean expressions.
Return	Use a Return statement to tell the calculation to return with or without a value.
While...Wend	Use While...Wend statements to execute a series of statements, as long as a given condition is true.

And

When you have two Boolean expressions, use this keyword to have the system return True if both Boolean expressions evaluate to True. If either or expression evaluates to False, AND returns False.

Syntax

```
AND
```

There are no parameters for this keyword.

See also

Or on page 464

Keyword Table on page 452

BeginSub

Use this function to begin each subroutine in a DAL subroutine library.

Syntax

```
BeginSub Name
```

Once a DAL library is loaded, you can reference the scripts contained in the library by name. You do not have to CALL or CHAIN to the script.

Parameter	Description
Name	Enter the name of the subroutine.

BeginSub and EndSub must be paired per script. You must have a space between BeginSub and the script name.

Example

Here is an example:

```
BeginSub SCRIPT1

* This script returns #x set to 2 if #x was equal to 1 on enter.
IF (#x = 1) THEN #x = 2;
END;
RETURN (#x);
EndSub

BeginSub Script2
* This script returns a negative one if #y was equal to 5.
if(#y = 5) then Return (-1);
end;
EndSub
```

SCRIPT1 is the name of the first script and Script2 is the name of the second script.

NOTE: SCRIPT1 and Script2 are only names, you can use any name you want as long as the name is not a DAL reserved function, statement, or key word such as CALL, FIND, IF, and so on. You can mix case in script names.

See also

Keyword Table on page 452

EndSub on page 460

Break

Break statements provide a way to exit a While...Wend statement block.

Syntax

```
Break (Levels)
```

Parameter	Description
Levels	(Optional) The value you enter defines how many nested While...Wend statement blocks you want to terminate.If you omit this parameter, control passes to the statement following the next Wend statement encountered.

You can only include Break statements inside While...Wend statement blocks. Break statements transfer control to the statement following the Wend statement.

When used within nested While...Wend statements, you can include the Levels parameter to transfer control to the statement following the Wend level you specify.

Here are some examples. (Ellipses in the following examples represent additional statements, not shown.)

```
While(1)
    ...
    While (2)
        ...
        Break
    Wend
    ...
Wend
```

In this example, the Break statement only terminates the While...Wend which contains the statement. Control passes to the first (outside) While...Wend statement block.

Here is another example:

```
While(1)
    ...
    While (2)
        ...
        While(3)
            ...
            Break(3)
        Wend
        ...
    Wend
    ...
Wend
```

In this example, the Break(3) statement terminates all three While...Wend blocks that are active.

See also Keyword Table on page 452

Continue

Use Continue statements to restart a While...Wend statement loop.

Syntax

```
Continue
```

There are no parameters for this keyword.

Executing the Continue statement stops the current sequence of statement execution and restarts program flow at the beginning of the loop. This causes the While statement to retest the condition and, if true, execute the loop again.

Statements after the Continue keyword are not executed. Continue is often, but not always, activated by an IF test.

Example

Here is an example:

(Ellipses in the following examples represent additional statements, not shown.)

```
While(#x < 10)
    ...
    If (value)
        Continue
    End
    ...
Wend
```

See also

Keyword Table on page 452

ELSE

An IF Statement with an ELSE condition contains an alternative calculation. If the logical expression is false, control passes to the statement after the ELSE keyword.

Syntax

```
Else
```

There are no parameters for this keyword.

Example

Here is an example:

```
IF (@("FirstAmount") < 1000.00) THEN
    $FinalAmount = @("FirstAmount") * .05;
ELSE
    $FinalAmount = @("FirstAmount") + 10.00;
END;
RETURN ($FinalAmount)
```

If the value of the section variable field FirstAmount is less than 1000.00 then the amount is multiplied by .05 and entered in the target variable $FinalAmount.

If, however, the value of the section variable field FirstAmount is greater than or equal to 1000.00 then 10.00 is added to the amount and entered in the target variable $FinalAmount.

The value of the $FinalAmount field is then returned to the caller or section variable field.

Use of the keyword connector THEN is optional.

See also

Keyword Table on page 452

If...End on page 462

ELSEIF

An IF statement with an ELSEIF condition is the most complicated type of IF statement:

- If the first logical expression is true, the statement block after IF is executed until the first ELSEIF statement is reached.
- If the first logical expression is false, the first ELSEIF logical expression is evaluated.
- If the ELSEIF logical expression is true, the statement block from the ELSEIF to the next ELSEIF (or ELSE) is executed.
- If the ELSEIF statement is false, the next ELSEIF is evaluated.
- If all logical expressions are false, control passes to the ELSE block.
- If there is no ELSE block, control passes to the statement following the END keyword.

An ELSEIF statement is considered part of the same IF statement. Only one END keyword is needed to end an IF, ELSEIF, ELSE statement. IF statements can be nested inside other IF statements. A nested IF statement requires its own END keyword. A missing or mismatched keyword results in a runtime syntax error.

Example

Here is a sample IF statement with ELSEIF condition:

```
IF (@("FirstAmount") < 1000.00)
    $FinalAmount = @("FirstAmount") * .05;
ELSEIF @("FirstAmount") < 5000.00
    $FinalAmount = @("FirstAmount") * .03;
ELSEIF @("FirstAmount") < 10000.00
    $FinalAmount = @("FirstAmount") * .02;
ELSE
    $FinalAmount = @("FirstAmount") + 10.00;
END;
RETURN ($FinalAmount)
```

If the value of the section variable field FirstAmount is less than 1000.00 then the amount is multiplied by .05 and entered in the target variable $FinalAmount.

See also

Keyword Table on page 452

If...End on page 462

END

An IF statement is executed based on the occurrence of a certain condition. IF statements must begin with the keyword IF and terminate with the keyword END.

Syntax

```
End
```

There are no parameters for this keyword.

See also

Keyword Table on page 452

If...End on page 462

EndSub

Use this function to end each subroutine in a DAL subroutine library.

Syntax

```
EndSub
```

There are no parameters for this keyword.

BeginSub and EndSub must be paired for each script.

Example

Here is an example:

```
BeginSub SCRIPT1
* This script returns #x set to 2 if #x was equal to 1 on enter.
IF (#x = 1) THEN #x = 2;
END;
RETURN (#x);
EndSub

BeginSub Script2
* This script returns a negative one if #y was equal to 5.
if(#y = 5) then Return (-1);
end;
EndSub
```

Script1 is the name of the first script. Script2 is the name of the second one.

See also

Keyword Table on page 452

BeginSub on page 454

Goto

A GOTO statement moves to a specific location within a calculation. The location has been named with a label. (See Labels on page 22 for more information.)

Syntax

```
GoTo Location
```

Parameter	Description
Location	Specify the location you want to go to. For instance, enter the name of a section on a form.

A GOTO statement must begin with the keyword GOTO.

Example

Here is an example:

```
GOTO SECTION_ONE:
```

The control jumps to SECTION_ONE in a calculation.

The destination label can occur anywhere in the script containing the GOTO statement. If the label cannot be located in the script, a syntax error will be generated.

GOTO will support retrieving the label from a target variable.

Here is another example:

```
SECTION = "MY_LABEL:"
GOTO SECTION
```

Since the word following the GOTO statement does not contain a colon, the program will assume the label is contained in the target variable named. In this case, control will jump to the location of MY_LABEL in the current script.

See also

Keyword Table on page 452

IF...END

An IF statement is executed based on the occurrence of a certain condition. IF statements must begin with the keyword IF and terminate with the keyword END.

Components within IF statements can be connected with the keywords AND or OR. IF statements can have three forms: a simple IF statement, an IF statement with an ELSE condition, or an IF statement with an ELSEIF condition.

- Simple IF Statement

  A simple IF Statement contains a single statement block. The calculation is performed only if the logical expression is true. If the logical expression is false, control passes to the next statement after the END keyword. Here is an example:

  ```
  IF (@("FirstAmount") < 1000.00)  THEN
      $FinalAmount = @("FirstAmount") * .05;
  END;
  RETURN ($FinalAmount)
  ```

  CALCULATION: If the value of the section variable field FirstAmount is less than 1000.00 then the value is multiplied by .05 and entered in the target variable $FinalAmount. The value of the $FinalAmount target variable is then returned to the section variable field.

  The use of the keyword connector THEN is optional.

- IF Statement with ELSE Condition

  An IF Statement with an ELSE condition contains an alternative calculation. If the logical expression is false, control passes to the statement after the ELSE keyword.

  Here is an example:

  ```
  IF (@("FirstAmount") < 1000.00) THEN
      $FinalAmount = @("FirstAmount") * .05;
  ELSE
      $FinalAmount = @("FirstAmount") + 10.00;
  END;
  RETURN ($FinalAmount)
  ```

  CALCULATION: If the value of the section variable field FirstAmount is less than 1000.00 then the amount is multiplied by .05 and entered in the target variable $FinalAmount.

  If the value of the section variable field FirstAmount is greater than or equal to 1000.00 then 10.00 is added to the amount and entered in the target variable $FinalAmount.

  The value of the $FinalAmount field is then returned to the caller or section variable field.

  The use of the keyword connector THEN is optional.

- IF Statement with ELSEIF Condition

An IF statement with an ELSEIF condition is the most complicated type of IF statement. If the first logical expression is true, the statement block after IF is executed until the first ELSEIF statement is reached. If the first logical expression is false, the first ELSEIF logical expression is evaluated. If the ELSEIF logical expression is true, the statement block from the ELSEIF to the next ELSEIF (or ELSE) is executed. If the ELSEIF statement is false, the next ELSEIF is evaluated. If all logical expressions are false, control passes to the ELSE block. If there is no ELSE block, control passes to the statement following the END keyword.

An ELSEIF statement is considered part of the same IF statement. Only one END keyword is needed to end an IF, ELSEIF, or ELSE statement. IF statements can be nested inside other IF statements. A nested IF statement requires its own END keyword. A missing or mismatched keyword results in a runtime syntax error. Here is a sample IF statement with ELSEIF condition:

```
IF (@("FirstAmount") < 1000.00)
    $FinalAmount = @("FirstAmount") * .05;
ELSEIF @("FirstAmount") < 5000.00
    $FinalAmount = @("FirstAmount") * .03;
ELSEIF @("FirstAmount") < 10000.00
    $FinalAmount = @("FirstAmount") * .02;
ELSE
    $FinalAmount = @("FirstAmount") + 10.00;
END;
RETURN ($FinalAmount)
```

CALCULATION: If the value of the section variable field FirstAmount is less than 1000.00 then the amount is multiplied by .05 and entered in the target variable $FinalAmount.

If the value of the section variable field FirstAmount is greater than or equal to 1000.00 but less than 5000.00 then the amount is multiplied by .03 and entered in the target variable $FinalAmount.

If the value of the section variable field FirstAmount is greater than or equal to 5000.00 but less than 10000.00 then the amount is multiplied by .02 and entered in the target variable $FinalAmount.

If the value of the section variable field FirstAmount is greater than or equal to 10000.00 then 10.00 is added to the amount and entered in the target variable $FinalAmount.

The value of the $FinalAmount field is then returned to the caller or section variable field.

See also Keyword Table on page 452

Else on page 457

ElseIf on page 458

OR

When you have two Boolean expressions, use this keyword to have the system return True if either Boolean expression evaluates to True. If neither expression evaluates to True, OR returns False.

Syntax

```
OR
```

There are no parameters for this keyword.

See also

And on page 453

Keyword Table on page 452

Return

A RETURN statement directs the calculation to return with or without a value. A RETURN statement must begin with the keyword RETURN. A RETURN statement may return the result of the calculation to be placed in the field that initiated the script.

A RETURN statement is also used to return results to one calculation script from another. Using a CALL statement temporarily suspends the current script calculation and sends control to another script file. A RETURN statement sends control back to the original script which may then continue processing.

Here are some sample RETURN statements:

```
RETURN(@(“LAST_NAME”) & ', ' & @(“FIRST_NAME”) & “ ” &
@(“MIDDLE_INIT”))
```

RESULT: Takes the data in the section variable field LAST_NAME adds a comma; adds the data in the section variable field FIRST_NAME; adds the data in the section variable field MIDDLE_INIT and places this data in another section variable field.

```
RETURN (CALL('FirstFile'))
```

RESULT: Returns the result of the calculation generated by calling the script FirstFile.

See also

Keyword Table on page 452

Call on page 160

While...Wend

Use While...Wend statements to execute a series of statements, as long as a given condition is true.

Syntax

```
While condition
[statements]
Wend
```

Parameter	Description
Condition	Required. The condition is any expression that evaluates to true or false. False is assumed to be a zero value. Any non-zero value is assumed to be true.
Statements	One or more statements executed while the condition is true.

If condition is true, the statements within the While block are executed. When the Wend statement is encountered, control returns to the While statement and condition is again evaluated. If condition is still true, the process repeats. If it is false, execution resumes with the statement which follows the Wend statement.

You can nest While...Wend loops to any level. Each Wend matches the most recent While.

NOTE: Keep in mind that you can start an endless loop if you specify a condition that can never be satisfied. The system cannot syntactically detect an endless loop, so if you create one, the program will lock up and you will have to kill the program.

(Ellipses in the following examples represent additional statements, not shown.)

```
While(10 > #value)
    ...
    While (#new = 1)
        ...
    Wend
    ...
Wend
```

You do not have to use tabs to indent nested While...Wend statements. Tabs are used in these examples, to help identify statement blocks. You may want to also use tabs in your code to make the source easier to read.

See also Keyword Table on page 452

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

Y

Z