



Documaker

DAL Reference

version 11.2

Skywire Software, L.L.C.
3000 Internet Boulevard
Suite 200
Frisco, Texas 75034
www.skywiresoftware.com

Phone:	(U. S.)	972.377.1110
	(EMEA)	+44 (0) 1372 366 200
FAX:	(U. S.)	972.377.1109
	(EMEA)	+44 (0) 1372 366 201
Support:	(U. S.)	866.4SKYWIRE
	(EMEA)	+44 (0) 1372 366 222
		support@skywiresoftware.com

PUBLICATION COPYRIGHT NOTICE

Copyright © 2008 Skywire Software, L.L.C. All rights reserved.

Printed in the United States of America.

This publication contains proprietary information which is the property of Skywire Software or its subsidiaries. This publication may also be protected under the copyright and trade secret laws of other countries.

TRADEMARKS

Skywire® is a registered trademark of Skywire Software, L.L.C.

Docucorp®, its products (Docucreate™, Documaker™, Docupresentment™, Docusave®, Documanager™, Poweroffice®, Docutoolbox™, and Transall™), and its logo are trademarks or registered trademarks of Skywire Software or its subsidiaries.

The Docucorp product modules (Commcommander™, Docuflex®, Documerge®, Docugraph™, Docusolve®, Docuword™, Dynacomp®, DWSD™, DBL™, Freeform®, Grafxc commander™, Imagecreate™, I.R.I.S.™, MARS/NT™, Powermapping™, Printcommander®, Rulecommander™, Shuttle™, VLAM®, Virtual Library Access Method™, Template Technology™, and X/HP™ are trademarks of Skywire Software or its subsidiaries.

Skywire Software (or its subsidiaries) and Mynd Corporation are joint owners of the DAP™ and Document Automation Platform™ product trademarks.

Docuflex is based in part on the work of Jean-loup Gailly and Mark Adler.

Docuflex is based in part on the work of Sam Leffler and Silicon Graphic, Inc.

Copyright © 1988-1997 Sam Leffler.

Copyright © 1991-1997 Silicon Graphics, Inc.

Docuflex is based in part on the work of the Independent JPEG Group.

The Graphic Interchange Format© is the Copyright property of CompuServe Incorporated. GIFSM is a Service Mark property of CompuServe Incorporated.

Docuflex is based in part on the work of Graphics Server Technologies, L.P.

Copyright © 1988-2002 Graphics Server Technologies, L.P.

All other trademarks, registered trademarks, and service marks mentioned within this publication or its associated software are property of their respective owners.

SOFTWARE COPYRIGHT NOTICE AND COPY LIMITATIONS

Your license agreement with Skywire Software or its subsidiaries, authorizes the number of copies that can be made, if any, and the computer systems on which the software may be used. Any duplication or use of any Skywire Software (or its subsidiaries) software in whole or in part, other than as authorized in the license agreement, must be authorized in writing by an officer of Skywire Software or its subsidiaries.

PUBLICATION COPY LIMITATIONS

Licensed users of the Skywire Software (or its subsidiaries) software described in this publication are authorized to make additional hard copies of this publication, for internal use only, as long as the total number of copies does not exceed the total number of seats or licenses of the software purchased, and the licensee or customer complies with the terms and conditions of the License Agreement in effect for the software. Otherwise, no part of this publication may be copied, distributed, transmitted, transcribed, stored in a retrieval system, or translated into any human or computer language, in any form or by any means, electronic, mechanical, manual, or otherwise, without permission in writing by an officer of Skywire Software or its subsidiaries.

DISCLAIMER

The contents of this publication and the computer software it represents are subject to change without notice. Publication of this manual is not a commitment by Skywire Software or its subsidiaries to provide the features described. Neither Skywire Software nor its subsidiaries assume responsibility or liability for errors that may appear herein. Skywire Software and its subsidiaries reserve the right to revise this publication and to make changes in it from time to time without obligation of Skywire Software or its subsidiaries to notify any person or organization of such revision or changes.

The screens and other illustrations in this publication are meant to be representative, not exact duplicates, of those that appear on your monitor or printer.

Contents

Chapter 1, Using DAL

- 2 Introduction to DAL
- 3 Using the Field's Properties Window
- 4 Entering Calculations in External Files
 - 4 Formatting the Script
- 5 Creating a DAL Script Library
 - 6 Loading a DAL library
- 7 Executing a DAL Script from a Menu
- 8 Using INI Options
- 10 Using Built-In Functions
- 11 Using DAL with XML
 - 11 Scenario 1
 - 11 Scenario 2
- 12 XML Built-in Functions
 - 12 LoadXMLList
 - 12 DestroyList
 - 12 GetListElem
 - 12 IsXMLError
 - 13 XMLFind
 - 13 XMLFirst
 - 13 XMLNext
 - 13 XMLGetCurName
 - 14 XMLGetCurText
 - 14 XMLFirstText
 - 14 XMLNextText
 - 14 XMLFirstAttrib
 - 14 XMLNextAttrib
 - 15 XMLAttrName
 - 15 XMLAttrValue
 - 15 XMLNthText
 - 15 XMLNthAttrName
 - 16 XMLNthAttrValue

16	XML Path Locator
16	Axes
16	Function calls
16	Operators or signs
17	Expressions
18	Element list
18	Attribute list
18	Text list
18	Text string
19	Checking KeyID Entries
22	Grammar and Syntax
22	Assignment Statements
22	Target variable
23	Source expression
24	Form set variable fields
24	Target variables
24	Numeric constants
25	String constants
25	Operators
26	Punctuation
26	Execution order
28	Implicit conversion
29	Labels
30	Flow Control Statements
30	Keywords
30	RETURN statements
31	IF statements
33	GOTO statements
33	CALL statements
33	CHAIN statements
34	Using While...Wend Statements
34	Break statements
35	Continue statements
36	GOTO statements
36	BeginSub and EndSub
36	BeginSub
37	EndSub
38	Data Storage Statements
39	Testing DAL Scripts
40	Using the DAL Debugger in Documaker Workstation

- 41 Runtime Error Messages
- 43 DAL Script Examples
 - 43 Preparing AFP or Metacode print streams for Docusave
 - 43 Preparing PCL print streams for Docusave
 - 43 Preparing AFP print streams for IBM's OnDemand

Chapter 2, Function Reference

- 48 Overview
- 49 Bit/Binary Functions
- 50 Database Functions
 - 51 ODBC Handler
 - 52 DB2/2 Handler
 - 53 Creating a Database Handler for an Excel Database
 - 55 Associating Tables with Handlers
 - 56 Accessing Database Fields
 - 57 Setting Up Memory Tables
- 58 Date Functions
 - 59 Date Formats
 - 61 Localities
- 64 Documaker Server Functions
- 65 Documaker Workstation Functions
- 66 Docupresentation Functions
- 67 Field Functions
 - 68 Field Formats
 - 69 Numeric Formats
 - 70 Locating Fields
- 74 File/Path Functions
- 75 Have Functions
- 76 Image Functions
- 77 INI Functions
- 78 Logo Functions
- 79 Mathematical Functions
- 80 Miscellaneous Functions

81	Name Functions
82	Page Functions
83	Printer/Recipient Functions
84	String Functions
86	Time Functions
86	Time Formats
88	WIP Functions
89	Locating Objects
92	Where DAL Functions are Used
101	@
103	?
105	ABS
106	AddAttachVAR
107	AddBlankPages
109	AddComment
110	AddDocuSaveComment
111	AddForm
112	AddForm_Propagate
114	AddImage
117	AddImage_Propagate
119	AddOvFlwSym
120	AFELog
121	AppendText
123	AppendTxm
125	AppendTxmUnique
128	AppldxRec
129	Ask
130	AssignWIP
131	Avg
133	BankRound
134	Beep
135	BitAnd
136	BitClear
137	BitNot
138	BitOr
139	BitRotate

141	BitSet
142	BitShift
144	BitTest
145	BitXor
146	BreakBatch
148	CFind
149	ChangeLogo
151	Char
152	CharV
153	CodeInList
154	Complete
155	CompressFlds
157	ConnectFlds
160	CopyForm
161	Count
164	CountRec
165	Cut
166	DashCode
169	Date
170	Date2Date
171	DateAdd
173	DateCnv
175	Day
176	DayName
177	DaysInMonth
178	DaysInYear
179	DBAdd
180	DBCclose
181	DBDelete
182	DBFind
184	DBFirstRec
185	DBNextRec
186	DBOpen
187	Creating Variable Length Records from Flat Files
188	DBPrepVars
189	DBUnloadDFD
190	DBUpdate

192 DDTSourceName
193 Dec2Hex
194 DeFormat
195 DelBlankPages
196 DelField
198 DelForm
199 DellImage
201 DelLogo
202 DelWIP
203 DeviceName
204 DiffDate
205 DiffDays
206 DiffHours
207 DiffMinutes
208 DiffMonths
209 DiffSeconds
210 DiffTime
211 DiffYears
213 DupForm
214 EmbedLogo
215 Exists
216 FieldFormat
217 FieldName
219 FieldPrompt
220 FieldRule
222 FieldType
223 FieldX
224 FieldY
225 FileDrive
226 FileExt
227 FileName
228 FilePath
229 Find
230 Format
231 FormDesc
232 FormName
233 FrenchNumText

234 FullFileName
235 GetAttachVAR
236 GetData
238 GetFormAttrib
240 GetINIBool
242 GetINIString
244 GetOvFlwSym
245 GetValue
246 GroupName
247 GVM
248 HaveField
250 HaveForm
251 HaveGroup
252 HaveGVM
253 HaveImage
254 HaveLogo
255 HaveRecip
256 Hex2Dec
257 Hour
258 ImageName
259 ImageRect
261 IncOvFlwSym
262 InlineLogo
263 Input
264 Insert
265 INT
266 JCenter
267 JLeft
268 JRight
269 JustField
271 KickToWIP
272 LeapYear
273 Left
274 LEN
275 ListInList
277 LoadINIFile
278 LoadLib

279 Logo
281 Lower
282 MailWIP
283 MajorVersion
284 MAX
286 MIN
288 MinorVersion
289 Minute
290 MLEInput
293 MLETranslate
296 MOD
297 Month
298 MonthName
299 MSG
300 NL
301 NUM
302 Numeric
303 NumText
305 PAD
306 PageImage
307 PageInfo
309 PaginateForm
310 ParseListCount
312 ParseListItem
314 PathCreate
315 PathExist
316 POW
317 Print
318 Print_It
319 PrinterClass
320 PrinterGroup
321 PrinterID
322 PrinterOutputSize
323 PutFormAttrib
325 PutINIBool
327 PutINIString
329 RecipBatch

330 RecipCopyCount
331 RecipientName
332 RecipName
333 Refresh
334 RemoveAttachVAR
335 RenameLogo
336 ResetFld
337 ResetOvFlwSym
338 Retain
339 Right
340 RootName
341 Round
342 RouteWIP
343 RPErrormsg
344 RPLogMsg
345 RPWarningMsg
346 SaveINIFile
347 SaveWIP
348 Second
349 SetDeviceName
350 SetEdit
352 SetFld
354 SetFont
355 SetGVM
356 SetImagePos
358 SetProtect
359 SetRecip
360 SetRequiredFld
361 SetWIPFld
362 Size
363 SlipAppend
364 SlipInsert
365 SpanField
367 SrchData
369 STR
370 STRCompare
372 SUB

373 SUM
376 SuppressBanner
377 Table
379 Time
380 Time2Time
381 TimeAdd
382 TotalPages
383 TotalSheets
384 TriggerFormName
385 TriggerImageName
386 TriggerRecsPerOvFlw
387 Trim
388 Upper
389 UniqueString
390 UserID
391 UserLvl
392 WeekDay
393 WhatForm
394 WhatGroup
395 WhatImage
396 WIPExit
397 WIPFld
398 WIPKey1
399 WIPKey2
400 WIPKeyID
401 Year
402 YearDay

403 Index

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 on page 2](#)
- [Using the Field's Properties Window on page 3](#)
- [Entering Calculations in External Files on page 4](#)
- [Creating a DAL Script Library on page 5](#)
- [Executing a DAL Script from a Menu on page 7](#)
- [Using INI Options on page 8](#)
- [Using Built-In Functions on page 10](#)
- [Using DAL with XML on page 11](#)
- [Checking KeyID Entries on page 19](#)
- [Grammar and Syntax on page 22](#)
- [Testing DAL Scripts on page 39](#)
- [Runtime Error Messages on page 41](#)
- [DAL Script Examples on page 43](#)

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 Image Check or during entry. This is a convenient way to disable the script without deleting it from the Properties window.

Understanding the System

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 images. 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 Image Editor 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 36](#).

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 ( @("multiply_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 278](#).

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

See also [Using INI Options on page 8](#)
[LoadLib on page 278](#)

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 <i>Fanelli</i> . 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 <i>Account</i> , the script assigns the documents to the user ID <i>Brown</i> . 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 39](#).

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 <i>DAL</i> . 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 it is <i>DAL</i> .
Title	Enter a title. The default is <i>DALRUN - Document Automation Language Runtime</i>
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 data 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 image 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 <i>SetRcpTb</i> .

The system also provides a number of specialized INI functions. For more information, see [INI Functions on page 77](#).

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), 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.

USING DAL WITH XML

You can use DAL XML API functions to let Documaker applications access specified XML documents and retrieve XML data via a DAL script. There are two scenarios in which you would use DAL XML API 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 API functions in a DAL script to access the XML tree and extract XML data.

Here are examples of the form set and image rules you would add and a DAL script that would call the XML API 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 image 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 BUILT-IN FUNCTIONS

The DAL XML API functions are registered in keywords, called built-in functions. A DAL 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.

NOTE: A list type DAL variable always begins with a percent sign (%) and an integer type DAL variable always begins with an octothorpe (#). Floating decimal numbers begin with a dollar sign (\$). A string type DAL variable does not begin with a leading symbol.

Here is a brief description of the DAL XML built-in functions:

LoadXMLList

```
%xListH=LoadXMLList(filename);
```

This function loads a XML document and extracts an XML tree. The only required input parameter is the XML document file name. This function returns the XML tree in the list type DAL variable.

For an example, see the DAL script in scenario 2.

DestroyList

```
#rc=DestroyList(%xListH);
```

This function destroys the XML tree created by LoadXMLList. The input parameter is a list type DAL variable that passes the XML tree handle. This function returns one (1) for success or zero (0) for failure. The return DAL variable is of integer type.

For an example, see the DAL script in scenario 2.

GetListElem

```
aStr=GetListElem(%xListH, SrchCriteria);
```

This function has two input parameters. The first is a list type DAL variable that passes the XML tree handle. The second is 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 success, it returns a text string which contains the first element that matches the search criteria.

This example returns the text of the first matched element node *Form* with the attribute name *ID* and value *Agent*.

```
%xListH=LoadXMLList("test.xml");  
aStr= GetListElem(%xListH, "Form", "ID", "Agent");  
return(aStr);
```

IsXMLError

```
IsXMLError;
```

This function checks the list for error status. The input parameter is a list type DAL variable that passes the XML tree handle. This function returns one (1) if there no errors occur or zero (0) if errors do occur.

XMLFind

```
Result=XMLFind(%xListH, srchnode, xpath);
```

This function 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, depending on the search request.

This function has three input parameters. The first is a list type DAL variable passed from either the XMLFileExtract rule or the LoadXMLList built-in function. The second is a string type DAL variable that passes a node name from which the search starts. The third is also 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.

Result can be a list type or a string type DAL variable.

For an example, see the next section.

XMLFirst

```
#rc=XMLFirst(%listH);
```

This function takes one input parameter, a list type DAL variable. The variable can be either a XML tree or a list of extracted elements. In any cases, it sets the current pointer to the first element in the specified list. This function returns one (1) for success or zero (0) for failure.

This example returns text from the last element in the list.

```
aStr="Text not found!";
%xListH=LoadXMLList("test.xml");
%listH=XMLFind(%xListH, "Forms", "Form[text()]");
#rc=XMLFirst(%listH);
loop:
if #rc=0
goto endloop:
end
aStr=XMLGetCurName(%listH);
#rc=XMLNext(%listH);
goto loop:
endloop:
#rc=DestroyList(%xListH);
return(aStr);
```

XMLNext

```
#rc=XMLNext(%listH);
```

This function is similar to XMLFirst. It sets the current pointer to the next node or element in the specified list and returns one (1) for success or zero (0) for failure.

For an example, see XMLFirst.

XMLGetCurName

```
aStr=XMLGetCurName(%listH);
```

This function takes one input parameter of the list type. It can be either an XML tree or a list of elements. It returns the element name from the current element. The return value is the string type.

For an example see XMLFirst.

XMLGetCurText

```
aStr=XMLGetCurText(%listH);
```

This function is similar to XMLGetCurName. It returns the text from the current element. The return value is the string type. The message is similar to that from the XMLGetCurName function.

For an example see XMLFirst.

XMLFirstText

```
XMLFirstText(List)
```

Use this function to set the current text to be the first text element in the XML search list and then retrieve that text.

Here is an example:

```
Mystring = XMLFirstText(%mylist)
```

XMLNextText

```
XMLNextText(List)
```

Use this function to retrieve the next text element in the XML search list.

Here is an example:

```
Mystring = XMLNextText(%textlistH);
```

XMLFirstAttrib

```
rc=XMLFirstAttrib(%listH);
```

This function has one input parameter of a list type variable. It can be an element or attribute list. 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 the input is an element list, use these functions to retrieve the attribute name and value:

- XMLAttrName
- XMLAttrValue

If the input is an attribute list, use these functions to retrieve attribute name and value:

- XMLNthAttrName
- XMLNthAttrValue

For examples, see XMLAttrName and XMLNthAttrName.

XMLNextAttrib

```
rc=XMLNextAttrib(%listH);
```

This function is similar to XMLFirstAttrib. It 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.

For an example, see XMLAttrName and XMLNthAttrName.

XMLAttrName

```
aStr=XMLAttrName(%listH);
```

This function takes a list type DAL variable of input parameter. It returns the name of the current attribute pointed to by the XMLFirstAttrib and XMLNextAttrib functions.

The example returns the second attribute name of the first Form is the list.

```
aStr="Attribute not found!";
%xList=LoadXMLList("test.xml");
%listH=XMLFind(%xList,"Forms","Form");
#rc=XMLFirst(%listH);
#rc=XMLFirstAttrib(%listH);
#rc=XMLNextAttrib(%listH);
if #rc > 0
aStr=XMLAttrName(%listH);
end
#rt=DestroyList(%xList);
return(aStr);
```

XMLAttrValue

```
aStr=XMLAttrValue(%listH);
```

This function is similar to XMLAttrName. It returns the value of the current attribute pointed to by the XMLFirstAttrib and XMLNextAttrib functions.

For an example, see XMLAttrName. Use XMLAttrValue to replace XMLAttrName.

XMLNthText

```
aStr=XMLNthText(%listH,#index);
```

This function has two input parameters. One is a list type DAL variable that passes a text list. The other is an integer type DAL variable that passes an index number. It returns the nth text value indicated by the index number.

In this example, LoadXMLList returns a text list and XMLNthText gets the first text.

```
Astr="Text not found";
%xList=LoadXMLList("test.xml");
%listH=XMLFind(%xList,"Forms","Form/text()");
aStr=XMLNthText(%listH, 1);
#rt=DestroyList(%xList);
return(aStr);
```

XMLNthAttrName

```
aStr=XMLNthAttrValue(%listH,#index);
```

This function has two input parameters. One is a list type DAL variable that passes an attribute list. The other is an integer type DAL variable that passes an index number. It returns the nth attribute name indicated by the index number.

In this example, XMLFind returns a list of attributes and XMLNthAttrName returns the name of the first attribute in the list.

```
aStr="Attribute not found!";
%xList=LoadXMLList("test.xml");
%listH=XMLFind(%xList,"Forms","Form/@*");
aStr=XMLNthAttrName(%listH, 1);
end
#rt=DestroyList(%xList);
return(aStr);
```

XMLNthAttrValue

```
aStr=XMLNthAttrValue(%list,#index);
```

This function is similar to XMLNthAttrName. It returns the nth attribute value indicated by the index number.

For an example, see XMLNthAttrName. Use XMLNthAttrValue to replace XMLNthAttrName.

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 Skywire Software'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(<i>node-set</i>)	string(<i>object</i>)
concat(<i>string, string, string...</i>)		

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"]
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

XMLFind 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.

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 400](#), [WIPKey1 on page 398](#), or [WIPFld on page 397](#).

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 299](#) 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 image variable fields.

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 an image 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 = "Skywire"
```

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.

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 image 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.
*	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: <code>!(10=9)</code> = true)

Operator Function

!=	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 47 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

Operator	Order of Execution
* / (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 image 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 image variable field named PremBasis1 adds the value in the image variable field named PremBasis2; multiplies the total of these two fields by the value in the image 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 assignment:

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: As demonstrated by this example, 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.

Another example follows:

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

In this example, 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.

Here's another example:

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

In this example, 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 30](#).

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 33](#) for more information. Here are some sample RETURN statements:

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

RESULT: Takes the data in the image variable field LAST_NAME adds a comma; adds the data in the image variable field FIRST_NAME; adds the data in the image variable field MIDDLE_INIT and places this data in another image 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 image 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 image 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 image 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 image 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 image 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 image 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 image 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 image 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 image 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 image 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 29](#) 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 END SUB

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 33); sends control back to the original script.
SetFld	Assigns a value or the results of a calculation (target variable) to a variable field on an image. The variable field maybe on any image 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 an image you specify.
AppendTxmUnique	Attaches text to the end of a multi-line text variable field from the first text area field found on an image 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	<p>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:</p> <pre>< DALRun > Script = file name</pre> <p>You can use any extension. The default is <i>DAL</i>.</p>
/INI	<p>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.</p> <p>Here are the INI options you can include in the INI file:</p> <pre>< 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)</pre>
/D	<p>The debug switch starts the DAL Debugger. When on, the script executes in single step mode and registers this DAL function: <i>DEBUG("message")</i>. The <i>DEBUG</i> function breaks execution, displays a message, and invokes the debugger in single step mode.</p>
/T	<p>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.</p>

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.
Cannot modify 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

Preparing AFP or Metacode print streams for Docusave

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

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.
- Skywire Software'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'
    mvfile= 'MVS_FILENAME'
    expbprep = 'EXBPBP'
    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',999999))

*      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 & procddate & 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:

- [Bit/Binary Functions on page 49](#)
- [Database Functions on page 50](#)
- [Date Functions on page 58](#)
- [Documaker Server Functions on page 64](#)
- [Documaker Workstation Functions on page 65](#)
- [Field Functions on page 67](#)
- [File/Path Functions on page 74](#)
- [Have Functions on page 75](#)
- [Image Functions on page 76](#)
- [INI Functions on page 77](#)
- [Logo Functions on page 78](#)
- [Mathematical Functions on page 79](#)
- [Miscellaneous Functions on page 80](#)
- [Name Functions on page 81](#)
- [Page Functions on page 82](#)
- [Printer/Recipient Functions on page 83](#)
- [String Functions on page 84](#)
- [Time Functions on page 86](#)
- [WIP Functions on page 88](#)
- [Locating Objects on page 89](#)
- [Where DAL Functions are Used on page 92](#)

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 associated parameters must be written in this syntax:

```
FUNCTION NAME( 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	Description
IF (FUNCTION()) then ... END	This example 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();	This example demonstrates assigning another variable the result returned from a function.
Y = FUNCTION(FUNCTION2());	This example 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();	Finally, this example 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.

Understanding the System

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.
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.
DBCclose	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

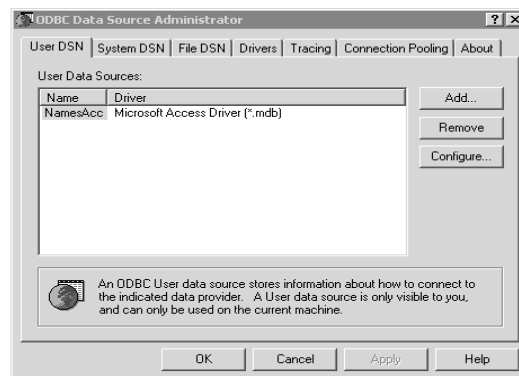
You define a database handler for a Microsoft Excel database in a similar manner to that described for an ODBC database. The following INI options are used to install the Excel handler to access a database defined as part of an Excel spreadsheet. The handler name in this example is *NamesExcel*.

```
< 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:

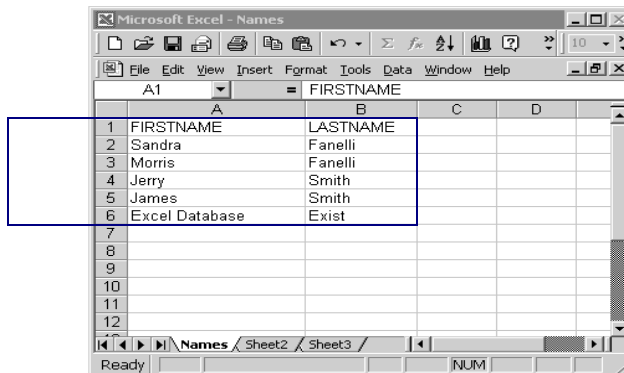
- 1 Click the Add button and select the ODBC driver to use. Then click Finish.
- 2 Enter the desired Data Source Name and description. You can enter up to 22 characters for the data source name.
- 3 Click the Workbook Selection button and select the path for the database. Then click Ok.



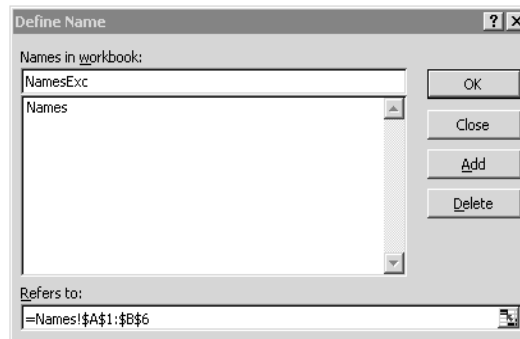
Here is an example of the steps you would follow to define a database in an Excel spreadsheet.

- 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 now supports multiple simultaneous ODBC connections via different ODBC drivers. This will, for instance, let you 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
- DUE DATE

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.DUE DATE
```

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 image 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)(Locality)

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, or N. The default is 1 (one). You must include a format type if you want to specify a separator, a year size, or a locality.
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 (07) or 4 (2007) to indicate a two- or four-digit year. We recommend that you use four-digit years. DAL functions will 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 02/17/07.
Case	(optional) To return an uppercase date, such as FEBRUARY 17, 2007, include this character before the Locality: > To return a lowercase date, such as february 17, 2007, include this character before the Locality: < For a mixed case date, such as February 17, 2007, omit this parameter.
Locality	For DAL functions, you can enter an additional component to specify the locality. This is done with @xxx, where xxx indicates the locality. You must include the @, or the system will ignore the locality code (xxx). US English is the default.

NOTE: Date formats are also used in the variable field properties in the Image Editor. 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.

Format	Date order	Description
1	MM/DD/YY	Month-Day-Year with leading zeros (02/17/2007)
2	DD/MM/YY	Day-Month-Year with leading zeros (17/02/2007)
3	YY/MM/DD	Year-Month-Day with leading zeros (2007/02/17)

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

Format	Date order	Description
4	Month D, Yr	Month name-Day-Year with no leading zeros (February 17, 2007)
5	M/D/YY	Month-Day-Year with no leading zeros (2/17/2007)
6	D/M/YY	Day-Month-Year with no leading zeros (17/2/2007)
7	YY/M/D	Year-Month-Day with no leading zeros (2007/2/17)
8	bM/bD/YY	Month-Day-Year with spaces instead of leading zeros (2/17/2007)
9	bD/bM/YY	Day-Month-Year with spaces instead of leading zeros (17/ 2/2007)
A	YY/bM/bD	Year-Month-Day with spaces instead of leading zeros (2007/ 2/17)
B	MMDDYY	Month-Day-Year with no separators (02172007)
C	DDMMYY	Day-Month-Year with no separators (17022007)
D	YYMMDD	Year-Month-Day with no separators (20070217)
E	MonDDYY	Month abbreviation-Day-Year with leading zeros (Feb172007)
F	DDMonYY	Day-Month abbreviation-Year with leading zeros (17Feb2007)
G	YYMonDD	Year-Month abbreviation-Day with leading zeros (2007Feb17)
H	day/YY	Day of year (counting consecutively from January 1)-Year (48/2007)
I	YY/day	Year-Day of Year (counting consecutively from January 1—often called the Julian date format) (2007/48)
J	D Month, Yr	Day-Month name-Year (17 February, 2007)
K	Yr, Month D	Year-Month name-Day (2007, February 17)
L *	Mon-DD-YYYY	Month abbreviation, Day with leading zeros, Year (Feb 17, 2007)
M *	DD-Mon-YYYY	Day with leading zeros, Month abbreviation, Year 17 Feb, 2007.
N	YYYYY-Mon-DD	Year, Month abbreviation, Day with leading zeros (2007, Feb 17) This format defaults to a two-digit year, but can be overridden to have four digits.

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

Format	Date order	Description
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.

Understanding the System

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.

Localities

Here is a list of the currently supported localities:

For this country	And this language	Use this code
Argentina	Spanish	ARS
Australia	English	AUD

For this country	And this language	Use this code
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
The Netherlands	Dutch	NLG
New Zealand	English	NZD
Norway	Norwegian	NOK

For this country	And this language	Use this code
Panama	Spanish	PAB
Paraguay	Spanish	PYG
Peru	Spanish	PES
Portugal	Portuguese	PTE
South Africa	English	ZAR
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, 2007*:

Example	Description	Result
1	Format type 1	12/18/07
1-	Format type 1 with dashes (-) as the separator characters	12-18-07
1/2	Format type 1 with backslashes (/) as the separator characters and a two-digit year	12/18/07
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	12/18/07
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)	12182007
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 locality, it must be the last component of the date format	décembre 18, 2007

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.
AppldxRec	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.
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 image (FAP file) name of the current SetRecipTb entry being processed.
TriggerRecsPerOvFlw	Retrieves the number of records per overflow image 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 image 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 images. 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.
DashCode	Creates a value to assign to a series of fields from the binary value of an integer.
DelField	Deletes a field from an image.
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 an image field.
SetFont	Change the font on a field.
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 an image 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.
STRCompare	Compares two strings, considering case.

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 image 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

Format	Definition	Description
T	Time	Accepts only time
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.
“Z”	<p>Tells the system to automatically suppress leading zeros in the specified positions of the field at data entry time.</p> <p>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.</p> <p>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).</p>

Component	Description
-----------	-------------

“\$”	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
ZZZZZZZZZZZZ
```

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 image. To locate specific fields, elsewhere in the document, requires additional information. Any field's location can be precisely determined by the following hierarchy:

```
Field -> Image -> Form -> Group
```

Fields occur on images. Images 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 image which is the image that contains the script being executed. If the field you wish to reference occurs on the current image, 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, image, or form names.

To locate a field on an image other than the current one requires additional information. Each field function accepts optional parameters to identify a specific field, image, 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 an image. However, that same field name might also be used on any number of other images. Further, there may be any number of occurrences of an image 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 images 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	Image 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 image.
"FLD"	"IMG"	*omitted*	*omitted*	Find the first occurrence of IMG (an image) on the current form. If located, find FLD on that image.
"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 image 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 image on any form within that group.
"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 image.
"FLD"	"IMG"	*omitted*	"GRP"	Find the first occurrence of IMG within the group, GRP. This image may occur on any form since that parameter was not specified. Then find FLD on that image.
"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 image.

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, image, 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	Image Name	Form Name	Group Name	Description
"FLD"	"IMG\2"	*omitted*	*omitted*	Find the second occurrence of IMG (an image) on the current form. If located, find FLD on that image.
"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 image 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 image or form within that group.
"FLD"	"IMG\5"	*omitted*	"GRP"	Find the fifth occurrence of IMG (an image) within the group, GRP. If located, find FLD on that image.

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/PATH FUNCTIONS

The File/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
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 image exists.
HaveLogo	Determines whether a named logo (bitmap) exists.
HaveRecip	Determines if a recipient name is defined in the FORM.DAT file.

IMAGE FUNCTIONS

The 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 image to a form as a new page.
DelForm	Deletes a specified form from the current document.
DelImage	Deletes an image from a form.
EmbedLogo	Embeds a logo into the NAFILE.DAT file.
SetImagePos	Repositions an image on a page.
SetRecip	Sets the recipient copy count for a form or group.

INI FUNCTIONS

INI functions let you 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.
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.

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

LOGO FUNCTIONS

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

Function	Result
ChangeLogo	Replaces an existing logo on the image with a new logo (bitmap).
DelLogo	Deletes a logo from a form.
ImageRect	Retrieves the coordinates of an image.
InlineLogo	In-lines a logo (bitmap) into the print stream
Logo	Places a new logo (bitmap) at a specified position on the image.

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
Exists	Determines if a DAL symbolic variable exists.
GetFormAttrib	Returns the content of the named user attribute (metadata) for the form you specify
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.
PutFormAttrib	Saves the named attribute and information to a form within your document set
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.
FieldRule	Executes a field-level rule from within a DAL script.
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 image's name.
PageImage	Returns the name of an image 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 image, form, or group.
RenameLogo	Renames a logo (bitmap).
RootName	Extracts and returns the root name, or the original part of the name, of a specified string.
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 image 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 image origins and re-paginates the form if necessary.

PRINTER/ 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.
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.
RecipCopyCount	Counts the number of recipient copies for specified images and returns that number.
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.

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.

Function	Result
NumText	Converts a number into a string of words and returns the result (in English).
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.
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.

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 image.

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

Format	Time Segments	Description
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)

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 image to a document and propagates global data onto that image.
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.
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.

LOCATING OBJECTS

Many of the logo, 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 70](#).

```
item -> Image -> Form -> Group
```

A number of different object types are supported by images. Three objects that can be located on an image are fields, logos, and recipients. For information about fields, see [Locating Fields on page 70](#). Fields, logos, and recipients are all objects that belong or are defined on an image.

Images 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 image 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 an image are unique to that image. However, that same object name might also be used on any number of other images. Further, there may be any number of occurrences of an image 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 images 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	Image Name	Form Name	Group Name	Description
"ITEM"	*omitted*	*omitted*	*omitted*	Find the object on the current image.
"ITEM"	"IMG"	*omitted*	*omitted*	Find the first occurrence of IMG (an image) on the current form. If located, find ITEM on that image.
"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 image 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 image on any form within that group.

Item Name	Image 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 image.
"ITEM"	"IMG"	*omitted*	"GRP"	Find the first occurrence of IMG within the group, GRP. This image may occur on any form since that parameter was not specified. Then find ITEM on that image.
"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 image.

Locating images

	"IMG"	*omitted*	*omitted*	Find the occurrence of the image on the current form.
	"IMG"	"FRM"	*omitted*	Find the occurrence of the image on the form named. The form is assumed to be in the current group.
	"IMG"	*omitted*	"GRP"	Locate the image within the named group.
	"IMG"	"FRM"	"GRP"	Locate the form in the specified group. Then locate the image 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	Image Name	Form Name	Group Name	Description
"ITEM"	"IMG\2"	*omitted *	*omitted *	Find the second occurrence of IMG (an image) on the current form. If located, find ITEM on that image.
"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 (an image) within the group, GRP. If located, find ITEM on that image.

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
AppldxRec	Yes	Yes
Ask	Yes	No
AssignWIP	Yes	No
Avg	Yes	Yes
BankRound	Yes	Yes
Beep	Yes	No
BitAnd	Yes	Yes
BitClear	Yes	Yes

Function/Procedure	Affects form entry (Documaker Workstation)	Affects form processing (Documaker Server)
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	No
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
DateCnv	Yes	Yes
Day	Yes	Yes
DayName	Yes	Yes
DaysInMonth	Yes	Yes
DaysInYear	Yes	Yes

Function/Procedure	Affects form entry (Documaker Workstation)	Affects form processing (Documaker Server)
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
DeviceName	Yes	Yes
DiffDate	Yes	Yes
DiffDays	Yes	Yes
DiffHours	Yes	Yes
DiffMinutes	Yes	Yes
DiffMonths	Yes	Yes
DiffSeconds	Yes	Yes
DiffTime	Yes	Yes

Function/Procedure	Affects form entry (Documaker Workstation)	Affects form processing (Documaker Server)
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
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
GetOvFlwSym	No	Yes

Function/Procedure	Affects form entry (Documaker Workstation)	Affects form processing (Documaker Server)
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
IncOvFlwSym	No	Yes
InlineLogo	Yes	Yes
Input	Yes	No
Insert	Yes	Yes
INT	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

Function/Procedure	Affects form entry (Documaker Workstation)	Affects form processing (Documaker Server)
ListInList	Yes	Yes
LoadINIFile	Yes	Yes
LoadLib	Yes	Yes
Logo	Yes	Yes
Lower	Yes	Yes
MailWIP	Yes	No
MajorVersion	Yes	Yes
MAX	Yes	Yes
MIN	Yes	Yes
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

Function/Procedure	Affects form entry (Documaker Workstation)	Affects form processing (Documaker Server)
PathCreate	Yes	Yes
PathExist	Yes	Yes
POW	Yes	Yes
Print	Yes	No
Print_It	Yes	Yes
PrinterClass	Yes	Yes
PrinterGroup	Yes	Yes
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

Function/Procedure	Affects form entry (Documaker Workstation)	Affects form processing (Documaker Server)
RPLogMsg	No	Yes
RPWarningMsg	No	Yes
SaveINIFile	Yes	Yes
SaveWIP	Yes	No
Second	Yes	Yes
SetDeviceName	Yes	Yes
SetEdit	Yes	No
SetFld	Yes	Yes
SetFont	Yes	Yes
SetGVM	Yes	Yes
SetImagePos	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

Function/Procedure	Affects form entry (Documaker Workstation)	Affects form processing (Documaker Server)
Time2Time	Yes	Yes
TimeAdd	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
Year	Yes	Yes
YearDay	Yes	Yes



Use this function to return the current value contained in an image 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, Image, Form, Group)

Parameter	Description	Defaults to...
Field	Name of an image field.	the current field name
Image	Name of an image that contains the field named.	the current image
Form	Name of a form that contains the image and/or field named.	the current form
Group	Name of the form group that contains the form, image, or field.	the current group

This function uses the parameters you provide to search for one field on an image and return that field's data. If the field is defined as a numeric data type, the result is returned as 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 image *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 image.
Return (@("Formset Page Num"))	3	Returns the value in the field named "Formset Page Num" on the current image.
Return (@(Form Page Num"), "Header3", "ABC")	2	Returns the value in the field named "Form Page Num" in image, "Header3" on form "ABC".

See also [Field Functions on page 67](#)
[Field Formats on page 68](#)
[Locating Fields on page 70](#)

?

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

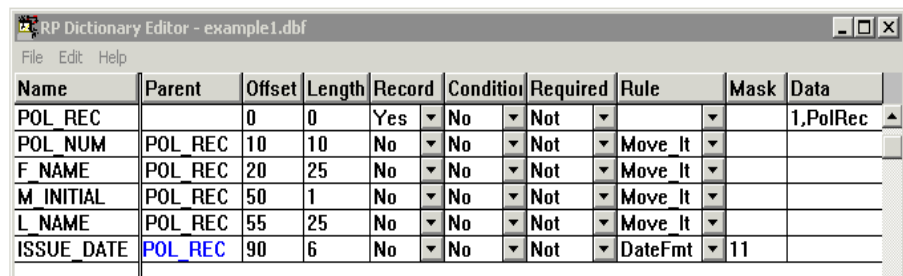
Syntax ?(LookUpName, Occurrence)

Parameter **Description**

LookUpName	Specifies the entry name in the XDB that defines the data to retrieve.
Occurrence	Optional. Defines 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 Dictionary Editor chapter in the [Docucreate User Guide](#), which describes how to define extract file records and fields in the XDB database.

Let's assume you have these entries defined in the XDB:



Name	Parent	Offset	Length	Record	Condition	Required	Rule	Mask	Data
POL_REC		0	0	Yes	No	Not			1,PolRec
POL_NUM	POL_REC	10	10	No	No	Not	Move_It		
F_NAME	POL_REC	20	25	No	No	Not	Move_It		
M_INITIAL	POL_REC	50	1	No	No	Not	Move_It		
L_NAME	POL_REC	55	25	No	No	Not	Move_It		
ISSUE_DATE	POL_REC	90	6	No	No	Not	DateFmt	11	

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	Fanelli	09221957
PolRec	GRA0001	Sandra	J	Fanelli	09211959
PolRec	GRA0001	Vincent	M	Fanelli	12311981

Example 1 Assume the Driver field has this script in its DDT file 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. Fanelli
```

Example 2 In this example, assume there are ten fields (driver01, driver02, and so on) on the image, the first field has this script in its DDT file, 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. Fanelli
Sandra J. Fanelli
Vincent M. Fanelli
```

Example 3 In this example, assume the License Issued field has this script in its DDT file 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 220](#)
[GetData on page 236](#)
[Documaker Server Functions on page 64](#)
[Field Formats on page 68](#)
[Locating Fields on page 70](#)

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	Defaults to...
Number	any number data type	the current field value

This function 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 79](#)

ADDATTACHVAR

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

Parameter	Description
Name	Name of the attachment variable.
Value	Value to be added.
DSI queue input or output	(optional) Enter 1 for input or 2 for output. Defaults to output.
Returns	1 if successful, 0 if not.

See also [Docupresentment Functions on page 66](#)
[GetAttachVAR on page 235](#)
[RemoveAttachVAR on page 334](#)

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	Defaults to...
FAP	The name of the FAP file you want the system to use as a filler page.	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 Server System Reference](#) 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.

If the current page is	And the next 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

Understanding the System

You can also add blank or filler pages using custom code or by using the `DPRAddBlankPages` function, which is available with the Internet Document Server. See the SDK Reference 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 image name is NULL, a blank page is created when a filler page is needed. If the image name is not NULL, the image name is loaded when a filler page is needed. If you include an image name, include only the name of the FAP file—omit the path and file extension.

See also [DelBlankPages on page 195](#)
[SuppressBanner on page 376](#)
[Page Functions on page 82](#)
[Miscellaneous Functions on page 80](#)
[Creating a DAL Script Library on page 5](#)

AddComment

Use this procedure to add a comment to the print stream. Products like Skywire Software'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 PJI (Printer Job Language). PJI 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 43](#).

Syntax AddComment (Comment, Convert)

Parameter	Description	Defaults to...
Comment	Enter the string you want used as a comment in the print stream or the name of an image variable field that contains the comment.	no default
Convert	Enter one of these codes: 0 - converts the string to EBCDIC 1 - converts the string to ASCII 2 - do not convert the string For OnDemand, you will always want EBCDIC comments.	0 (zero)

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/Recipient Functions on page 83](#)

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	Defaults to...
Comment	Enter the string to be written as a comment in the print stream.	
Convert	(optional) Zero (0) tells the system to convert the string to EBCDIC (default). Enter one (1) to tell the system to convert the string to ASCII. Enter two (2) to tell the system not to convert the string. For Docusave, you will always want EBCDIC comments.	Zero (0)

Example Here are some examples:

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

See also [Printer/Recipient Functions on page 83](#)
 [DAL Script Examples on page 43](#)

ADDFORM

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

Syntax AddForm (Form, Insert, Group)

Parameter	Description	Defaults to...
Form	Name of a form in the specified group.	no default
Insert	The name of an existing form <i>after</i> which the new form should be inserted.	append after the last form in the group
Group	Name of a group to contain the specified form.	the current group

Optionally, this procedure 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.

NOTE: If you use this procedure to add 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 Supervisor Guide](#).

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 112](#)

[CopyForm on page 160](#)

[DupForm on page 213](#)

[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	Defaults to...
Form	Name of a form in the specified group.	No default.
Insert	The name of an existing form after which the new form should be inserted.	Appended after the last form in the current group
Group	Name of a group to contain the specified form.	The current group.

Optionally, this procedure 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 Supervisor 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:	Skywire Software	form name =	C22510WGI
Employee:	J. Stewart	image name =	M
Date of Loss:	12/11/07		GENRCHDR
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 image 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:	Skywire Software	form name =	C22510WGIM\
Employee:	J. Stewart	image name =	2
Date of Loss:	12/11/07		GENRCHDR
File Number:	12345		
State Case Num:			
Samford and Son			
Sincerely,			
Workers' Compensation Unit			

See also

[AddForm on page 111](#)

[CopyForm on page 160](#)

[DupForm on page 213](#)

[WIP Functions on page 88](#)

ADDIMAGE

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

Syntax **AddImage (FAP, Image, Form, Group, Flag, Paginate)**

Parameter	Description	Defaults to...
FAP	The name of the image file to load and add to the form.	no default
Image	Name of an existing image which will precede the new image.	the current image
Form	Name of a form in the form set. If you specify the Image parameter, that image must occur on this form.	the current form
Group	Name of a group that contains the specified form.	the current group
Flag	Determines if the image is inserted on the same page or on a new page. 0 = new page 1 = same page	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 image. If the image does contain an origin rule and you omit the Paginate option or set it to zero (0), the image origin rule executes upon insertion. Whether the inserted image has an origin rule or not, the positioning of this image 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 image causes it to overlap another image or to be out of the page boundary, no additional re-pagination occurs. If you are manipulating multiple images 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: <pre>AddlImage ("myFAP", "mainImage" , , , 1,1)</pre> 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.	zero (0)

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

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

NOTE: If you use this procedure to add images 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 Supervisor Guide](#).

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

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

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

Understanding the System

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

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 an image 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 image, 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 image, NEW1, after the third occurrence of IMG, within GRP. This image is placed on a new page after the third occurrence of the specified image.
AddImage ("IMG1" ,,,, 1)	1 - if successfully added. 0 - if not added.	Insert the named image, IMG1, after the current image on the same page.

Procedure	Result	Explanation
AddImage("NEW1", "IMG\3", , , 1)	1 - if successfully added. 0 - if not added.	Insert the named image, NEW1, after the third occurrence of IMG on the same page.

See also [DellImage on page 199](#)
[ImageRect on page 259](#)
[PaginateForm on page 309](#)
[SetImagePos on page 356](#)
[Refresh on page 333](#)
[Image Functions on page 76](#)

ADDIMAGE_PROPAGATE

Use this procedure/function during GenData processing to add a new image and propagate global data onto the newly added image 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, Image, Form, Group, Flag)

Parameter	Description	Defaults to...
FAP	The name of the image file to load and add to the form.	no default
Image	Name of an existing image which will precede the new image.	the current image
Form	Name of a form in the form set. If you specify the Image parameter, that image must occur on this form.	the current form
Group	Name of a group that contains the specified form.	the current group
Flag	Determines if the image is inserted on the same page or on a new page. 0 = new page 1 = same page	zero

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

This procedure adds a copy of the image you specify to a form. The system loads the new image onto the page after the image, form, or group you specified or onto a new page which it creates *after* the image, form, or group you specified. The image 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 images already defined for a page. Therefore, you can overlay existing images on the page.
- Make sure you do not unintentionally overlay an existing image. You can move the new image using the ImageRect and SetImagePos procedures.
- If you use this procedure to add images 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 Supervisor Guide](#).

- When adding an image, there is no way for you to specify what image options or recipients you want included on the new image. So, the AddImage_Propagate procedure takes the missing information from an associated image. 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 an image 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 image, 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 image, NEW1, after the third occurrence of IMG, within GRP. This image is placed on a new page after the third occurrence of the specified image.
AddImage_Propagate ("IMG1",,,, 1)	1 - if successfully added. 0 - if not added.	Insert the named image, IMG1, after the current image on the same page.
AddImage_Propagate("NEW1", "IMG\3", , , 1)	1 - if successfully added. 0 - if not added.	Insert the named image, NEW1, after the third occurrence of IMG on the same page.

See also [AddImage on page 114](#)
[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	Required?
Form	The name of the form that contains the fields on which overflow processing will occur.	Yes
Symbol	The name you want to use as the overflow symbol.	Yes
MaxRecords	Defines the maximum number of overflow records to be processed for the image per page of output.	Yes

This procedure creates an overflow symbol associated with the specified image. Optionally, this procedure returns one (1) on success or zero (0) on failure.

Example Here are some examples:

Assume that the image, 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 image, *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 in a field's DDT file 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 in the field's DDT file or the XDB name associated with the field, as follows:

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

See also [GetOvFlwSym on page 244](#)

[IncOvFlwSym on page 261](#)

[ResetOvFlwSym on page 337](#)

[Documaker Server Functions on page 64](#)

AFELOG

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

Syntax AFELog (String)

Parameter	Description	Required
String	A valid string.	Required, no default.

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](#)

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, Image, Form, Group)**

Parameter	Description	Defaults to...
File	Required. Name of an external text file including any file extension. This text is appended to the field you specify.	no default
Field	A field name that identifies a multi-line text area. This is the field that receives the appended text.	the current field
Image	Name of an image that contains the field named.	the current image
Form	Name of a form that contains the image and/or field named.	the current form
Group	Name of the form group that contains the form, image, and/or field named.	the current form group

Optionally, this procedure 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 images 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, <i>MyFile.Txt</i> . The named file does not specify a path, therefore the system tries to locate the file where form images are normally located.
AppendText ("C:\MyFile.txt", "MyField")	1 or 0	<i>MyField</i> will be located on the current image. If found, the field receives the text from the file named, <i>MyFile.Txt</i> . 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, <i>MyField</i> , will be located on the form, <i>MyForm</i> . Since an image was not specified, it may occur on any image on that form. Once located, the text from the specified file is appended to the field.

See also [Field Functions on page 67](#)
 [Field Formats on page 68](#)
 [Locating Fields on page 70](#)
 [Refresh on page 333](#)

APPENDTXM

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

Syntax

AppendTxm (FAP, InsertFld, Field, Image, Form, Group)

Parameter	Description	Defaults to...
FAP	Required. Enter the name of the image 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 image in the forms directory you specified using the File, Library Setup option.	no default
InsertFld	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 image) 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.	appends after the last field on the image
Field	This field name identifies the multi-line text area which will receive the appended text.	the current field (it <i>must</i> be a multi-line text field)
Image	Name of the image that contains the field you specified in the Field parameter.	the current image
Form	Name of the form that contains the image you specified in the Image parameter.	the current form
Group	Name of the group that contains the form you specified in the Form parameter.	the current form group

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

This procedure opens the image you defined in the FAP parameter and copies the text from the *first* text area field found on that image. 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
<code>#rc = AppendTxm ("Message", , "Name_Line"); Refresh (</code> <code>);</code>	1 or 0	The text in the first text area on the image named <i>Message</i> is appended to the multi-line text field, called <i>Name_Line</i> . The system then refreshes the display.
<code>#rc = AppendTxm (".\mstrres\messages\m sg1", , "Name_Line", "Mailer"); Refresh ();</code>	1 or 0	The path, <i>.\mstrres\message\</i> , is appended to the multi-line text field, called <i>Name_Line</i> , which is on the image named <i>Mailer</i> . The system then refreshes the display.
<code>#rc = AppendTxm("message", "Address1", "Name_Line"); Refresh (</code> <code>);</code>	1 or 0	The fields in the text area are inserted before the variable field named <i>Address1</i> , in the tabbing sequence. The system then refreshes the display.

See also [Field Functions on page 67](#)
[Field Formats on page 68](#)
[Locating Fields on page 70](#)
[AppendTxmUnique on page 125](#)
[Refresh on page 333](#)

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 image and you expect the user to tab through the imported field and enter new data.
- This procedure lets you import the same image 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 image that contains the text area. This prevents naming conflicts with prior images.

This procedure only works on multi-line text fields.

Syntax

AppendTxmUnique (FAP, InsertFld, Field, Image, Form, Group)

Parameter	Description	Defaults to...
FAP	Required. Name of an image file containing a text area. This text area is appended to the field specified. If there are several text areas in the image 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.	no default
InsertFld	Name of a field before which you want the embedded fields in the imported text area inserted.	appends after the last field on the image
Field	A field name that identifies a multi-line text area. This is the field that receives the appended text.	the current field
Image	Name of an image that contains the field.	the current image
Form	Name of a form that contains the image and/or field.	the current form
Group	Name of the group that contains the form, image, and/or field.	the current form group

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

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

The system locates the image in the directory where images (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 image 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 image 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 image. 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 image 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 image).

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 image 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 image 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 <i>Name_line</i> multi-line text field.
#rc = AppendTxmUnique (".\mstrres\messages\msg1", , "name_line", "mailer image"); 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 <i>Name_line</i> multi-line text field, which is in the Mailer image 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 <i>Name_line</i> 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 67](#)
[Locating Objects on page 89](#)
[Field Formats on page 68](#)
[Locating Fields on page 70](#)
[AppendTxm on page 123](#)
[Refresh on page 333](#)

APPIDXREC

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

Syntax **AppldxRec ()**
There are no parameters for this function.

Example Here are some examples. Assume that...

- The rundate is 01/10/1999
- The sub-string of extract record being processed is:

SCO1234567HEADERREC00000...

- The FORM.DAT file contains the following:

 ;SAMP;CO;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 = AppldxRec() Print_It (Comment)	1 or 0	

See also [Documaker Server Functions on page 64](#)
[Print_It on page 318](#)
[DAL Script Examples on page 43](#)

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	Defaults to...
Msgline1	Any string message	Yes
Msgline2	Second line of message	no default
Msgline3	Third line of message	no default
Title	Title of message window	no default
Defans	Integer one (1) or zero (0) that specifies whether the Yes or No button should be selected as the default.	1 (Yes)

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 65](#)

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	Required?
UserID	Any valid user ID.	Yes

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 returns success if no error occurred during the process, otherwise a failure. This procedure only works with the Entry module and does not work with the data entry mode of the Image Editor.

Example Here is an example:

Procedure	Result	Explanation
AssignWIP (MVV)	User ID for the work-in-process is changed to <i>MVV</i> , 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 <i>MVV</i> 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 Supervisor Guide](#)
[Documaker 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, Image, Form, Group)**

Parameter	Description	Defaults to...
PartialName	A valid string. The string must be the common (prefix) portion of a set of field names that occur on the current image.	the current field
Image	Name of an image that contains the field named.	the current image.
Form	Name of a form that contains the image and/or field named.	the current form
Group	Name of the form group that contains the form, image, and/or field named	the current form group

This function calculates 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 an image. Using the default will probably not give the expected result, unless you created the form and understand the naming conventions.

See also [Mathematical Functions on page 79](#)

[Field Formats on page 68](#)

[Locating Fields on page 70](#)

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

Field	Image	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
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 image 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 image. 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 image 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 image 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 an image 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 image. 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 [Field Formats on page 68](#)
[Locating Fields on page 70](#)

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 84](#)

[Round on page 341](#)

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	Defaults to...
Integer	0 = Warning sound 1 = Message sound 2 = Error sound	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 65](#)

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 136](#)
[BitNot on page 137](#)
[BitOr on page 138](#)
[BitRotate on page 139](#)
[BitSet on page 141](#)
[BitShift on page 142](#)
[BitTest on page 144](#)
[BitXor on page 145](#)
[Bit/Binary Functions on page 49](#)

BITCLEAR

This function returns 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 135](#)
 [BitNot on page 137](#)
 [BitOr on page 138](#)
 [BitRotate on page 139](#)
 [BitSet on page 141](#)
 [BitShift on page 142](#)
 [BitTest on page 144](#)
 [BitXor on page 145](#)
 [Bit/Binary Functions on page 49](#)

BitNot

This function returns 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 135](#)

[BitClear on page 136](#)

[BitOr on page 138](#)

[BitRotate on page 139](#)

[BitSet on page 141](#)

[BitShift on page 142](#)

[BitTest on page 144](#)

[BitXor on page 145](#)

[Bit/Binary Functions on page 49](#)

BitOr

This function returns 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 135](#)

[BitClear on page 136](#)

[BitNot on page 137](#)

[BitRotate on page 139](#)

[BitSet on page 141](#)

[BitShift on page 142](#)

[BitTest on page 144](#)

[BitXor on page 145](#)

[Bit/Binary Functions on page 49](#)

BITROTATE

This function returns 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 142](#) 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 135](#)
[BitClear on page 136](#)
[BitNot on page 137](#)

[BitOr on page 138](#)
[BitSet on page 141](#)
[BitShift on page 142](#)
[BitTest on page 144](#)
[BitXor on page 145](#)
[Bit/Binary Functions on page 49](#)

BITSET

This function returns 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 135](#)
[BitClear on page 136](#)
[BitNot on page 137](#)
[BitOr on page 138](#)
[BitRotate on page 139](#)
[BitShift on page 142](#)
[BitTest on page 144](#)
[BitXor on page 145](#)
[Bit/Binary Functions on page 49](#)

BITSHIFT

This function returns 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 139](#) 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 135](#)
[BitClear on page 136](#)
[BitNot on page 137](#)
[BitOr on page 138](#)

[BitRotate on page 139](#)

[BitSet on page 141](#)

[BitTest on page 144](#)

[BitXor on page 145](#)

[Bit/Binary Functions on page 49](#)

BITTEST

This function returns TRUE (1) if the specified bit in a value is a 1; otherwise FALSE (0) is returned.

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 135](#)
 [BitClear on page 136](#)
 [BitNot on page 137](#)
 [BitOr on page 138](#)
 [BitRotate on page 139](#)
 [BitSet on page 141](#)
 [BitShift on page 142](#)
 [BitXor on page 145](#)
 [Bit/Binary Functions on page 49](#)

BITXOR

This function returns 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 135](#)
[BitClear on page 136](#)
[BitNot on page 137](#)
[BitOr on page 138](#)
[BitRotate on page 139](#)
[BitSet on page 141](#)
[BitShift on page 142](#)
[BitTest on page 144](#)
[Bit/Binary Functions on page 49](#)

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.
- Producing PDF files on Z/OS requires a separate license. Contact your Skywire Software sales representative for more information.
- 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/Recipient Functions on page 83](#)
[DeviceName on page 203](#)
[SetDeviceName on page 349](#)
[UniqueString on page 389](#)

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. A zero (0) is returned if none of the search characters are found in the text string.

Syntax CFind (String, Charset, Integer)

Parameter	Description	Defaults to...
String	A valid string. This is the string that is searched.	the value of current field text
Charset	A set of one or more characters, any of which may be found in the target string.	no default
Integer	Enter zero (0) for a left to right search. Enter one (1) for a right to left search.	zero (0)

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 <i>Your Name</i> 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 84](#)

CHANGELOGO

Use this procedure/function to replace an existing logo (bitmap) on an image with a different logo.

Syntax **ChangeLogo (Logfile, Logo, Image, Form, Group)**

Parameter	Description	Defaults to...
Logfile	Name of a file containing a valid logo.	no default
Logo	Name of the current logo on an image.	no default
Image	Name of an image containing the specified logo	the current image
Form	Name of a form containing the image or logo	the current form
Group	Name of a group to use to locate a specific object.	the current group

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

This procedure expects to locate the named logo in the same way and location used to load any other logo. You must include the Logo parameter.

If you omit the Logfile parameter or the logo cannot be loaded, the system will insert an empty logo. A placeholder appears during entry to indicate the logo position, however, nothing will print if a logo is not loaded. This procedure lets you remove a signature from a form if necessary.

The Logo parameter tells the system to look for the name that appears in the Name field on the Logo Properties window in the 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 image after it changes the logo.

When you use the ChangeLogo procedure with Documaker, you must include the CheckImageLoaded rule as one of the image level rules for the image 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 image has a logo named *sign*.)

Procedure	Result	Explanation
ChangeLogo ("johndoe", "sign")	1 or 0	Replaces the existing logo contained by sign with a new logo (johndoe). The existing logo is assumed to exist on the current image.
ChangeLogo (, "sign", "IMG")	1 or 0	Locate the specified image on the current form. If found replace the existing logo contained by sign with an empty logo.

See also [DelLogo on page 201](#)
[HaveLogo on page 254](#)
[InlineLogo on page 262](#)
[RenameLogo on page 335](#)
[Logo on page 279](#)
[Refresh on page 333](#)
[Logo Functions on page 78](#)

CHAR

Use this function to convert an integer into a single character.

Syntax Char (Integer)

Parameter	Description	Defaults to...
Integer	An integer value that ranges zero (0) to 255.	No default.

Example Here is an example:

```
what_char = Char (64)
```

The variable, *what_char*, is set to the character: '@'.

See also [CharV on page 152](#)
[String Functions on page 84](#)

CHARV

Use this function to convert a single character into an integer value.

Syntax CharV (String)

Parameter	Description	Defaults to...
String	A character string. If the string contains more than one character, only the first character is converted. The remaining characters are ignored.	No default.

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 151](#)
 [String Functions on page 84](#)

CODEINLIST

Use this function to search for a string in a list of a strings.

Syntax

CodeInList (String,List)

Parameter	Description	Defaults to
String	Enter the string you want to search for. Keep in mind the system considers spaces when matching strings and that the strings <i>must</i> match exactly.	no default
List	Enter the name of the list of strings. Use commas to separate each string entry you want to search for.	no default

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: EEach,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 84](#)

COMPLETE

Use this procedure/function to complete the work-in-process.

Syntax **Complete (PrintFlag, ExportFlag, ExportType,ExportFile)**

Parameter	Description	Defaults to...
PrintFlag	Indicates whether the system should print the form set.	No
ExportFlag	Indicates whether the system should export the work-in-process data to a file.	No
ExportType	TD, SI, and so on. Indicates the type of export file.	TD
Exportfile	The file name for the Standard Export file, if specified in the INI options.	no default

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 Supervisor Guide](#)
[Documaker 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, Image, Form, Group)

Parameter	Description	Defaults to...
FieldList	A list of the fields you want to compress, separated by commas. Here is an example: "FIELD1, FIELD2, FIELD3"	no default
Image	(optional) Name of an image that contains the fields you named.	the current image
Form	(optional) Name of a form that contains the image and/or field you named.	the current form
Group	(optional) Name of the form group that contains the form, image, or fields.	the current group

NOTE: When using this function in Documaker Server processing, make sure the fields exist on the image. 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 images 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 barcode or multi-line text fields.

Example For this example, assume the following fields and data:

This field	Contains
FIELD_A	ABCDEFGH
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	ABCDEFGH
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	ABCDEFGH
FIELD_D	TUVWXYZ

See also [Field Functions on page 67](#)

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 image — either the FAP file or the compiled version of the FAP file — if the image has not already been loaded. FAP files must be loaded to provide some of the information required to perform the operation.

Syntax

ConnectFlds(FieldList, Image, Form, Group)

Parameter	Description	Defaults to...
FieldList	<p>A list of the fields you want to connect, preceded by a movement flag and separated with commas. Here is an example:</p> <pre>"FIELD1, FIELD2, FIELD3"</pre> <p>If a field name is not preceded by a movement flag or if it is preceded by the <i>F</i> movement flag, which indicates it is a fixed field, the field is not moved.</p> <p>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.</p>	no default
Image	(optional) Name of an image that contains the fields you named.	the current image
Form	(optional) Name of a form that contains the image and/or field you named.	the current form
Group	(optional) Name of the form group that contains the form, image, or fields.	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 image.

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.
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 barcode 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:

```
DEFABCDXYZ
```

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 67](#)

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 111](#)
[AddForm_Propagate on page 112](#)
[DupForm on page 213](#)
[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, Image, Form, Group)**

Parameter	Description	Defaults to...
PartialField	A valid string. The string must be the common (prefix) portion of a set of field names that occur on the current image.	the current field
Image	Name of an image that contains the field named.	the current image
Form	Name of a form that contains the image and/or field named.	the current form
Group	Name of the form group that contains the form, image, and/or field named.	the current form 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 an image 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 images (IMG A and IMG B). Each image has fields of the same name as a field in the other image.

Field	Image	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
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 image 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 image. 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 image 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 image 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 an image 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 image. Eight fields meet the naming criteria, but only five fields actually have values.

See also [Mathematical Functions on page 79](#)
 [Field Formats on page 68](#)
 [Locating Fields on page 70](#)

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	Defaults to...
SearchMask	The search mask you want to use for the search.	No default
MinNumber	Number of records that must exist in the transaction.	(optional) 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 64](#)
[Field Formats on page 68](#)
[Locating Fields on page 70](#)

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	Defaults to...
String	A valid string.	the value of current field text
Position	Position within the first parameter to begin cutting - must be an integer.	1
Length	Length to cut from text - must be an integer.	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*.)

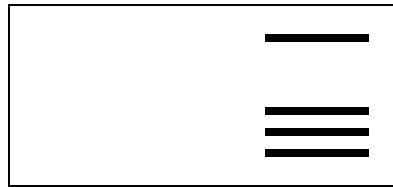
Function	Result	Explanation
Return(Cut ())	Your Name	No length is specified for the cut function; therefore the field remains the same.
Return(Cut (, , 5))	Name	Five characters are cut from the current field beginning at position 1.
Return(Cut ("Complete all the blanks.", 10, 4))	Complete the blanks	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 84](#)

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, Image, 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	<p>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 image.</p> <p>For instance, if <i>MVALUE_</i> is passed as the RootName, the first fill value is assigned to <i>MVALUE_1</i>, the second to <i>MVALUE_2</i>, <i>MVALUE_3</i>, and so on, until the maximum number of bits specified are all mapped. If all 32 bits are mapped, the last field would be <i>MVALUE_32</i>.</p> <p>The associated fields will be filled to their defined length. In most dash code (barcode) type situations, you will want all the fields to be the same length.</p>

Parameter	Description
Image	The name of an image that contains the field named. You can enter an asterisk (*) to tell the function to search all images. Keep in mind, however, that including an asterisk (*) degrades performance.
Form	The name of a form that contains the image and/or field named. You can enter an asterisk (*) to tell the function to search all forms. Keep in mind, however, that including an asterisk (*) degrades performance.
Group	The name of the form group that contains the form, image, 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	<p>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.</p> <p>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.</p> <p>For instance, suppose the field length is five; OnString is <i>ABC</i>; and OffString is <i>XY</i>. If the bit value for this field is one (1), the fill value generated will be: <i>ABCAB</i>. If the bit value is zero (0), the fill value generated for this field will be <i>XYXYX</i>.</p>
OffString	<p>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.</p> <p>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.</p>
Direction	<p>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.</p> <pre> 0000 0000 0000 0000 0000 0000 0000 0000 Bit 32 Bit 1 </pre>
AltLens	<p>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.</p> <p>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 <i>first bit</i> 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.</p>

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 image, 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 [Field Functions on page 67](#)

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	Defaults to...
Format	A valid date format string.	format 1
Day	A valid day value - must be an integer.	the current date
Month	A valid month value - must be an integer.	the current date
Year	A valid year value - must be an integer.	the current date

This function returns a date string that contains a formatted date value. If any values (day, month, or year) are omitted the appropriate value from the current date is substituted.

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 07/01/99.)

Function	Result	Explanation
Return(Date())	07/01/1999	No parameters entered, defaults to current date in date format 1.
Return(Date("4 4"))	July 1, 1999	Date format 4 selected, with a four-digit year length. Defaults to the current date in the selected format.
Return(Date(,18,5,1999))	05/18/1999	Defaults to date format 1 using the given values.
Return(Date("12",18,5))	99/138	Date format 1 selected with a two-digit year length. Enters the given date values in the selected format.

See also [Date Functions on page 58](#)

[Date Formats on page 59](#)

[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	Defaults to...
Date	A valid date string. Assumed to be in the format specified by the next parameter.	the current date
Format	A valid date format string. Describes the first parameter.	date format 1
NewFormat	A valid date format. Describes the desired new date format.	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 07/01/99 and the variable field called, *arc_date*, contains the hexadecimal value, *BC6792Do*)

Function	Result	Explanation
Return(Date2Date())	07/01/1999	No parameters entered—defaults to current date in date format 1.
Return(Date2Date("02/01/99", "1", "44"))	February 1, 1999	Changes the given date (02/01/99) from date format "1" to date format "4", with a four-digit year.
Return(Date2Date("99/138", "G"))	05/18/99	Changes the given date (99/138) 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, 2000	Converts the hexadecimal date to month name DD, YYYY without leading zeros.

See also [Date Functions on page 58](#)
[Date Formats on page 59](#)
[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	Defaults to...
Date	A valid date string. Assumed to be in the format specified by the next parameter.	the current date
Format	A valid date format string. Describes the first parameter.	date format 1
Days	A number of days - must be an integer.	0
Months	A number of months - must be an integer.	0
Years	A number of years - must be an integer.	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 02/28/2003, the result is 03/01/2004—not 02/29/2004.

To get 02/29/2004 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 07/01/99):

Function	Result	Explanation
Return(DateAdd (Date(), "1", 10))	07/11/1999	Defaults to the current date which is specified as Date() and adds 10 days.
Return(DateAdd (02/01/99, , 44))	10/01/2002	Uses the given date (02/01/99) and adds 44 months. (Note that if you enter "44" as a string, it is automatically converted to an integer.)
Return(DateAdd ("99/138", "1", , , -3))	96/139	The given date (99/138) using date format "1" is May 18, 1999. Subtracting three years results in the date May 18, 1996. Because 1996 was 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 58](#)
 [Date Formats on page 59](#)

DATECNV

Use this function to convert two-digit years into four-digit years.

Syntax **DateCnv (Date, Format, DivideYear, Century)**

Parameter	Description	Defaults to...
Date	A date string.	the current date
Format	A date format string describing the Date parameter.	date format 1
DivideYear	A dividing year value used to determine if the date value belongs to the specified century or the next.	the current year + 40
Century	The century to assign if the date falls in the dividing year. Otherwise, the result is this century plus one.	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 $((\text{current year} + 40) \% 100)$.

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 58](#)
 [Date Formats on page 59](#)
 [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)**

Parameter	Description	Defaults to...
Date	A valid date string. Assumed to be in the format specified by the next parameter.	the current date
Format	A valid date format string. Describes the first parameter.	date format 1

This function 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 07/01/99.)

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, 1999. This date is returned to the target variable <i>datestring</i> . The date is then used by the Day function and the integer value of 16 is returned.
Return(Day("99/ 138", "l"))	18	The given date (99/138) in date format l is May 18, 1999. Therefore, the integer value of 18 is returned.

See also [Date Functions on page 58](#)

[Date Formats on page 59](#)

[DateAdd on page 171](#)

DAYNAME

Use this function to enter the name of the day of the week.

Syntax **DayName (Dayofweek)**

Parameter	Description	Defaults to...
Dayofweek	An integer designating the day of the week. 1 = Sunday 2 = Monday 3 = Tuesday 4 = Wednesday 5 = Thursday 6 = Friday 7 = Saturday	the current day of the week

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 2, 1999.)

Function	Result	Explanation
Return(DayName())	Saturday	Defaults to the current day of the week and returns Saturday.
DayName(WeekDay("99/33", "I"))	Tuesday	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: Tuesday.
DayName(WeekDay(DateAdd(, -1)))	Monday	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 Monday, January 1, 1999 and returns the day name: Monday.

See also [Date Functions on page 58](#)
[Using INI Options on page 8](#)
[DateAdd on page 171](#)
[WeekDay on page 392](#)

DAYSINMONTH

Use this function to get the number of days in the specified month of a given year.

Syntax **DaysInMonth (Month, Year)**

Parameter	Description	Defaults to...
Month	A valid month number from 1 to 12.	the current month
Year	A valid year value - must be an integer.	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 07/01/99.)

Function	Result	Explanation
DaysInMonth ()	31	Defaults to the current date and returns the value 31 since July has 31 days.
DaysInMonth (Month ("04/15/ 1999"))	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, 1996)	29	The year 1996 was a leap year, February had 29 days. Therefore the integer 29 is returned.

See also [Date Functions on page 58](#)
[Month on page 297](#)

DAYSINYEAR

Use this function to get the number of days in the specified year.

Syntax **DaysInYear (Year)**

Parameter	Description	Defaults to...
Year	A valid year value. Must be an integer.	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 07/01/96.)

Function	Result	Explanation
DaysInYear ()	366	1996 is a leap year, therefore the returned value is 366.
DaysInYear (99)	365	The year 1999 is not a leap year and has 365 days.
DaysInYear (Year("1999/09/ 09", "34"))	365	First the Year function extracts the year number (1999) 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 58](#)
[Year on page 401](#)

DBAdd

Use this procedure/function to add a new record to a database table.

Syntax **DBAdd (Table, PrefixVariable)**

Parameter	Description	Defaults to...
Table	The name of an open table.	no default
PrefixVariable	Optional. The name of a DAL variable to associate with the record fields of the table.	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
<pre> RECORD.Company="Sky wire"; RECORD.Lob="Util"; RECORD.Rundate=DATE() ; DBAdd("APPIDX", "RECORD") </pre>	1 or 0	Assuming the table APPIDX has the columns <i>Company</i> , <i>Lob</i> , and <i>Rundate</i> , a new record will be added to the table whose values in those columns are Skywire, Util, and the current date, respectively.

See also [Database Functions on page 50](#)

DBCLOSE

Use this procedure/function to close a database table.

Syntax **DBCLOSE (Table)**

Parameter	Description	Defaults to...
Table	The name of the table to close.	no default

The procedure closes a table and returns one (1) if the table was successfully closed. Possible causes when a table can not be closed include:

- Table was not open, such as if it had already been closed
- 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 187](#)
[Setting Up Memory Tables on page 57](#)
[DBOpen on page 186](#)
[Database Functions on page 50](#)

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	Defaults to...
Table	The name of an open table.	no default
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 <i>Company</i> and <i>Lob</i> are valid key components for the APPIDX table, the procedure will delete all records with the value SAMPCO in the column named <i>Company</i> and the value Util in the column named <i>Lob</i> .

See also [Database Functions on page 50](#)

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	Defaults to...
Table	The name of an open table.	no default
Variable	The name of a DAL variable to associate with the record fields retrieved by the procedure.	Table
KeyName, KeyValue, ...	Each KeyName specifies a column to search. For some database handlers, it may be a <i>case-sensitive</i> . Each KeyValue is the value of the corresponding KeyName for which to search. For some database handlers, this may be a <i>case-sensitive</i> comparison.	no default (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","Skywire", "Lob", "DM")	1 or 0	Assuming that the APPIDX table has columns named <i>Company</i> and <i>Lob</i> , and that these columns are a key, the first record containing "Skywire" and "DM" in the appropriate column will be retrieved and associated with the prefix variable RECORD.

See also [Database Functions on page 50](#)
[DBPrepVars on page 188](#)

DBFIRSTREC

Use this procedure/function to retrieve the first record in a database table.

Syntax **DBFirstRec (Table, PrefixVariable)**

Parameter	Description	Defaults to...
Table	The name of an open table.	no default
PrefixVariable	The name of a DAL variable to associate with the record fields retrieved by the procedure.	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 50](#)
 [DBNextRec on page 185](#)

DBNEXTREC

Use this procedure/function to retrieve the next record in sequence from a database table.

Syntax **DBNextRec (Table, PrefixVariable)**

Parameter	Description	Defaults to...
Table	The name of an open table.	no default
PrefixVariable	The name of a DAL variable to associate with the record fields retrieved by the procedure.	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 50](#)
[DBFirstRec on page 184](#)

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 50](#) for more information.
- Tables open in the same database.

Syntax

DBOpen (Table, Handler, DFDFFile, Mode, Truncate)

Parameter	Description	Defaults to...
Table	The name of the table to open.	no default
Handler	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.
DFDFFile	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	The mode in which to open the file as a string. 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	Removes 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.	no default

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\m ytable.dfd", "READ&W RITE&TRUNCATE")		This DAL statement removes all rows from the table named <i>MYTABLE</i> .

See also [Setting Up Memory Tables on page 57](#)

[DBCclose on page 180](#)

[Database Functions on page 50](#)

DBPREPVARs

Use this procedure/function to create the DAL variables associated with a database table record.

Syntax **DBPrepVars (Table, PrefixVariable)**

Parameter	Description	Defaults to...
Table	The name of an open table.	no default
PrefixVariable	PrefixVariable is the name of the DAL variable to associate with the record fields retrieved by the procedure.	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 50](#)

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 50](#)

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	Defaults to...
Table	The name of an open table.	no default
Variable	Variable is the name of the stem variable containing the new information. This variable must first be filled by DBFind, DBFirstRec, or DBNextRec, after which you can modify individual fields before calling DBUpdate.	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.	no default (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
<code>DBFirstRec("APPIDX", RECORD");</code> <code>RECORD.RUNDATE = DATE();</code> <code>DBUpdate("APPIDX", "RECORD", "UNIQUE_I D", RECORD.UNIQUE_I D)</code>	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 50](#)
[DBFind on page 182](#)
[DBFirstRec on page 184](#)
[DBNextRec on page 185](#)

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 64](#)

DEC2HEX

This function returns the hexadecimal equivalent of an integer value.

Syntax **Dec2Hex(Value1, Digits)**

The value1 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.

The Digits 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 256](#)
[Bit/Binary Functions on page 49](#)

DEFORMAT

Use this function to remove formatting from a specified string and return the result.

Syntax **DeFormat (String, Fieldtype, Format)**

Parameter	Description	Defaults to...
String	A valid string of formatted text.	the value of current field text
Fieldtype	The field type indicator used to format the first parameter.	the value of current field type
Format	The format of the first parameter. This is the field format entered in the Properties window.	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 68](#)
[String Functions on page 84](#)

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 ()

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
```

Understanding the System

You can also remove blank or filler pages using custom code or by using the DPRDelBlankPages procedure, which is available with the Internet Document Server. See the [SDK Reference](#) 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 */
```

NOTE: See the [Documaker Server System Reference](#) for information on Using Banner Processing.

See also [AddBlankPages on page 107](#)
 [Page Functions on page 82](#)
 [SuppressBanner on page 376](#)
 [Miscellaneous Functions on page 80](#)
 [Creating a DAL Script Library on page 5](#)

DELFIELD

Use this procedure/function to delete a field from an image. The system only deletes the field if found and if it is not the current field.

Syntax **DelField (Field, Image, Form, Group)**

Parameter	Description	Defaults to...
Field	Name of a field.	the current field
Image	Name of an image that contains the field named.	the current image
Form	Name of a form that contains the image and/or field named.	the current form
Group	Name of the form group that contains the form, image, and/or field named.	the current form 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 the AFGJOB RP rules: PreTransDAL and PostTransDAL.

Example Lets assume you have the following forms in your form set; *Information* and *Multi-image* in the group named *DAL Test Company*.

The form named *Information* is comprised of two images; *Part1* and *Part2*. Part1 has these fields: abc1, abc2, and abc3. Part2 has these fields: abc3 and abc4.

The form named *Multi-image* is comprised of three images: Image1, Image2, and Image3. Image1 has objects with these field names: *a/n*, *date*, *yes/no*, and *multi-line*.

Image2 has the same objects with the same field names as Image1.

Image3 has following objects: *logo*, *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	<i>Abc3</i> on <i>Information/Part1</i> is deleted because the image, field, and group parameters were omitted specified. The system defaulted to the current image, form, and group.
Return(DelField ("abc3", "part2"));	1	Abc3 on <i>Information/Part2</i> is deleted because you specified Part2 and the form defaulted to the current form, <i>Information</i> . Note that <i>Abc3</i> will still exist on <i>Information/Part1</i> .
Return(DelField ("test"));	0	<i>Test</i> is not deleted because it is the current field.

Procedure	Result	Explanation
Return(DelField ("a/n"));	0	The field <i>a/n</i> is not deleted because it is not on <i>Information/Part1</i> .
Return(DelField ("a/n", "image1"));	0	The field <i>a/n</i> is not deleted because image1 is not a field on the current form (Information).
Return(DelField ("a/n", "image1", "Multi-image"));	1	The field <i>a/n</i> on <i>Multi-image/Image1</i> is deleted because this field is on the specified form/image.
Return(DelField ("a/n", , "Multi-image"));	1	The field <i>a/n</i> on <i>Multi-image/Image1</i> is deleted because field is on the specified form and the image parameter defaults to the first image on the form. Field <i>a/n</i> on <i>Multi-image/Image1</i> will still exist. If you immediately execute the script again, the field <i>a/n</i> on <i>Image2</i> would be deleted.
Return(DelField ("a/n", , , "DAL Test Company"));	1	The field <i>a/n</i> on <i>Multi-image/Image1</i> is deleted because it was is the first field in the group, <i>DAL Test Company</i> . Field <i>a/n</i> on <i>Multi-image/Image2</i> will still exist. If you immediately execute the script again, the field <i>a/n</i> on <i>Image2</i> would be deleted.
Return(DelField ("box", "image3", "Multi-image"));	0	The field <i>Box</i> 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 DelLogo to delete logos.

See also [Field Functions on page 67](#)
[Locating Fields on page 70](#)
[DelLogo on page 201](#)

DELFORM

Use this procedure/function to remove a form from the document.

Syntax **DelForm (Form, Group)**

Parameter	Description	Defaults to...
Form	Name of an existing form.	no default
Group	Name of a group to contain the specified form.	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 [Image Functions on page 76](#)

DELIMAGE

Use this procedure/function to remove an image from a form. You can use the Paginate parameter to specify whether form pagination should occur after the image is deleted.

Syntax

DellImage (Image, Form, Group, Paginate)

Parameter	Description	Defaults to...
Image	The name of an existing image.	no default
Form	Name of a form within the form set.	the current form
Group	Name of a group to contain the specified form.	the current group
Paginate	<p>(optional) This parameter follows the Group parameter. If you enter anything other than a zero (o), it tells the system that you want form pagination to occur upon the successful removal of the image.</p> <p>If you omit this parameter or enter zero(o), the image is deleted, but no other images are moved to occupy the space left vacant. Subsequent form re-pagination and the application of image origins may change the layout of the form.</p> <p>Here is an example:</p> <pre>DelImage("myImage", , , 1)</pre> <p>This example omits the Form and Group parameters, but does include the Paginate parameter.</p> <p>Note: If you enter zero (o) or omit this parameter, the function works as it prior to version 11.2.</p>	zero (o)

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

This procedure removes the specified image from the form. It cannot delete the current image. You can delete any image on the current form, as long as it is not the current image.

If the deleted image is the only image on that page, the system also removes the page from the form. If other images occur on that page, space occupied by the deleted image is left blank.

NOTE: Removing an image means that all data associated with that image 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
<code>DellImage("IMG")</code>	1 or 0	Delete the specified image from the current form. This assumes that the named image is not the current image.
<code>DellImage("IMG\3", ,"GRP")</code>	1 or 0	Locate the third occurrence of IMG in the specified GRP. If this is not the current image, delete the image.

See also [AddImage on page 114](#)
[PaginateForm on page 309](#)
[Image Functions on page 76](#)

DELLOGO

Use this procedure/function to delete a logo (bitmap) from a form in the current form set.

Syntax DelLogo (Logo, Image, Form, Group)

Parameter	Description	Defaults to...
Logo	Name of the logo to be deleted from an image or form. Logo names are assigned in the Image Editor.	no default
Image	Name of an image that contains the specified logo.	the current image
Form	Name of a form that contains the image.	the current form
Group	Name of a group to use to locate the specified object.	the current group

This procedure deletes the specified logo from the image 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("Img1")	1 or 0	Deletes Img1 on the current image, form, group.
DelLogo("New1", "IMH1\3", "UpRate")	1 or 0	Deletes New1 on the 3rd occurrence of the named image, IMH1 on the form, UpRate, in the default group.

See also [ChangeLogo on page 149](#)

[HaveLogo on page 254](#)

[InlineLogo on page 262](#)

[RenameLogo on page 335](#)

[Logo on page 279](#)

[Refresh on page 333](#)

[Logo Functions on page 78](#)

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 the 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 Supervisor Guide](#)
[Documaker User Guide](#)

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()

Here is an example of the logic that might appear in 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...

- 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-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. DeviceName and UniqueString are applicable to both Entry and Documaker Server.

See also [Printer/Recipient Functions on page 83](#)

[BreakBatch on page 146](#)

[SetDeviceName on page 349](#)

[UniqueString on page 389](#)

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	Defaults to...
Date1	A valid date string. Assumed to be in the format specified by the next parameter.	the current date
Format1	A valid date format string. Describes the first parameter (date1).	date format 1
Date2	A valid date string. Assumed to be in the format specified by the next parameter.	the current date
Format2	A valid date format string. Describes the third parameter (date2).	date format 1

This function 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 58](#)
[Date Formats on page 59](#)

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	Defaults to...
Date1	A valid date string. Assumed to be in the format specified by the next parameter.	the current date
Format1	A valid date format string. Describes the first parameter (date1).	date format 1
Date2	A valid date string. Assumed to be in the format specified by the next parameter.	the current date
Format2	A valid date format string. Describes the third parameter (date2).	date format 1

This function 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 58](#)

[Date Formats on page 59](#)

[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	Defaults to...
Time1	A valid time string. Assumed to be in the format specified by the next parameter.	the current time
Format1	A valid time format string. Describes the first parameter (time1).	time format 1, which is HH:MM:SS
Time2	A valid time string. Assumed to be in the format specified by the next parameter.	the current time
Format2	A valid time format string. Describes the third parameter (time2).	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 86](#)

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	Defaults to...
Time1	A valid time string. Assumed to be in the format specified by the next parameter.	the current time
Format1	A valid time format string. Describes the first parameter (time1).	time format 1
Time2	A valid time string. Assumed to be in the format specified by the next parameter.	the current time
Format2	A valid time format string. Describes the third parameter (time2).	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 86](#)

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	Defaults to...
Date1	A valid date string. Assumed to be in the format specified by the next parameter.	the current date
Format1	A valid date format string. Describes the first parameter (date1).	date format 1
Date2	A valid date string. Assumed to be in the format specified by the next parameter.	the current date
Format2	A valid date format string. Describes the third parameter (date2).	date format 1

This function 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. This function always enters 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 58](#)
 [Date Formats on page 59](#)
 [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	Defaults to...
Time1	A valid time string. Assumed to be in the format specified by the next parameter.	the current time
Format1	A valid time format string. Describes the first parameter (time1).	time format 1
Time2	A valid time string. Assumed to be in the format specified by the next parameter.	the current time
Format2	A valid time format string. Describes the third parameter (time2).	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 86](#)

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	Defaults to...
Time1	A valid time string. Assumed to be in the format specified by the next parameter.	the current time
Format1	A valid time format string. Describes the first parameter (time1).	time format 1
Time2	A valid time string. Assumed to be in the format specified by the next parameter.	the current time
Format2	A valid time format string. Describes the third parameter (time2).	time format 1

This function 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 86](#)

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	Defaults to...
Date1	A valid date string. Assumed to be in the format specified by the next parameter.	the current date
Format1	A valid date format string. Describes the first parameter (date1).	date format 1
Date2	A valid date string. Assumed to be in the format specified by the next parameter.	the current date
Format2	A valid date format string. Describes the third parameter (date2).	date format 1

This function calculates the number of complete years between the given dates. For example, from 2/10/99 to 2/10/00 is considered one year, while 3/1/99 to 2/29/00 is considered zero years. This function always results in 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 07/01/99):

01/31/2000 to 01/30/2001 = zero years difference (it will not be a year until 01/31/2001 as the year 2000 is a leap year)

Function	Result	Explanation
DiffYears ("7/15/99")	0	The second parameter defaults to the current date. Since the value is not an entire year, the result is zero (0).
DiffYears ("01/31/2000", "4", "01/30/2001", "1")	0	The result will not become one (1) until January 31, 2001.
DiffYears ("01/01/95", "1")	4	The second parameter defaults to the current date.
DiffYears ("October 31, 1965", "4", "10/31/99", "1")	34	Note that the result includes numerous years. In addition, two different date formats are used.

See also [Date Functions on page 58](#)
 [Date Formats on page 59](#)
 [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 for that form in the Form Set Manager.

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 111](#)

[AddForm_Propagate on page 112](#)

[CopyForm on page 160](#)

[WIP Functions on page 88](#)

EMBEDLOGO

Use this procedure/function to save logo data, including full color data, inside the NAFILE.DAT file. This lets you capture and archive form set specific image data such as pictures, scans, or signatures along with the form set.

Syntax **EmbedLogo (Logo,ImageName,FormName,GroupName)**

Parameter	Description
Logo	The name of the logo you want to embed.
ImageName	The name of an image that contains the logo. If the current image does not contain the logo being referenced this parameter is required to locate the image; otherwise ImageName is optional.
FormName	The name of the form that contains the logo being referenced. If the current form does not contain the image for the logo being referenced this parameter is required to locate the logo; otherwise, FormName is optional.
GroupName	The name of the form group that contains the logo being referenced. If the current form is not in the form group that contains the logo being referenced this parameter is required to locate the logo; otherwise, GroupName is optional.

Execute this DAL procedure for each logo on the form or image. This procedure sets the embedded logo flag in the logo 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 logo 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 image level rule, CheckImageLoaded.

Example Here is an example:

Procedure	Result	Explanation
rc = EmbedLogo("JaneDoe");	1	The embedded logo flag in the JaneDoe bitmap structure will be set to On.

See also [Image Functions on page 76](#)

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	Defaults to
Symbol	Accepts a string that specifies the name of a DAL symbolic variable. This can be from an expression or from another string variable.	None, but you must make an entry.

This function returns True (1) if the variable exists. It returns False (0) if there is no such symbol defined.

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 245](#)
 [Miscellaneous Functions on page 80](#)

FIELDFORMAT

Use this function to return the format string associated with the field's type.

Syntax **FieldFormat (Field, Image, Form, Group)**

Parameter	Description	Defaults to...
Field	Name of a field.	the current field
Image	Name of an image that contains the field named.	the current image
Form	Name of a form that contains the image and/or field named.	the current form
Group	Name of the form group that contains the form, image, and/or field named.	the current form 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 67](#)

[Field Formats on page 68](#)

[Locating Fields on page 70](#)

FIELDNAME

Use this function to return the name of a field relative to another field.

Syntax **FieldName (Count, Field, Image, Form, Group)**

Parameter	Description	Defaults to...
Count	A positive or negative number used to move beyond the field specified.	0
Field	Name of a field.	current field
Image	Name of an image that contains the field named.	the current image
Form	Name of a form that contains the image and/or field named.	the current form
Group	Name of the form group that contains the form, image, and/or field named.	the current form 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, image, 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 will first locate the specified field. If the fieldname parameter is not specified, the current field will be assumed. Then the count will be used to move to another field on the image.

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 image's edit list, not necessarily to physical position on the image. 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 image 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.
Return(Field Name (-1, "Second"))	First	Locate the field named Second and then move to the previous field.

Function	Result	Explanation
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 81](#)
[Field Formats on page 68](#)
[Locating Fields on page 70](#)

FIELDPROMPT

Use this function to return the text of the prompt for a field.

Syntax

FieldPrompt (Field, Image, Form, Group)

Parameter	Description	Defaults to...
Field	Name of a field.	the current field
Image	Name of an image that contains the field named.	the current image
Form	Name of a form that contains the image and/or field named.	the current form
Group	Name of the form group that contains the form, image, and/or field named.	the current form group

The value returned from this function is a string. If the system cannot find a field that matches the specified information, the system returns an empty string.

NOTE: For optimal performance, avoid using wildcards (*) when searching for field, image, or form names.

Example

Here are some examples:

Function	Result	Explanation
Return(FieldPrompt ("Name"))	Name	Locates the field on the current image and returns its prompt.
Return(FieldPrompt ("Address1"))	Street Address	Locates the field on the current image and returns its prompt.

See also

[Field Functions on page 67](#)

[Field Formats on page 68](#)

[Locating Fields on page 70](#)

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 an image 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 ()

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 in a DDT file. 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_i  
t:::!/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
Format mask	*	blank

Parameter	Description	Example
Field rule name	*	KickToWip
Rule parameters	* (also called <i>data</i>)	blank
Flag1	(also called <i>not required</i>)	N
Flag2	(also called <i>host required</i>)	N
Flag3	(also called <i>operator required</i>)	Y
Flag4	(also called <i>either required</i>)	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 (“oooo”) 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 64](#)

[Field Formats on page 68](#)

[Locating Fields on page 70](#)

FIELDTYPE

Use this function to return the data type information associated with the image field.

Syntax **FieldType (Field, Image, Form, Group)**

Parameter	Description	Defaults to...
Field	Name of a field.	the current field
Image	Name of an image that contains the field named.	the current image
Form	Name of a form that contains the image and/or field named.	the current form
Group	Name of the form group that contains the form, image, and/or field named.	the current form 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 <i>K</i> 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, <i>M</i> corresponds with the X or Space field type.

See also [Field Functions on page 67](#)
 [Field Formats on page 68](#)
 [Locating Fields on page 70](#)

FIELDX

Use this function to return the X coordinate of a variable field object.

Syntax **FieldX (Field, Image, Form, Group)**

Parameter	Description	Defaults to...
Field	Name of an image field	the current field name
Image	Name of an image that contains the field named.	the current image
Form	Name of a form that contains the image and/or field named.	the current form
Group	Name of the form group that contains the form, image, and/or field named.	the current form 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 image.
Return(FieldX("MyField", "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 67](#)
[Field Formats on page 68](#)
[Locating Fields on page 70](#)
[FieldY on page 224](#)

FIELDY

Use this function to return the Y coordinate of a variable field object.

Syntax **FieldY (Field, Image, Form, Group)**

Parameter	Description	Defaults to...
Field	Name of an image field	the current field name
Image	Name of an image that contains the field named.	the current image
Form	Name of a form that contains the image and/or field named.	the current form
Group	Name of the form group that contains the form, image, and/or field named.	the current form 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 image.
Return(FieldY("MyField", , "FRM"))	6020	Returns the first occurrence of MyField on the specified form.

See also [Field Functions on page 67](#)
[Field Formats on page 68](#)
[Locating Fields on page 70](#)
[FieldX on page 223](#)

FILEDRIVE

Use this function to get the drive component of a file name.

Syntax **FileDrive("FullFileName")**

This function accepts a string containing a fully qualified file name, returns a string that contains the drive component of that file name.

Here is an example:

```
MYDRIVE = FileDrive("d:\mypath\myfile.ext")
```

In this example, MYDRIVE would contain:

```
"d:"
```

See also [FilePath on page 228](#)
 [FileName on page 227](#)
 [FileExt on page 226](#)
 [FullFileName on page 234](#)
 [File/Path Functions on page 74](#)

FILEEXT

Use this function to get the extension component of a file name.

Syntax **FileExt("FullFileName")**

This function accepts a string containing a fully qualified file name, returns a string that contains the extension component of that file name.

Here is an example:

```
MYEXT = FileExt("d:\mypath\myfile.ext")
```

In this example MYEXT would contain:

“.ext”

See also [File/Path Functions on page 74](#)
 [FullFileName on page 234](#)
 [FileDrive on page 225](#)
 [FilePath on page 228](#)
 [FileName on page 227](#)

FILENAME

Use this function to get the name component of a file name.

Syntax `FileName("FullFileName")`

This function accepts a string containing a fully qualified file name, returns a string that contains the name component of that file name.

Here is an example:

```
MYNAME = FileName("d:\mypath\myfile.ext")
```

In this example, MYNAME would contain:

"myfile"

See also [File/Path Functions on page 74](#)

[FullFileName on page 234](#)

[FileDrive on page 225](#)

[FilePath on page 228](#)

[FileExt on page 226](#)

FILEPATH

Use this function to get the path component of a file name.

Syntax **FilePath("FullFileName")**

This function accepts a string containing a fully qualified file name, returns a string that contains the path component of that file name.

Here is an example:

```
MYPATH = FilePath("d:\mypath\myfile.ext")
```

In this example, MYPATH would contain:

"\mypath\"

See also [File/Path Functions on page 74](#)
 [FullFileName on page 234](#)
 [FileDrive on page 225](#)
 [FileExt on page 226](#)
 [FileName on page 227](#)

FIND

Use this function to return the position of a substring within another string.

Syntax Find (String, Substring, Integer)

Parameter	Description	Defaults to...
String	A valid string.	the value of current field text
Substring	A string of one or more characters that will be located in parameter one.	no default
Integer	0 = a left to right search 1 = a right to left search	zero (0)

This function's default search order is from left to right. You can also specify a right to left search order. Both search methods return a position relative to the first (left-hand) character of the string parameter. A zero (0) is returned if the substring is not found in the search string. 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 84](#)

FORMAT

Use this function to format a string field and return the result.

Syntax **Format (String, FieldType, Format)**

Parameter	Description	Defaults to...
String	A valid string of non-formatted text.	the value of current field text
FieldType	The field type indicator to be used to format the first parameter.	the value of current field type
Format	The format to be used to format the first parameter.	the value of current field format

This function 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 68](#)
[String Functions on page 84](#)

FORMDESC

Use this function to retrieve the description specified in the 111 file for a specific form.

Syntax **FormDesc (Count, StartForm, Group)**

Parameter	Description	Defaults to...
Count	An index reference to locate a form before or after the specified form. To move backwards, enter a negative number.	zero (0)
StartForm	Name of a form from which to start the search.	the current form
Group	Name of a group which contains the specified form.	the current group

This function lets you get the description specified in the 111 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 forward or backward 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 function 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 232](#)

[ImageName on page 258](#)

[Name Functions on page 81](#)

[DAL Script Examples on page 43](#)

FORMNAME

Use this function to get the name from a form. This function returns the name of the form located.

Syntax **FormName (Count, Startform, Group)**

Parameter	Description	Defaults to...
Count	Any index reference to locate a form before or after the specified form.	zero (0)
Startform	Name of a form from which to start the search.	the current form
Group	Name of a group to contain the specified form.	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, image, or form names.

This function lets you get the name of a form relative to a known form. If you omit all parameters, the function returns the name of the current form. The Count parameter moves a number of forms forward 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 function 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 231](#)
[Name Functions on page 81](#)

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	Defaults to...
Number	any valid amount.	the current field value
Dollarword	any word that should be used rather than dollars.	“dollars et”
Centword	any word that should be used rather than cents.	“cents”
Decimode	1 = numeric decimal amount 2 = spell decimal amount 3 = suppress o, numeric decimal amount 4 = suppress zero, spell decimal amount	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 o cents

(Assume the current field value is 2,000,000.)

Function	Result	Explanation
Return(FrenchNumText ())	deux millions de dollars et o cents	The current field value is returned in a written form using <i>dollars et</i> and <i>cents</i> . The zero (o) is displayed in a numeric decimal amount format.
Return(FrenchNumText (123.45,,,2))	cent vingt-trois dollars et quarante-cinq cents	The written equivalent for 123.45 is displayed using Decimode 2 with the decimal spelled out.

See also [String Functions on page 84](#)
[NumText on page 303](#)

FULLFILENAME

Use this function to make the full file name.

Syntax `FullFileName("Drive","Path","Name","Ext")`

This function 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 an Z/OS example:

```
FullFileName(,"DD:DEFLIB()", "MEMBER")
```

In this example, the result would be:

DD:DEFLIB(MEMBER)

See also [File/Path Functions on page 74](#)

[FileDrive on page 225](#)

[FileExt on page 226](#)

[FileName on page 227](#)

[FilePath on page 228](#)

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, DSI Queue)**

Parameter	Description
Name	The name of the attachment variable.
DSI Queue	(optional) Enter one (1) for input or two (2) for output. The default is 1.

See also [AddAttachVAR on page 106](#)
 [RemoveAttachVAR on page 334](#)
 [Docupresentment Functions on page 66](#)

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, Occurrence)

Parameter	Description	Defaults to...
SearchMask	Defines what data to look for.	none
Occurrence	Lets you specify which occurrence of the data to get.	the first occurrence

GetData returns the data from the extract file based on the search mask. Format the search mask as shown here:

“extract search mask offset, length”

Example

Here is an example:

```
GetData( "11,HEADERREC 40,17")
```

In this example, the GetData function finds the extract record designated by “11,HEADERREC” 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 367](#)
[Documaker Server Functions on page 64](#)

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	Defaults to...
Name	Name of the user attributes (metadata) to retrieve.	No default
Form	Name of a form from which to retrieve data.	Current form
Group	Name of the group that contains the specified form	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 an image 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 an image 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 323](#)
[Miscellaneous Functions on page 80](#)

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	Required?
Context	A name (valid name) associated to a set of INI control groups and options that have been loaded into cache memory.	Optional
Group	The group name (valid string) which contains the INI option Boolean value to retrieve.	Yes
Option	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 (o).	Yes
Default	Default string value to return from the function instead of the actual control group and option value.	Optional

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.

The system returns one (1) if no error occurs, otherwise a zero (o) is returned.

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
<code>bool_value = GetINIBool (,"Control", "LogEnabled");</code>	The variable <i>bool_value</i> now contains a zero (o).	The function scanned the loaded INI control groups and options. It found the specified control group and option in the <i>FSIUSER.INI</i> first. The <i>FSIUSER.INI</i> set is searched first, followed by the <i>FSISYS.INI</i> set and then any other loaded sets, in order.

Function	Result	Explanation
<code>bool_value = GetINIBool ("MVF", "Control", "LogEnabled");</code>	The variable <i>bool_value</i> now contains a one (1).	The function scans only the control group and option set associated with the context name <i>MVF</i> .
<code>bool_value = GetINIBool ("MVF", "Control", "LogEnabled", 1);</code>	The variable <i>bool_value</i> now contains a one (1). If <i>Control</i> and <i>LogEnabled</i> are not found, <i>string_value</i> is set to zero (0).	The function scans only the control group and option set associated with the context name <i>MVF</i> .

See also [INI Functions on page 77](#)
[Using INI Options on page 8](#)
[GetINIString on page 242](#)
[LoadINIFile on page 277](#)

GETINISTRING

Use this function to retrieve from cache memory the identified INI control group and option string.

Syntax **GetINIString (Context, Group, Option, Default);**

Parameter	Description	Required?
Context	A name (valid string) associated to a set of INI control groups and options which have been loaded into cache memory.	Optional
Group	The control group name (valid string) which contains the INI option string to retrieve.	Yes
Option	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.	Yes
Default	Default string value to return from the function instead of the actual control group and option value.	Optional

This function retrieves the specified control group and option string.

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.

The function returns one (1) if no error occurs, otherwise a zero (0) is returned.

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 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 <i>string_value</i> now contains this string: <i>Test group 1</i>	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 <i>string_value</i> now contains this string: <i>MVF’s group/option</i>	The function scans only the control group and option set associated with the context name <i>MVF</i> .
string_value = GetINIString (“MVF”, “Control”, “Title”, “Bob’s group/option”);	The variable <i>string_value</i> now contains this string: <i>MVF’s group/option</i> If <i>Control</i> and <i>Title</i> are not found, <i>string_value</i> is set to: <i>Bob’s group/option</i>	The function scans only the control group and option set associated with the context name <i>MVF</i> .

See also [INI Functions on page 77](#)
[Using INI Options on page 8](#)
[GetINIBool on page 240](#)

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	Required?
Form	The name of the form that contains the fields on which overflow processing will occur.	Yes
Symbol	The name you want to use as the overflow symbol.	Yes

This function 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 119](#)
[IncOvFlwSym on page 261](#)
[ResetOvFlwSym on page 337](#)
[Documaker Server Functions on page 64](#)

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	Defaults to
Symbol	Accepts a string that specifies the name of a DAL symbolic variable. This can be from an expression or from another string variable.	None, but you must make an entry.

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 image:

```
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 215](#)
[Miscellaneous Functions on page 80](#)

GROUPNAME

Use this function to get the name from a group of forms. This function returns the name of the group located.

Syntax **GroupName (Count, StartGroup)**

Parameter	Description	Defaults to...
Count	An index reference to locate a group before or after the specified group. Enter a negative number to move backwards.	0
StartGroup	Name of a group from which to start the search.	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, image, or form names.

This function 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()	<i>GROUPONE</i>	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 81](#)

GVM

Use this function to retrieve the contents of a GVM variable. The system returns the content of the variable if it exists or a blank string if not.

Syntax **GVM (Name, Instance)**

Parameter	Description	Defaults to...
Name	Required. The name of the GVM variable.	no default
Instance	The instance number of the GVM variable.	1

Example Here is an example:

Function	Result	Explanation
<pre>If (HaveGVM('Company')) AddComment(GVM('Company')) End</pre>	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 64](#)

[HaveGVM on page 252](#)

[AddComment on page 109](#)

[DAL Script Examples on page 43](#)

[SetGVM on page 355](#)

HAVEFIELD

Use this function to determine if a specified field can be located.

Syntax **HaveField (Field, Image, Form, Group)**

Parameter	Description	Defaults to...
Field	Name of a field.	the current field
Image	Name of an image that contains the field named.	the current image
Form	Name of a form that contains the image and/or field named.	the current form
Group	Name of the form group that contains the form, image, and/or field named.	the current form 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, image, or form names.

This function searches for the specified field on a particular image, 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 image 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 75](#)
 [Field Formats on page 68](#)
 [Locating Fields on page 70](#)

HAVEFORM

Use this function to determine if a given form is contained in the document.

Syntax **HaveForm (Form, Group)**

Parameter	Description	Defaults to...
Form	Name of an existing form.	the current form
Group	Name of a group to contain the specified form.	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, image, 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 75](#)

HAVEGROUP

Use this function to determine if a given group is part of a document. This function returns one (1) if the group is located and zero (0) if it cannot be found.

Syntax HaveGroup (Group)

Parameter	Description	Defaults to...
Group	Name of a group to locate.	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, image, 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 75](#)

HAVEGVM

Use this function to determine if a GVM variable exists. The system returns one (1) if it locates the GVM variable or a zero (0) if it cannot find the variable.

Syntax **HaveGVM (Name, Instance)**

Parameter	Description	Defaults to...
Name	Required. The name of the GVM variable.	no default
Instance	The instance number of the GVM variable.	1

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 64](#)

[GVM on page 247](#)

[AddComment on page 109](#)

[DAL Script Examples on page 43](#)

[SetGVM on page 355](#)

[GVM on page 247](#)

HAVEIMAGE

Use this function to determine if a given image is contained in the document. This function returns one (1) if the form is located and zero (0) if it cannot be found.

Syntax **HavelImage (Image, Form, Group)**

Parameter	Description	Defaults to...
Image	Name of an image to locate.	the current image
Form	Name of a form that is assumed to contain the specified image.	the current form
Group	Name of a group to contain the specified image or form.	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, image, or form names.

Several of the DAL functions might return a value that may indicate an image is or is not a part of the document. However, those functions also intend to perform some other procedure beyond searching for the image. This function simply identifies whether a given image 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 image.

Example Here are some examples:

Function	Result	Explanation
HavelImage ("IMG")	1 or 0	Attempts to locate the named image on the current form. If found, return 1.
HavelImage ("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 75](#)

[Where DAL Functions are Used on page 92](#)

HAVELOGO

Use this function to determine if a logo (bitmap) exists on an image or form which is in the current form set. This function returns one (1) if it finds the logo and zero (0) if it does not.

Syntax HaveLogo (Logo, Image, Form, Group)

Parameter	Description	Defaults to...
Logo	Name of the logo to find. Logo names are assigned in the Image Editor.	no default
Image	Name of an image that contains the logo you specified.	the current image
Form	Name of a form that contains the image you specified.	the current form
Group	Name of a group to use to locate the specified object.	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, image, or form names.

Example Here are some examples:

Function	Result	Explanation
HaveLogo("Log1")	1 or 0	Determines if Log1 exists on the current image, form, group.
HaveLogo("Log01", "IMH1\3", "UpRate")	1 or 0	Determines if Logo1 exists on the 3rd occurrence of the image, IMH1, on the form, UpRate, within the default group.

See also [ChangeLogo on page 149](#)
[DelLogo on page 201](#)
[HaveField on page 248](#)
[HaveForm on page 250](#)
[HaveGroup on page 251](#)
[HaveImage on page 253](#)
[InlineLogo on page 262](#)
[Logo on page 279](#)
[RenameLogo on page 335](#)
[Have Functions on page 75](#)

HAVERECIP

Use this function to see if the specified recipient name is defined in the form set for the specified image, form, or group. This function returns one (1) if true or a zero (0) if false.

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, Image, Form, Group)

Parameter	Description	Defaults to...
Recipient	Name of a recipient.	no default
Image	Name of an image that contains the field named.	the current image
Form	Name of a form that contains the image and/or field named.	the current form
Group	Name of the form group that contains the form, image, or field.	the current group

NOTE: You must enter a recipient name.

See also [RecipientName on page 331](#)
 [Have Functions on page 75](#)

HEX2DEC

This function returns the integer equivalent of a hexadecimal string.

Syntax **Hex2Dec(value1)**

The parameter specifies a string of characters that are to be 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 193](#)
 [Bit/Binary Functions on page 49](#)

Hour

Use this function to extract the number of hours from a time.

Syntax **Hour (Time1, Format1)**

Parameter	Description	Defaults to...
Time1	A valid time string. Assumed to be in the format specified by the next parameter.	the current time
Format1	A valid time format string. Describes the first parameter (time1).	time format 1

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 86](#)

IMAGENAME

Use this function to get the name of an image. This name is returned.

Syntax **ImageName (Count, Startimage, Form, Group)**

Parameter	Description	Defaults to...
Count	An index reference to locate a form before or after the specified form.	0
Startimage	Name of an image from which to begin the search.	the current image
Form	Name of a form containing the requested image.	the current form
Group	Name of a group to contain the specified form.	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, image, or form names.

This function returns the name of an image relative to another image on the same form. If no parameters are provided to this function, the current image's name will be returned. The count parameter will move a number of images forward or backwards (negative) from a located image before returning the image name.

In the event the starting image cannot be located or the count parameter causes the system to move beyond the number of images contained on the form, this function returns an empty string.

If there is more than one copy of an image on the located form, the name returned will contain the occurrence notation used by DAL functions to locate images. 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 image is named *IMG*.)

Function	Result	Explanation
ImageName()	IMG	No parameters will result in returning the current image name.
ImageName(2, "IMG", "FormC")		Locate FORMC in the current group. Next, locate IMG on that form. Then, return the name of the image two positions beyond the located image.

See also [Name Functions on page 81](#)

IMAGERECT

Use this procedure/function to retrieve the rectangular coordinates of an image in a form set (document).

Syntax **ImageRect (PrefixVariable, Image, Form, Group)**

Parameter	Description	Defaults to...
PrefixVariable	Coordinates for the image.	See below.
Image	Name of an image in the form set.	the current image.
Form	Name of the form that contains the image.	the current form.
Group	Name of the form group that contains the form and image.	the current form group.

This procedure gets the coordinates for the image 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 image 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=10 0 MyImage.right=200	The procedure returns the coordinates for the current image (<i>Image25</i>) on the current form in the current form group. If it does not exist, the procedure returns zero (0).
IMAGERECT ("MyImage", "Image50")	Internal variables equal: MyImage.top=125 MyImage.left=150 MyImage.bottom=20 0 MyImage.right=200	The procedure returns the coordinates for <i>Image50</i> 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 image <i>MVF</i> on the form <i>XYZ</i> into the DAL target variables. If it does not exist, the procedure returns zero (0).

See also [Image Functions on page 76](#)
[SetImagePos on page 356](#)

INCovFLWSYM

Use this procedure/function to increment an overflow symbol. This procedure provides DAL the Documaker Server equivalent to image level rule, IncOvFlwSym that you place in the image DDT file, with the exception that it would only increment by one.

Syntax IncOvFlwSym (Form, Symbol)

Parameter	Description	Required?
Form	The name of the form that contains the fields on which overflow processing will occur.	Yes
Symbol	The name you want to use as the overflow symbol.	Yes

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

Syntax [AddOvFlwSym on page 119](#)
 [GetOvFlwSym on page 244](#)
 [ResetOvFlwSym on page 337](#)
 [Documaker Server Functions on page 64](#)

INLINELOGO

Use this procedure/function to cause a logo (bitmap) to be *in-lined* in the print stream. This means you do not have to store the logo as a printer resource on the printer.

Syntax **InlineLogo(Logo,Option,Image,Form,Group)**

Parameter	Description	Defaults to...
Logo	Name of the logo to be in-lined in the print stream. Logo names are assigned in the Image Editor.	no default
Option	Sets the inline flag. One (1) equals On; zero (0) equals Off.	1 is the default
Image	Name of an image that contains the specified logo.	the current image
Form	Name of a form that contains the image.	the current form
Group	Name of a group to use to locate the specified object.	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 image, 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 image, IMH1, on the form, UpRate) into the print stream.

See also [ChangeLogo on page 149](#)
[DelLogo on page 201](#)
[HaveLogo on page 254](#)
[Logo on page 279](#)
[RenameLogo on page 335](#)
[Logo Functions on page 78](#)

INPUT

Use this function to create a window with a title and a prompt which asks the user to enter information. This function returns the input results.

Syntax **Input (Prompt, Title, Length, DefText)**

Parameter	Description	Defaults to...
Prompt	A text string to assign as the prompt for the field.	Text
Title	A text string to assign as the title of the window.	Title
Length	Maximum input text length.	Windows defaults
DefText	A text string to assign as the default input data.	no default

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 <i>Name Entry</i> . The user sees the prompt <i>Please enter your name:</i> . 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 <i>Confirm this result</i> . 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 65](#)

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	Defaults to...
String	A valid string.	the value of current field text
Position	Position in the field to perform the insert.	one
Substring	A valid string to insert.	No default

This function adds the substring to the string 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 <i>Type</i> .
Return(Insert (, CFIND (, " "), " First"))	Your First Name	First the CFind function locates a blank space at position 5 within <i>Your Name</i> . Next, <i>First</i> is inserted at position 5.
Return(Insert ("Complete blank.", 10, "every "))	Complete every blank.	Goes to position 10 and inserts <i>every</i> .
Return(Insert ("Complete blanks", 17, "with black ink."))	Complete blanks with black ink.	Increases the length of the field to 17 and appends <i>with black ink</i> .

See also [String Functions on page 84](#)
[CFind on page 148](#)

INT

Use this function to return the integer portion of a number.

Syntax **INT (Number)**

Parameter	Description	Defaults to...
Number	Any valid numeric data type	the integer value of the current field

This function 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, <i>\$TEMP</i> will be 99.99 and <i>#RESULT</i> 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 79](#)

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	Defaults to...
String	A valid string.	the value of the current field text.
Length	Desired length of output.	the length of the input string.

This function 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 269](#)

[String Functions on page 84](#)

[Size on page 362](#)

JLEFT

Use this function to left justify text within a specified length and return the result.

Syntax **JLeft (String, Length)**

Parameter	Description	Defaults to...
String	A valid string.	the value of the current field text
Length	Desired length of output.	the length of the input string

This function 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 <i>Name</i> and the X.

See also [String Functions on page 84](#)

[@ on page 101](#)

[Size on page 362](#)

JRIGHT

Use this function to right justify text within a specified length and return the result.

Syntax **JRight (String, Length)**

Parameter	Description	Defaults to...
String	A valid string.	the value of the current field text
Length	Length of output.	the length of the input string

This function 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 <i>Name</i> .

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 84](#)
 [@ on page 101](#)
 [Size on page 362](#)

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, Image, Form, Group)

Parameters	Description	Defaults to
Mode	Enter L (left), R (right), or C (center)	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	The name of the field.	current field
Image	The name of the image that contains the field.	current image
Form	The name of the form that contains the image and/or field.	current form
Group	The name of the group that contains the form, image, and/or field.	current group

Examples

This example centers the original address lines data in the image, 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	Skywire Software
line 2	Atlanta, GA 30339-4000
line 3	770.859.9900
	5,000 FAP units

line 1	Skywire Software
line 2	Atlanta, GA 30339-4000
line 3	770.859.9900
	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 266](#)
[Field Functions on page 67](#)

KickToWIP

Use this function to send a transaction to WIP from the GenData program. This function lets you use DAL instead of the KickToWIP field-level rule or the field properties Attributes required field flag.

Syntax **KickToWIP()**

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 image-level rules:

```
<Image Rules>
...
;PreImageDAL;;KickToWIP( )
```

In this example, the Manual batch flag is set if the image is triggered.

Here is an example of the field-level rules:

```
;0;0;areal;0;0;areal;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 64](#)

LEAPYEAR

Use this function to find out whether or not the specified year is a leap year. This function returns one (1) if the year is a leap year and zero (0) if it is not.

Syntax **LeapYear (Year)**

Parameter	Description	Defaults to...
Year	A valid year value - must be an integer.	the current year

This function accepts either a two- or four-digit number.

If a two-digit number is used, the current century is added to create the year value. 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 07/01/98.)

Function	Result	Explanation
LeapYear ()	1	The parameter defaults to the current year (1998). Since 1998 is a leap year, one (1), which represents true, is the result.
LeapYear (97)	0	The year 1997 was not a leap year. Therefore, the result is zero (0), representing false.
LeapYear (Year ("1999/09/ 09", "34"))	0	First the Year function extracts the year number (1999) from the date, which is given in the date format "34". Then LeapYear determines that 1999 is not a leap year and returns zero (0.)

See also [Using INI Options on page 8](#)
[Date Formats on page 59](#)
[Year on page 401](#)
[Date Functions on page 58](#)

LEFT

Use this function to return a specified number of left most characters.

Syntax Left (String, Length)

Parameter	Description	Defaults to...
String	A valid string	the value of current field text
Length	Length of output	the length of first parameter (string)

This function 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 339](#)

[String Functions on page 84](#)

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	Defaults to...
String	A valid string.	the value of current field text

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 an image 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 image and counts the length of the data. The data in this field is <i>Jane Doe</i> , so the number 8 is returned.

See also [String Functions on page 84](#)
[@ on page 101](#)
[Size on page 362](#)

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 (String_list, List_string)

Parameter	Description	Defaults to
String_list	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.	No default
List_string	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.	No default

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 <i>e_codes</i> contains: <i>ABC,A</i> .
ListInList(GetValue("e_codes"), "ABC,AB,DE,A,GFHI")	2	DAL variable, <i>e_codes</i> , contains: <i>AB,abcd</i> .
ListInList(?("e_codes"), "ABC,AB,DE,A,GFHI,ABCD")	3	XDB entry <i>e_codes</i> returns: <i>DE,a</i> .
ListInList(?("e_codes"), ?("t_codes"))	4	XDB entry <i>e_codes</i> returns <i>A</i> . The entry <i>t_codes</i> contains: <i>ABC,AB,DE,A,GFHI,ABCD</i> .
ListInList(?("e_codes"), "ABC,AB,DE,A,GFHI,ABCD")	0	XDB entry <i>e_codes</i> returns: <i>XYZ</i> .

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
<code>#rc = ListInList(GetValue(col_name1), col_name1_codes)</code>	3
<code>#rc = ListInList(GetValue(col_name1), GetValue(col_name1_codes))</code>	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 84](#)

LOADINIFILE

Use this procedure/function to load an INI file into cache memory.

Syntax **LoadINIFile (Context, File)**

Parameter	Description	Required?
Context	A name (valid string) that will be associated to the set of INI control groups and options contained in the physical file.	Optional
File	The name of the INI file to load. If you omit the extension, the system assumes <i>INI</i> . The system searches in the current directory, or uses a full path name if you specify one	Yes

This procedure loads into cache memory the specified INI file.

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.

This procedure returns success (1) if no error occurred during its execution, otherwise a failure (0) is returned.

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 <i>INI</i> .
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 <i>Run_process</i> .	The INI file is loaded into cache memory.

See also [INI Functions on page 77](#)
 [Using INI Options on page 8](#)
 [SaveINIFile on page 346](#)
 [GetINIBool on page 240](#)
 [GetINIString on page 242](#)
 [PutINIBool on page 325](#)
 [PutINIString on page 327](#)

LOADLIB

Use this procedure/function to load into cache memory a file which contains a library of DAL scripts.

Syntax LoadLib (File)

Parameter	Description	Defaults to...
File	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.	no default

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 80](#)
 [Creating a DAL Script Library on page 5](#)

LOGO

Use this procedure/function to place a logo at a specified position in the image.

Syntax **Logo (Logo, Xcoordinate, Ycoordinate, Image, Form, Group)**

Parameter	Description	Defaults to...
Logo	A valid name for a logo. Must be a variable field object.	no default
Xcoordinate	A valid X coordinate location.	no default
Ycoordinate	A valid Y coordinate location.	no default
Image	An image name that should contain the new logo.	the current image
Form	A form name containing the specified image.	the current form
Group	A group name containing the specified image or form.	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 logo 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 logo 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 image 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 logo on the current image 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 logo at the same location as the field, "MyField".

See also [ChangeLogo on page 149](#)

[DelLogo on page 201](#)

[HaveLogo on page 254](#)

[InlineLogo on page 262](#)

[FieldX on page 223](#)

[FieldY on page 224](#)

[Refresh on page 333](#)

[RenameLogo on page 335](#)

[Logo Functions on page 78](#)

LOWER

Use this function to convert all alphabetic characters to lowercase characters and return the result.

Syntax **Lower (String, Length)**

Parameter	Description	Defaults to...
String	A valid string.	the value of current field text
Length	Length of output.	the length of first parameter (string)

If the length specified 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 388](#)
[String Functions on page 84](#)

MAILWIP

Use this procedure/function to send the current work-in-process to another user via email.

Syntax **MailWIP (Address)**

Parameter	Description	Defaults to...
Address	A valid email address	the mail address window to allow selection of 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 0	string	Returns the system's major version number.

See also [Miscellaneous Functions on page 80](#)

[MinorVersion on page 288](#)

[DAL Script Examples on page 43](#)

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, Image, Form, Group)**

Parameter	Description	Defaults to...
PartialName	A valid string. The string must be the common (prefix) portion of a set of field names that occur on the current image.	the current field
Image	Name of an image that contains the field named.	the current image
Form	Name of a form that contains the image and/or field named.	the current form
Group	Name of the form group that contains the form, image, and/or field named.	the current form group

This function 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 an image. 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 images (IMG A and IMG B). Each image has fields of the same name as a field in the other image.

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

Field	Image	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 image 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 image. 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 image 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 image 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 an image 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 image. Eight fields meet the naming criteria, but only five fields actually have values.

See also [Field Functions on page 67](#)
[Field Formats on page 68](#)
[Locating Fields on page 70](#)

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, Image, Form, Group)**

Parameter	Description	Defaults to...
PartialName	A valid string. The string must be the common (prefix) portion of a set of field names.	the current field
Image	Name of an image that contains the field named.	the current image
Form	Name of a form that contains the image and/or field named.	the current form
Group	Name of the form group that contains the form, image, and/or field named.	the current form group

This function 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 an image. 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 images (IMG A and IMG B). Each image has fields of the same name as a field in the other image.

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

Field	Image	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 image 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 image. 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 image contains two fields that begin with the name <i>MyField</i> . 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 image 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 an image 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 image. 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 67](#)
[Field Formats on page 68](#)
[Locating Fields on page 70](#)

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
<code>vers = MajorVersion() & '.' & MinorVersion()</code>	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 80](#)
[MajorVersion on page 283](#)
[DAL Script Examples on page 43](#)

MINUTE

Use this function to extract the number of minutes from a time.

Syntax **Minute (Time, Format)**

Parameter	Description	Defaults to...
Time	A valid time string. Assumed to be in the format specified by the next parameter.	the current time
Format	A valid time format string. Describes the first parameter.	time format 1

Example Here are some examples:

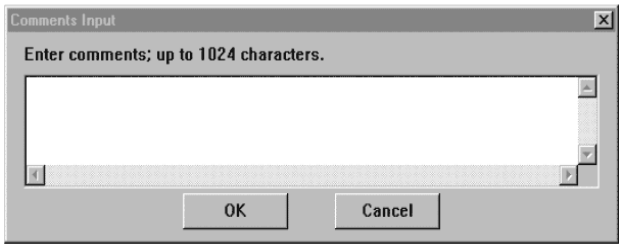
(Assume the current time is 03:05:09.)

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 86](#)

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	Defaults to...
Prompt	A text string to assign as the prompt for the field	No default
Title	A text string to assign as the title of the window.	No default
Length	Maximum input text length	1024
DefText	A text string to assign as the default input data.	No default

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: - <i>Now is the time</i> in blank line 1 - <i>gray area:</i> 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, <i>output</i> : 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, <i>output</i> , contains six lines, as shown in item 2. This example uses a multi-line variable field.

Function	Results	Explanation
<code>input_data = MLETranslate (MLEInput ("Enter comments; up to 1024 characters.", "Comments Input", , @"output"), "\n"); SetFld (input_data, "output1");</code>	<p>1. Window</p> <p>line 1 blank line line 3 line 4 blank line line 6</p> <p>2. Input_data</p> <p>line 1\nNow is the time\nline 3\nline 4 gray area\n\nline 6</p> <p>3. Multi-line variable <i>output1</i>:</p> <p>line 1 Now is the time line 3 line 4 gray area blank line line 6</p>	<p>(Assume this DAL script is executed after the example above.)</p> <p>The window contains the data under item 1 after the DAL script is executed.</p> <p>Assuming you entered:</p> <p>- <i>Now is the time</i> in blank line 1</p> <p>- <i>gray area</i> after the data 'line 4 '</p> <p>and then clicked Ok, the DAL internal variable, 'input_data', contains the string shown in item 2.</p> <p>The multi-line variable field, <i>output1</i>, contains the data shown in item 3.</p> <p>This example uses a multi- line variable field.</p>

See also [MLETranslate on page 293](#)
[Documaker Workstation Functions on page 65](#)

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. This function returns the translated data string for display, storage, or both.

Syntax **MLETranslate (String, ReplaceChar)**

Parameter	Description	Defaults to...
String	Text string returned from the MLEInput function.	No default
ReplaceChar	A text string to replace each set of <code>\n</code> characters in a returned MLEInput data string.	No default

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<br=""></input_data> (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 <i>Now is the time.</i> followed by the entry key. Plus added <i>gray area</i> 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<br=""></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<br=""></input_data> (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, <i>output</i> , would contain six lines as shown in item 2. This example uses a multi-line variable field

Function	Results	Explanation
<pre>input_data = MLETranslate (MLEInput ("Enter comments; up to 1024 characters.", "Comments Input", , @"(output)"), "\n"); SetFld (input_data, "output1");</pre>	<p>1. Window</p> <p>line 1</p> <p>blank line</p> <p>line 3</p> <p>line 4</p> <p>blank line</p> <p>line 6</p> <p>2. Input_data</p> <p>line 1 \nNow is the time. \nline 3 \nline 4 gray area \n\nline 6 \n</p> <p>3. Multi-line variable <i>output1</i></p> <p>line 1</p> <p>Now is the time.</p> <p>line 3</p> <p>line 4 gray area</p> <p>blank line</p> <p>line 6</p>	<p>(Assume this DAL script is executed after the example above.)</p> <p>After executing the script, the window contains the data shown in item 1.</p> <p>Assuming you entered:</p> <p><i>Now is the time.</i> for blank line 1 area, and added <i>gray area</i>, after the data 'line 4 ' and then clicked Ok.</p> <p>The DAL internal variable, 'input_data', would contain the data string shown in item 2.</p> <p>The multi-line variable field, 'output1', would contain the data shown in item 3.</p> <p>This example uses a multi- line variable field.</p>

See also [MLEInput on page 290](#)
[Documaker Workstation Functions on page 65](#)

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.

This function 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;FEATURE1550;
```

Here is an example:

```
BeginSub Feature1550
  #rec = CountRec("1,Feature1550,31,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 79](#)

MONTH

Use this function to determine the number of the month in a given date and return the number.

Syntax **Month (Date, Format)**

Parameter	Description	Defaults to...
Date	A valid date string. Assumed to be in the format specified by the next parameter.	the current date
Format	A valid date format string. Describes the first parameter.	the date format 1

This function 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 07/01/99.)

Function	Result	Explanation
Month ()	7	The parameter defaults to the current date.
Month ("99/138", "l")	5	The given date (99/138) in the date format l is the equivalent of May 18, 1999. 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, 1999 is returned to the target variable <i>datestring</i> . The Month function then returns the number of the month of October (10).

See also [Date Functions on page 58](#)

[Date Formats on page 59](#)

[DateAdd on page 171](#)

[MonthName on page 298](#)

MONTHNAME

Use this function to find the name of the month in a given date and return that name.

Syntax **MonthName (Month)**

Parameter	Description	Defaults to...
Month	A valid month value, from 1 to 12.	the current month

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 07/01/99.)

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 ("99/138", "I")))	May	First the Month function determines that the month number for the given date is 5. (99/138 is equivalent to May 18, 1999) Then MonthName returns the corresponding month name of May.

See also [Date Functions on page 58](#)
 [Month on page 297](#)

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	Defaults to...
Msgline1	Any string message	no default
Msgline2	Second line of message	no default
Msgline3	Third line of message string	no default
Title	Title of message window	No default

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 65](#)

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 the Rules Processing and Entry systems.

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 84](#)

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, Image, Form, Group)**

Parameter	Description	Defaults to...
Field	Name of an image field.	the current field name
Image	Name of an image that contains the field named.	the current image
Form	Name of a form that contains the image and/or field named.	the current form
Group	Name of the form group that contains the form, image, and/or field named.	the current form group

This function uses the parameters provided to search for one field on an image and return that field's data as a number. The field need not 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 image.
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 67](#)
[Field Formats on page 68](#)
[Locating Fields on page 70](#)
[@ on page 101](#)

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	Defaults to...
String	Any valid string within an image.	the current field value

This function 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 79](#)

NUMTEXT

Use this function to convert a numeric value into a series of descriptive words.

Syntax NumText (Number, Dollarword, Centword, Decimode)

Parameter	Description	Defaults to...
Number	any valid amount	the current field value
Dollarword	any word to use instead of dollars	“dollars and”
Centword	any word to use instead of cents	“cents”
Decimode	1 = numeric decimal amount 2 = spell decimal amount 3 = suppress zero, numeric decimal amount 4 = suppress zero, spell decimal amount	1

This function returns the written word equivalent of a numeric value. This function attempts to remove formatting information from the parameter number. If the value after deformating 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 (o) 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.
NumText(12.00, , 3)	Twelve dollars	A decimal mode of 3 suppresses the zero decimal.

Function	Result	Explanation
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 84](#)
[FrenchNumText on page 233](#)

PAD

Use this function to add trailing spaces or characters and return the result.

Syntax **PAD (String, Length, Char)**

Parameter	Description	Defaults to...
String	A valid string.	the value of current field text
Length	Desired length of output.	the length of first parameter (string)
Char	A valid string containing the pad characters you want to use.	a space character

This function 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 84](#)

PAGEIMAGE

Use this function to return the name of an image 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 images by that name. Once you have an image name, you can use other DAL functions to query the image, to insert a new image, or to delete the image.

Syntax **PageImage(Page, Recipient, Form, Group)**

Parameter	Description
Page	Include this parameter to indicate the specific page where you want to locate an image. If you omit this parameter, an image 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 images located by that recipient. If you omit this parameter, the name of the first image 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 image 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 image occurs more than once. For instance, if an image named, MYIMAGE, is located on the given page, but this is the second occurrence of the image within the named form, the name returned will be *MYIMAGE\2*.

See also [PageInfo on page 307](#)
[Name Functions on page 81](#)

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, Image, 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 an image of the page located. If you omit this parameter, the function matches the first page identified by the remaining search criteria.
Image	(optional) This parameter names an image 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 Image, 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 an image, but no form or group, the assumption is the image 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 an image and a group, but no form, the assumption is the image can occur on any form within that group.
- If you omit the image, 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 image on a page with the required recipient. If no recipient is specified, the first image 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 image on a page is a custom size, the width and height will reflect the *best* values.

Generally when an image 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 image size, then the larger of the values is returned.

Also remember since page size is determined by the first image 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 image, form, or group (or combination thereof) cannot be located within the specified document.

See also [PageInfo on page 307](#)
[Page Functions on page 82](#)

PAGINATEFORM

Use this function/procedure to apply image origins and re-paginate the form if necessary. During this re-pagination, the function will create or delete pages as needed.

NOTE: The AddImage and DellImage DAL functions include a parameter (Paginate) which you can use to force re-pagination after the affected image 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 <i>line of business</i> 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 image origin rules.

See also [AddImage on page 114](#)
 [DellImage on page 199](#)
 [Page Functions on page 82](#)

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.

This function 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 312](#)
[String Functions on page 84](#)

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 310](#)

[String Functions on page 84](#)

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 function 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 315](#)
[File/Path Functions on page 74](#)

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 function 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 314](#)

[File/Path Functions on page 74](#)

POW

Use this function to raise a number to an exponential power. This function returns a one (1) on success or zero (0) on failure.

Syntax **POW (Base, Exponent)**

Parameter	Description	Defaults to...
Base	The base number, positive or negative, to be raised to an exponential power.	1.00
Exponent	Exponent (power) to which the base number will be raised.	0

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 79](#)

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	Defaults to...
Text	Required. A string to be printed to the console.	no default

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 <i>Company</i> is printed to the console.

See also [DAL Script Examples on page 43](#)
 [Miscellaneous Functions on page 80](#)

PRINTERCLASS

Use this function to find out the type of print stream the system is generating.

Syntax **PrinterClass ()**

Example Here are some examples. Assume the following INI options exist.

```
< Printer >
  PrtType = AFP
< PrtType:AFP >
  PrintViewOnly = Yes
  OnDemandScript = OnDemand
```

Function	Result	Explanation
type = PrinterClass ()	a string	The DAL target variable, <i>type</i> will contain <i>AFP</i> .
If (PrinterClass() = 'PrtType:AFP') Then AddComment(AppldxRec()) End	a string	If the print type is <i>AFP</i> then execute following statement.

See also [AddComment on page 109](#)
 [DAL Script Examples on page 43](#)
 [Printer/Recipient Functions on page 83](#)

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 ()**

Example Here are some examples. Assume following 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 242](#)
 [DAL Script Examples on page 43](#)
 [Printer/Recipient Functions on page 83](#)

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/Recipient Functions on page 83](#)

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/Recipient Functions on page 83](#)

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 existing attribute on the named form.

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	Defaults to...
Name	The name of the form attribute (metadata).	No default
Data	The value associated with the form attribute (metadata).	Defaults to an empty string.
Form	Name of a form to retrieve data from.	Current form
Group	Name of the group that contains the specified form.	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 an image 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 an image 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 238](#)
[Miscellaneous Functions on page 80](#)

PUTINIBool

Use this procedure/function to store a Boolean value in an INI control group and option Boolean variable. This procedure returns one (1) if no error occurred during execution and zero (0) if there was an error.

Syntax PutINIBool (Context, Group, Option, Default)

Parameter	Description	Required?
Context	A name (valid string) associated with a set of INI control groups and options loaded into cache memory.	Optional
Group	The group name (valid string) which contains the INI option Boolean variable.	Yes
Option	The option name (valid string) which contains the INI Boolean variable to be stored into. If the control group and option does not exist, one will be created.	Yes
Default	Default Boolean value to store into the control group and option Boolean variable. If a default value is not defined, then a zero (0) is stored.	Optional

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
<code>rc = PutINIBool (,"control", "LogEnabled");</code>	The variable <i>bool_value</i> in the FSIUSER.INI file now contains a zero (0). The return code <i>rc</i> 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.
<code>rc = PutINIBool ("MVF", "control", "LogEnabled");</code>	The variable <i>bool_value</i> in the TEST1.INI file now contains a zero (0). The return code <i>rc</i> is set to one (1).	The procedure scans only the control group and option set associated with the context name <i>MVF</i> .
<code>rc = PutINIBool ("MVF", "control", "LogEnabled", 1);</code>	The variable <i>bool_value</i> in the TEST1.INI file now contains a one (1). If <i>Control</i> and <i>LogEnabled</i> are not found, the system creates a control group and option and sets the Boolean variable <i>LogEnabled</i> to one (1).	The procedure scans only the control group and option set associated with the context name <i>MVF</i> .

See also [INI Functions on page 77](#)
[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. This procedure returns one (1) if no error occurred during execution and zero (0) if there was an error.

Syntax PutINIString (Context, Group, Option, Default)

Parameter	Description	Required?
Context	A name (valid string) associated with a set of INI control groups and options loaded into cache memory.	Optional
Group	The group name (valid string) which contains the INI option string variable.	Yes
Option	The option name (valid string) which contains the INI string variable to be stored into. If the control group and option does not exist, one will be created.	Yes
Default	Default string value to store in the control group and option string variable. If a default value is not defined, a <i>Null</i> is stored.	Optional

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 FSI SYS.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
<code>rc = PutINIString ("Control", "Title");</code>	The string variable <i>Title</i> in the FSIUSER.INI file now contains <i>Bob's string</i> . The return code <i>rc</i> 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.
<code>rc = PutINIString ("MVF", "Control", "Title");</code>	The string variable <i>Title</i> in the TEST1.INI file now contains <i>MVF's string</i> . The return code <i>rc</i> will be set to one (1).	The procedure scans only the control group and option set associated with the context name <i>MVF</i> .
<code>rc = PutINIString ("MVF", "Control", "Title", "New string");</code>	The string variable <i>Title</i> in the TEST1.INI file now contains <i>New string</i> . If <i>Control</i> and <i>Title</i> are not found, the system creates them and sets the string variable <i>Title</i> to <i>New string</i> .	The procedure scans only the control group and option set associated with the context name <i>MVF</i> .

Example [INI Functions on page 77](#)
 [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
<code>rb = RecipBatch();</code>	Batch1	Returns the name of the recipient batch being processed.

See also [RecipCopyCount on page 330](#)
[RecipName on page 332](#)
[Printer/Recipient Functions on page 83](#)

RECIPCOPYCOUNT

Use this function to count the number of recipient copies for specified images and return that number.

Syntax **RecipCopyCount(Recip,Image,Form,Group)**

Parameter	Description
Recip	(optional) Enter the names of the recipients you want included in the count.
Image	Enter the names of the images 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, Image, 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 329](#)
 [RecipName on page 332](#)
 [Printer/Recipient Functions on page 83](#)

RECIPIENTNAME

Use this function to return from the FORM.DAT file the recipient name related to the specified image, 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, Image, Form, Group)

Parameter	Description	Defaults to...
Count	An indexed reference to locate a recipient in the FORM.DAT file.	the first recipient in the FORM.DAT file
Image	Name of an image that contains the recipient.	the current image
Form	Name of a form that contains the recipient.	the current form
Group	Name of the form group that contains the recipient.	the current group

If you omit the parameters, the system uses the first recipient it finds in the FORM.DAT file for the image, form, or group.

If the image, 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 image, form, or group, an empty string is returned.

See also [HaveRecip on page 255](#)
[Name Functions on page 81](#)

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
<code>rb = RecipName();</code>	Insured	Returns the name of the recipient batch being processed.

See also [RecipCopyCount on page 330](#)
[RecipBatch on page 329](#)
[Printer/Recipient Functions on page 83](#)

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 logo (bitmap).

NOTE: This procedure is valid only in Documaker Workstation scripts.

Example Here is an example:

Procedure	Result	Explanation
Refresh ()	Repaints the screen.	New logos or text now appears.

See also [Documaker Workstation Functions on page 65](#)

[AppendText on page 121](#)

[AppendTxm on page 123](#)

[DelLogo on page 201](#)

[Logo on page 279](#)

[ChangeLogo on page 149](#)

REMOVEATTACHVAR

Use this function to remove an attachment variable. You can use this function when creating print comments using Documaker Bridge.

Parameter	Description
Name	The name of the attachment variable.
DSI queue input or output	(optional) Enter one (1) for input or two (2) for output. The default is 1.
Returns	One (1) if the variable was found and zero (0) if it was not found.

See also [Docupresentment Functions on page 66](#)
[AddAttachVAR on page 106](#)
[GetAttachVAR on page 235](#)

RENAMELOGO

Use this procedure/function to rename a logo (bitmap). This procedure returns one (1) on success or zero (0) on failure.

Syntax **RenameLogo(LogoName,NewName,Image,Form,Group)**

Parameter	Description	Defaults to...
LogoName	Name of the logo to be renamed. Logo names are assigned in the Image Editor.	no default
NewName	New name for the logo.	no default
Image	Name of the image that contains the specified logo.	the current image
Form	Name of the form that contains the image.	the current form
Group	Name of the group to use to locate the specified object.	the current group

This procedure renames the logo you specify. The Logo procedure, which adds a logo on the fly, names the new logo using the name you specify.

If you want a more generic name so you can address the logo again without knowing the file associated with it, use this procedure *after* you use the Logo procedure.

You must specify both the LogoName and NewName parameters.

Example Here are some examples:

Procedure	Result	Explanation
RenameLogo("Log1", "Jane Doe")	1 or 0	Renames Log1 (on the current image, 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 image, IMH1, on the form, UpRate) to Jane Doe.

See also [ChangeLogo on page 149](#)
[DelLogo on page 201](#)
[HaveLogo on page 254](#)
[InlineLogo on page 262](#)
[Logo on page 279](#)
[Name Functions on page 81](#)

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, Image, Form, Group)**

Parameter	Description	Defaults to...
Field	Enter the name of the field you want to reset. Enclose the field name in quotation marks. Here is an example: "FIELD01"	No default, entry required
Image	Enter the name of the image that contains the field name.	The current image
Form	Enter the name of a form that contains the image or field name or both.	The current form
Group	Enter the name of the form group that contains the form, image, and/or field name.	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 image, BOAT PREM.

RESETOVFLWSYM

Use this procedure/function to reset the value in an overflow symbol to zero. Optionally, this procedure returns one (1) on success or zero (o) on failure.

Syntax **ResetOvFlwSym (Form, Symbol)**

Parameter	Description	Required?
Form	The name of the form that contains the fields on which overflow processing will occur.	Yes
Symbol	The name you want to use as the overflow symbol.	Yes

This procedure stores a zero (o) 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 (o) on failure.

Syntax [AddOvFlwSym on page 119](#)
 [GetOvFlwSym on page 244](#)
 [IncOvFlwSym on page 261](#)
 [Documaker Server Functions on page 64](#)

RETAIN

Use this procedure to identify DAL variables that should not be cleared between the processing of transactions.

Syntax Retain (Variable)

Parameter	Description	Defaults to...
Variable	Names of the DAL variables (as a quoted string) you want to retain during the processing of transactions.	No default.

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 80](#)

RIGHT

Use this function to return a specified number of right most characters.

Syntax **Right (String, Integer)**

Parameter	Description	Defaults to...
String	A valid string.	the value of current field text
Integer	Desired length of output.	the length of first 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 84](#)
[Left on page 273](#)

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.

Understanding the System

Documaker requires that all fields on an image be uniquely named. Image Editor forces 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 (FieldName)

Parameter	Description
FieldName	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")
```

Returns *Street address*.

```
MYFIELDNAME = "Comment #003"  
RootName(MYFIELDNAME)
```

Returns *Comment*.

```
RootName(FieldName())
```

Returns the root name of the current field.

See also [Name Functions on page 81](#)

ROUND

Use this function to round a number to the nearest specified decimal point and return the result.

Syntax **Round (Number, Places)**

Parameter	Description	Defaults to...
Number	A valid numeric value with decimals.	the current field value
Places	Number of requested decimal places.	2

This function 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 an image 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 84](#)

ROUTEWIP

Use this procedure/function to send all the work contained in WIP to all the recipients specified in a routing slip. Optionally, this procedure returns one (1) on success or zero (0) on failure.

Syntax **RouteWIP (Slip)**

Parameter	Description	Defaults to...
Slip	The name of a routing slip.	a window that lets the user select a routing slip

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, <i>manager</i> . 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](#)

RPErrMsg

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 **RPErrMsg (Message)**

Parameter	Description	Defaults to...
Message	A message can consist of static text, DAL functions, and procedures.	Optional, defaults to a blank string.

Example Here is an example:

```
RPErrMsg ( )
```

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:

```
RPErrMsg ("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:

```
RPErrMsg (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 344](#)

[RPWarningMsg on page 345](#)

[Documaker Server Functions on page 64](#)

RPLoGMsg

Use this procedure to write a message into Documaker Server's log file (LOGFILE.DAT).

Syntax **RPLoGMsg (Message)**

Parameter	Description	Defaults to...
Message	The message can consist of static text, DAL functions, and DAL procedures.	Optional, defaults to 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 [RPErrMsg on page 343](#)
[RPWarningMsg on page 345](#)
[Documaker Server Functions on page 64](#)

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	Defaults to...
Message	The message can consist of static text, DAL functions, and procedures.	Optional, defaults to 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 [RPErrMsg on page 343](#)

[RPLogMsg on page 344](#)

[Documaker Server Functions on page 64](#)

SAVEINIFILE

Use this procedure/function to save a set of INI control groups and options that were loaded into cache memory. This procedure returns one (1) if no error occurred during execution and zero (0) if there was an error.

Syntax **SaveINIFile (Context, File)**

Parameter	Description	Defaults to...
Context	(optional) A name (valid string) associated with a set of INI control groups and options loaded into cache memory.	no default
File	Name of the file in which you want to store the specified set of INI control groups and options. Assumes a file extension, if not present, of <i>.INI</i> . Stores in the current directory or uses a full path name if specified.	no default

This procedure saves into a physical file the set of specified INI control groups and options. 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 <i>.INI</i> .
SaveINIFile ("Run_process" , "DALRun.ini");	The set INI control groups and options referenced by the context name, <i>Run_process</i> , are saved in a file.	The INI control groups and options are saved to the specified file.

See also [INI Functions on page 77](#)
[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	Defaults to...
Time	A valid time string. Assumed to be in the format specified by the next parameter.	the current time
Format	A valid time format string. Describes the first parameter (time1).	time format 1

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 Formats on page 86](#)

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. This function is used when splitting recipient batches into multiple print stream files.

Syntax SetDeviceName(Device)

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/Recipient Functions on page 83](#)
[BreakBatch on page 146](#)
[DeviceName on page 203](#)
[UniqueString on page 389](#)

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 images, 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, Image, Form, Group)

Parameter	Description	Defaults to...
Field	Name of a field.	the current field
Count	A positive or negative number used to move beyond the field specified.	zero (0)
Image	Name of an image that contains the field named.	the current image
Form	Name of a form that contains the image and/or field named.	the current form
Group	Name of the form group that contains the form, image, and/or field named.	the current form 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 image's edit list, not necessarily its physical position on the image. 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 image has three fields named FIRST, SECOND, and THIRD. The fields occur in that order.

Procedure	Result	Explanation
SetEdit("THIRD")	1	Locates the field named <i>THIRD</i> on the current image. If found and if editable, that field will be the next field to receive focus.
SetEdit("THIRD".-2)	1	Locates the field named <i>THIRD</i> on the current image. If found, moves two fields prior to <i>THIRD</i> --to the field named <i>FIRST</i> .
SetEdit("MyField",,, "FRM")	1 or 0	Locates the form named <i>FRM</i> in the current form group. Then locates <i>MyField</i> on that form. If found, focus changes to that form and field.

See also [Documaker Workstation Functions on page 65](#)
[Field Formats on page 68](#)
[Locating Fields on page 70](#)

SETFLD

Use this procedure/function to assign a value to an image 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, Image, Form, Group)

Parameter	Description	Defaults to...
String	Any value appropriate for the field being assigned.	an empty string
Field	A field name.	the current field
Image	Name of an image that contains the field named.	the current image
Form	Name of a form that contains the image and/or field named.	the current form
Group	Name of the form group that contains the form, image, and/or field named.	the current form group

This procedure 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 DDT file for the images 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 image 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.
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 an image name was not given, the field may occur on any image on that form.

See also

[Field Functions on page 67](#)

[Field Formats on page 68](#)

[Locating Fields on page 70](#)

SetFont

Use this function to change the font on a field. For instance, you can use SetFont on non-multi-line text fields or barcode fields. You cannot use the SetFont function to reformat a text area.

Syntax **SetFont(FontID, Field, Image, Form, Group)**

Parameter	Description	Defaults to...
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.	no default
Field	(optional) A field name that identifies a multi-line text area. This is the field that receives the appended text.	the current field
Image	(optional) Name of an image that contains the field named.	the current image
Form	(optional) Name of a form that contains the image and/or field named.	the current form
Group	(optional) Name of the form group that contains the form, image, and/or field named.	the current form group

This function 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 67](#)

SETGVM

Use this procedure/function to update the contents of a GVM variable. You can also use this procedure to create a GVM variable. The system returns a one (1) if successful or a zero (0) if not.

Syntax **SetGVM (Name, Data, Instance, Type, Size)**

Parameter	Description	Defaults to...
Name	Required. The string which contains the name of the GVM variable	No default
Data	Required. The data you want to store in the GVM variable	No default
Instance	Instance number of the GVM variable	1
Type	Type of GVM variable to create: C = Character array S = Short L = Long F = Float D = Double Q = Long double	No default
Size	Number of bytes to reserve when creating a GVM variable. This parameter isn't used if the GVM already exists.	No default

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, <i>Company</i> , to the string <i>My Company</i> .
If (HaveGVM('My Variable') = 0) then; SetGVM('My Variable', 'My Data', 'C', 50) End	1 or 0	If the GVM variable, <i>My Variable</i> , does not exist; then create one that is a character array with a size of 50 plus store <i>My Data</i> in it.

See also [HaveGVM on page 252](#)
 [DAL Script Examples on page 43](#)
 [Documaker Server Functions on page 64](#)

SETIMAGEPOS

Use this procedure/function to reposition an image on a page.

Syntax **SetImagePos (PrefixName, Image, Form, Group)**

Parameter	Description	Required?
PrefixName	A prefix name to be associated with the coordinates returned by the procedure.	Yes
Image	The name of an existing image in the form set.	Defaults to current image.
Form	Name of a form within the form set that contains the image.	Defaults to current form.
Group	Name of the form group that contains the form and image.	Defaults to current form group.

This procedure repositions an image at the coordinates you specify in the PrefixName parameter: *prefix name.top*, *prefix name.left*.

NOTE: The image remains the same size.

This procedure retrieves these variables and sets the image'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 image 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 image height and width, which are not changed by this DAL function.

Procedure	Result	Explanation
SetImagePos ("MyImage")	New coordinates for the current image, <i>Image25</i> , will be: <pre>Myimage.top = 125 Myimage.left = 150 Myimage.bottom = 200 Myimage.right = 200</pre>	Sets the coordinates for the current image to the internal DAL variables: <i>Myimage.top</i> , <i>Myimage.left</i> , <i>Myimage.bottom</i> , and <i>Myimage.right</i> .
SetImagePos ("MyImage", "Image50")	New coordinates for the image, <i>Image50</i> , will be: <pre>Myimage.top = 125 Myimage.left = 150 Myimage.bottom = 200 Myimage.right = 200</pre>	Sets the coordinates for the image, <i>Image50</i> , to the internal DAL variables: <i>Myimage.top</i> , <i>Myimage.left</i> , <i>Myimage.bottom</i> , and <i>Myimage.right</i> .
SetImagePos ("m", "MVF\2", "XYZ")	The image is reposition to: <pre>m.top = top m.left = left m.bottom = coordinate m.right = right</pre>	The second occurrence of the image 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 image 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 [Image Functions on page 76](#)
[ImageRect on page 259](#)

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. This procedure optionally returns one (1) on success or zero (0) on failure.

Syntax **SetProtect (Mode, Field, Image, Form, Group)**

Parameter	Description	Defaults to...
Mode	An integer. Any non-zero value specifies field protection. Zero (0) means the field should be unprotected.	1, which means to protect the field
Field	Name of a field.	the current field
Image	Name of an image that contains the field named.	the current image
Form	Name of a form that contains the image and/or field named.	the current form
Group	Name of the form group that contains the form, image, and/or field named.	the current form group

This procedure returns zero (0) if the named field could not be changed or does not exist on the image. This procedure returns one (1) if the field was successfully protected.

Example Here are some examples:

(Assume the image 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, <i>Second</i> 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 <i>Second</i> based on the same criteria.

See also [Field Functions on page 67](#)

[Field Formats on page 68](#)

[Locating Fields on page 70](#)

SETRECIP

Use this procedure/function to assign the recipient copy count for a particular image, form, or group. Optionally, this procedure returns one (1) on success or zero (0) on failure.

Syntax **SetRecip (Recipient, Count, Image, Form, Group)**

Parameter	Description	Defaults to...
Recipient	A valid recipient name for the images being changed.	no default, required entry
Count	A positive number, zero (0) or greater, that represents the total copies this recipient should receive of the designated images.	zero (0)
Image	An image name to be located.	the current image
Form	A form name containing the specified image.	the current form
Group	A group name containing the specified image or form.	the current group

This procedure lets you specify how many copies of an image should print for a designated recipient. Setting the copy count to zero (0) for a recipient means the image will not print for that recipient.

Unlike other procedures, this one does not strictly apply the hierarchy rules for image, form, and group. In other words, you can specify a form without naming an image and the procedure will assign the copy count to all images on that form designated for that recipient. Likewise, if you only specify the group parameter, without image or form, all the images in that group will receive the new copy count for the designated recipient.

NOTE: This procedure cannot add a new recipient to an image. Images are predefined for specific recipients. This procedure can only change the copy count of known recipients for any particular image.

Example Here are some examples:

Procedure	Result	Explanation
SetRecip ("Insured", 2)	1 or 0	Defaults to the current image. If this image 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 image that specifies HOME OFFICE as a recipient the new copy count of one (1).

See also [Image Functions on page 76](#)

SETREQUIREDFLD

Use this function to change the required option of a field to Required or Not Required.

Syntax **SetRequiredFld (Required, Field, Image, Form, Group)**

Parameter	Description
-----------	-------------

Required	Enter Yes if you want to make the field required. Enter No if you want to make the field not required.
Field	(optional) Enter the name of the field. If you omit this parameter, they system uses the current field.
Image	(optional) Enter the name of the image. If you omit this parameter, they system uses the current image.
Form	(optional) Enter the name of the form. If you omit this parameter, they system uses the current form.
Group	(optional) Enter the name of the group. If you omit this parameter, they system uses 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 Image parameter, but omit the field parameter, the system uses the first field on that image. If you omit the Image and Field parameters, but include the Form, the system looks for the first field on the first image of the form you specified, and so on.

See also [Field Functions on page 67](#)
 [Field Formats on page 68](#)

SETWIPFLD

Use this procedure/function to set WIP fields from DAL to the record in memory.

Syntax **SetWIPFld (Field, Data);**

Parameter	Description	Defaults to...
Field	Variable field name.	no default, required entry
Data	The data you want stored in the field.	no default

NOTE: You cannot change the FormsetID field which is used to associate WIP records with data files.

This procedure 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, Image, Form, Group)**

Parameter	Description	Defaults to...
Field	Variable field name.	the current field
Image	Name of an image that contains the field named.	the current image
Form	Name of a form that contains the image and/or field named.	the current form
Group	Name of the form group that contains the form, image, and/or field named.	the current form group

This function 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 image. 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 68](#)

[Locating Fields on page 70](#)

[LEN on page 274](#)

[Field Functions on page 67](#)

SLIPAPPEND

Use this procedure/function to add an email address to the end of the routing slip associated with the form set. Optionally, this procedure returns one (1) on success or zero (0) on failure.

Syntax **SlipAppend (Address, Mode)**

Parameter	Description	Defaults to...
Address	An email address	no default
Mode	0 = linear recipient 1 = CC recipient	0 -- linear recipient

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 will be appended to the end of the current routing slip. Defaults to 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. Optionally, this procedure returns one (1) on success or zero (0) on failure.

Syntax **SlipInsert (Address, Mode)**

Parameter	Description	Defaults to...
Address	An email address	no default
Mode	0 = linear recipient 1 = CC recipient	0 -- linear recipient

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 will be 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. Defaults to 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 image 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 image (FAP or compiled FAP) if it is not already loaded.

Syntax

SpanField(SpanField, LeftField, RightField, Image, Form, Group)

Parameter	Description	Defaults to...
SpanField,	Specifies the filler character 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 be placed against the right-most field.	a period (.)
LeftField	Enter the name of the field on the left of the area you want to span.	no default
RightField	Enter the name of the field to the right of the area you want to span.	no default
Image	(optional) Name of an image that contains the fields you named.	the current image
Form	(optional) Name of a form that contains the image and/or field you named.	the current form
Group	(optional) Name of the form group that contains the form, image, or fields.	the current group

The SpanField parameter is always the first parameter, but you can specify the LeftField and RightField parameters in any order. The function 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 image designer is responsible for vertical alignment.

See also

[Field Functions on page 67](#)

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(Search criteria, Offset, Length, Occurrence)

Parameter	Description	Defaults to...
Search criteria	Defines the criteria you want used to look for the data in the extract file.	No default
Offset	For XML extract files, enter the offset into the data were the desired data starts. For flat files, enter the offset into the record were the data starts.	Zero
Length	Enter the number of characters to return.	Zero
Occurrence	This lets you specify which occurrence of the data to return. Entering one (1) or zero (0) returns the first occurrence of the data. This parameter is not valid for XML extract files.	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,HEADERREC* 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 236](#)
[Documaker Server Functions on page 64](#)

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, Image, Form, Group)

Parameter	Description	Defaults to...
Field	Name of an image field.	the current field name
Image	Name of an image that contains the field named.	the current image
Form	Name of a form that contains the image and/or field named.	the current form
Group	Name of the form group that contains the form, image, and/or field named.	the current form group

This function will use the parameters provided to search for one field on an image and return that fields data value as formatted. The field may 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 image.
STR("MyField\2", , "MyForm")	automobile	The second occurrence of MyField already contained a string value.

See also

[STRCompare on page 370](#)

[Field Functions on page 67](#)

[Field Formats on page 68](#)

[Locating Fields on page 70](#)

[@ on page 101](#)

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

#RTN = STRCompare (String1 , String2 , #count)

Parameter	Description	Defaults to...
String1	The text for the first string you want to compare	an empty string
String2	The text for the second string you want to compare	an empty string
#Count	Optional. 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.	-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 369](#)
[Field Functions on page 67](#)

SUB

Use this function to return a substring from a string at a specified position.

Syntax **SUB (String, Position, Length)**

Parameter	Description	Defaults to...
String	A valid string.	the value of current field text
Position	Position where sub should begin.	1
Length	Length to retrieve from the text.	the length of parameter 1 that remains, beginning at parameter 2

This function 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 84](#)

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, Image, Form, Group)

Parameter	Description	Defaults to...
PartialName	A valid string. The string must be the common (prefix) portion of a set of field names that occur on the current image.	the current field
Image	Name of an image that contains the field named.	the current image
Form	Name of a form that contains the image and/or field named.	the current form
Group	Name of the form group that contains the form, image, and/or field named.	the current form group

This function 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 an image. 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 images (IMG A and IMG B). Each image has fields of the same name as a field in the other image.

Field	Image	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
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 image 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 image. 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 image 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 image 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 an image 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 image. 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 79](#)
 [Field Formats on page 68](#)
 [Locating Fields on page 70](#)

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 Server System Reference](#).

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/Recipient Functions on page 83](#)
[DelBlankPages on page 195](#)
[AddBlankPages on page 107](#)

TABLE

Use this procedure/function to look up and return a value from a standard table.

Syntax **Table (RetCode, Key, Table, File)**

Parameter	Description	Defaults to...
RetCode	A return code value designated by the letters <i>K</i> and <i>D</i> . K = key code D = code description K + D = key code and code description D + K = code description and key code	the value of current field table return value
Key	Table key code.	the value of current field text
Table	Name of the table you want to search.	the value of current field table name—case sensitive
File	Name of the file that contains the table from the previous parameter.	the value of current field table file name, or the current image table file name—not case sensitive

This utility 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 the 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 <i>table1</i> . Then returns the description (Georgia) for the key code <i>GA</i> .

See also [Documaker Workstation Functions on page 65](#)

TIME

Use this function to build a time from a given time, or the current time.

Syntax **Time (Format, Hour, Minutes, Seconds)**

Parameter	Description	Defaults to...
Format	A valid time format string.	time format 1
Hour	A valid hour value - must be an integer.	the current time
Minutes	A valid minute value - must be an integer.	the current time
Seconds	A valid seconds value - must be an integer.	the current time

This function returns a time string that contains a formatted time value. If any values (hour, minutes, seconds) are omitted the appropriate value from the current time is substituted.

Example Here are some examples:

(Assume the current time is 07:07:32 am.)

Function	Result	Explanation
Time()	07:07:32	No parameters entered—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 86](#)

TIME2TIME

Use this function to convert a time from one format to another. Changes the existing time format into a new format.

Syntax **Time2Time (OldTime, OldFormat, NewFormat)**

Parameter	Description	Defaults to...
OldTime	A valid time string. Assumed to be in the format specified by the next parameter.	the current time
OldFormat	A valid time format string. Describes the first parameter (time1).	time format 1
NewFormat	A valid time string. Assumed to be in the format specified by the next parameter.	time format 1

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 86](#)

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	Defaults to...
Time	A valid time string. Assumed to be in the format specified by the next parameter.	the current time
Format	A valid time format string. Describes the first parameter (time1).	time format 1
Seconds	The number of seconds to be added.	0
Minutes	The number of minutes to be added.	0
Hours	The number of hours to be added.	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 86](#)

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 image for a recipient. A duplex sheet with front and back images counts as two pages.

Syntax **TotalPages(Recipient)**

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. An image 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 function 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 383](#)
[Documaker Workstation Functions on page 65](#)

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 images 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)

If you include the Recipient parameter, the count only reflects the sheets that print for that recipient. If you omit the Recipient parameter, then the count reflects all recipients.

The count takes into consideration recipient copy counts and duplex options. An image 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 function 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 382](#)

[Documaker Workstation Functions on page 65](#)

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 385](#)
 [TriggerRecsPerOvFlw on page 386](#)
 [Documaker Server Functions on page 64](#)

TRIGGERIMAGENAME

If you are using DAL scripts during Documaker Server SetRecip trigger processing, use this function to return the image (FAP file) name of the current SetRecipTb entry being processed.

Syntax **TriggerImageName()**

There are no parameters for the function.

Example Here is an example that uses this function.

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 image name are the specified
*** conditions then trigger the image.
```

```
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 384](#)
 [TriggerRecsPerOvFlw on page 386](#)
 [Documaker Server Functions on page 64](#)

TRIGGERRECSPEROVFLW

Use this function to retrieve the number of records per overflow image 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 image.

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_fl1550;;;Customer(1);;1,M;25;0;1;;DALTrigger;FEATURE1550;
```

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 image value of 25, which is used in the MOD function.

See also

[MOD on page 296](#)

[TriggerFormName on page 384](#)

[TriggerImageName on page 385](#)

[Documaker Server Functions on page 64](#)

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	Defaults to...
String	A valid string.	the value of current field text
Integer	A valid integer. Valid values are: 0 = remove trailing spaces 1 = remove leading spaces 2 = remove leading and trailing spaces	2 -- remove spaces from both ends

This function 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 84](#)

UPPER

Use this function to convert all characters to uppercase and return the result.

Syntax **Upper (String, Length)**

Parameter	Description	Defaults to...
String	A valid string	the value of current field text
Length	Length of output	the length of 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 84](#)
[Lower on page 281](#)

UNIQUESTRING

Use this function to return a 45-character globally unique string.

Syntax UniqueString()

Here is an example:

```
DataPath = GetINIString(, "Data", "DataPath")
Drive = FileDrive(DataPath)
Path = FilePath(DataPath)
UniqueID = UniqueString()
Outputname = FullFileName(Drive, Path, UniqueID, ".PDF")
SetDeviceName(Outputname)
```

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-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. DeviceName and UniqueString are applicable to both Entry and Documaker Server.

See also [Miscellaneous Functions on page 80](#)

[BreakBatch on page 146](#)

[DeviceName on page 203](#)

[SetDeviceName on page 349](#)

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
<code>result = UserID()</code>	TOMJ	Identifies the current user ID as TOMJ.
<code>SetFld (UserID (), "MyField")</code>	TOMJ	First UserID determines that the current user ID is TOMJ, then the field named <i>MyField</i> is assigned the value TOMJ by the SetFld procedure.

See also [WIP Functions on page 88](#)
[UserLvl on page 391](#)

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 ()**

Parameter	Description	Defaults to...
None	No parameters are specified for this function	No default

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 390](#)

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)**

Parameter	Description	Defaults to...
Date	A valid date string. Assumed to be in the format specified by the next parameter.	the current date
Format	A valid date format. Describes the first parameter.	date format 1

This function 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, 1999.)

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, 1999. Then WeekDay returns 5, which corresponds to Thursday.

See also [Date Functions on page 58](#)
[Using INI Options on page 8](#)
[Date Formats on page 59](#)
[DateAdd on page 171](#)
[DayName on page 176](#)

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, Image, Form, Group)**

Parameter	Description	Defaults to...
Field	Name of the field.	the current field
Image	Name of the image.	the current image
Form	Name of an existing form.	the current form
Group	Name of a group to contain the specified form.	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, image, or form names.

Example Here is an example:

Function	Result	Explanation
form = WhatForm("Total Field\3", , ,"*");	The name of the form or o	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 394](#)
[WhatImage on page 395](#)
[Name Functions on page 81](#)

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, Image, Form, Group)**

Parameter	Description	Defaults to...
Field	Name of the field.	the current field
Image	Name of the image.	the current image
Form	Name of a form.	the current form
Group	Name of a group that contains the specified form.	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, image, or form names.

Example Here is an example:

Function	Result	Explanation
<code>group = WhatGroup(, , "MyForm", "*");</code>	The name of the form or o	Attempts to locate the group name that contains a specific form.

See also [WhatForm on page 393](#)
 [WhatImage on page 395](#)
 [Name Functions on page 81](#)

WHATIMAGE

Use this function to return the name of the image 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, Image, Form, Group)**

Parameter	Description	Defaults to...
Field	Name of the field.	the current field
Image	Name of the image.	the current image
Form	Name of a form.	the current form
Group	Name of a group that contains the specified form.	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, image, or form names.

Example Here is an example:

Function	Result	Explanation
image = WhatImage("Total Field\12", , ,"*");	The name of the form or o	Attempts to locate the twelfth occurrence of a field in a form set and returns the name of the image that contains that field.

See also [WhatForm on page 393](#)
[WhatGroup on page 394](#)
[Name Functions on page 81](#)

WIPEXIT

Use this procedure/function to close work-in-process.

Syntax **WIPExit (SaveFlag)**

Parameter	Description	Defaults to...
SaveFlag	A valid save flag, which is any positive number. A zero exits WIP and does not save your work.	Yes

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 work.	Work is saved with a valid positive flag.
WIPExit(0)	Exits WIP but does not save 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	Defaults to...
WIPfield	A string that represents a field name within a WIP record.	Yes

This function 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, an empty string is returned.

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 <i>W</i> 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 ()**

Parameter	Description	Defaults to...
None	No parameters are specified for this function.	no default

This function 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 *Skywire*.)

Function	Result	Explanation
result = WIPKey1()	Skywire	Determines the value contained in the WIP Key1 field and returns that value.
IF WIPKey1 () = "Skywire" SetFld("N/A"); END	1 N/A	Determines that the Key1 field contains the value <i>Skywire</i> , then executes the SetFld procedure and places <i>N/A</i> 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 397](#)
[WIPKey2 on page 399](#)
[WIPKeyID on page 400](#)
[SetFld on page 352](#)

WIPKEY2

Use this function to return the value of the Key2 field from the current WIP record.

Syntax

WIPKey2 ()

Parameter	Description	Defaults to...
None	No parameters are specified for this function.	no default

This function returns the Description of the Key2 field within 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 Insuranc e	Determines the value contained in the WIP Key2 field and returns that value.
IF WIPKey2 () = "Skywire" SetFld("N/A"); END	Nothing	Determines that the Key2 field does not contain the value "Skywire"; therefore the SetFld procedure does not execute.

See also

[WIP Functions on page 88](#)

[WIPFld on page 397](#)

[WIPKey1 on page 398](#)

[WIPKeyID on page 400](#)

[SetFld on page 352](#)

WIPKEYID

Use this function to replace the value of the KeyID field from the current WIP record.

Syntax **WIPKeyID ()**

Parameter	Description	Defaults to...
None	No parameters are specified for this function.	no default

This function 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 397](#)
[WIPKey1 on page 398](#)
[WIPKey2 on page 399](#)
[SetFld on page 352](#)

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)**

Parameter	Description	Defaults to...
Date	A valid date string. Assumed to be in the format specified by the next parameter.	the current date
Format	A valid date format string. Describes the first parameter.	the date format 1

This function determines the year portion of the given date based on the specified format in parameter 2.

Example Here are some examples:

(Assume the current date is 07/01/99.)

Function	Result	Explanation
Year ()	1999	Defaults to the current date and returns a four-digit year.
Year ("2-5-99", "1-2")	1999	Returns a four-digit year for the given date.

See also [Date Functions on page 58](#)
[Using INI Options on page 8](#)
[Date Formats on page 59](#)
[YearDay on page 402](#)

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)

Parameter	Description	Defaults to...
Date	A valid date string. Assumed to be in the format specified by the next parameter.	the current date
Format	A valid date format string which describes the first parameter.	date format 1

This function determines the day of the year portion of the given date based on the specified format in parameter 2.

Example Here are some examples:
(Assume the current date is 07/01/99.)

Function	Result	Explanation
YearDay ()	182	Defaults to the current date and returns the day of the year (counting consecutively from January 1).
YearDay ("7-1-92")	183	Returns the day of the year (counting consecutively from January 1) for the given date. (Since 1992 was a leap year the number is one greater.)

See also [Date Functions on page 58](#)
 [Date Formats on page 59](#)
 [Year on page 401](#)

Index

SYMBOLS

" (quotation marks) 24, 25
(octothorpes) 12
\$ (dollar signs) 12
% (percent signs) 12
& (ampersands) 186, 267, 268, 353
() (parentheses) 24, 26
* (asterisks) 4, 70
: (colons) 6
; (semicolons) 3, 6
? function 103
@ function
 defined 101
 NUM function 301
\ (backslashes) 91
' (apostrophes) 25

A

ABS function 105
accessing
 database fields 56
ACIF 43
AddBlankPages function 107
AddComment function 109
AddDocuSaveComment function 110
AddForm function
 CopyForm function 160

AddForm_Propagate function 112
AddImage function 114
AddImage_Propagate function 117
AddOvFlwSym function 119
AFEBatchDALProcess 7, 347
AFELOG file
 AFELog function 120
 DelWIP function 202
AFELog function 120
AFEProcedures control group 20
alphabetic field format 68
alphanumeric field format 69
AND operation 135
annuities 316
apostrophes (') 25
AppendText function 121
AppendTxm function
 AppendTxmUnique function 126
 defined 123
AppendTxmUnique function 125
AppldxRec function 128
archives
 adding comments 109
 Complete function 154
 retrieving records 128
ASCII files
 AddDocuSaveComment function 110
 DAL script libraries 5
 scripts 4
Ask function 129
assignment statements 22
AssignWIP function 130
asterisks (*)
 comment lines 4
 wildcards 70

AutoKeyID
 option 20
 table 19
Avg function 131

B

backslashes in object names 91
BankRound function 133
banner processing
 RecipBatch function 329
 RecipName function 332
 SuppressBanner function 376
bar code field format 68
barcode fields
 SetFont function 354
batch processing
 DDTSourceName function 192
Batch_DAL control group 7
Beep function 134
BeginSub function
 EndSub function 37
binding 166
bit logical shift operation 142
BitAnd function 135
BitClear function 136
bitmaps
 ChangeLogo function 149
 DelLogo function 201
 embedding logos 214
 HaveLogo function 254
 InlineLogo function 262
 Refresh function 333
 RenameLogo function 335

- BitNot function 137
- BitOr function 138
- BitRotate function 139
- BitSet function 141
- BitShift function 142
- BitTest function 144
- bitwise
 - AND operation 135
 - exclusive OR operation 145
 - inclusive OR operation 138
 - logical NOT operation 137
 - shift operation 142
 - shift-and-rotate operation 139
- BitXor function 145
- blank lines
 - formatting scripts 3, 4
- blank pages 107, 195
- Boolean values
 - GetINIBool function 240
 - PutINIBool function 325
- Break statements
 - and While...Wend statements 34
- built-in functions 47

C

- cache
 - GetINIString function 242
 - LoadINIFile function 277
 - LoadLib function 278
 - SaveINIFile function 346
- Calculation tab
 - assigning a calculation 2
 - comments 4
- calculations
 - entering in an external file 4
 - POW function 316
- CALL statements
 - defined 33
- carbon-copy recipients 363, 364
- carriage returns 4
- case
 - dates 59
 - sensitivity 3, 4
 - target variables 22
- CaseSensitiveKeys option 19
- century
 - cut-off 61
 - DateCnv function 173
- CFind function 148
- CHAIN statements
 - defined 33
- ChangeLogo function 149
- Char function 151
- character strings
 - converting to an integer value 256
 - ListInList function 275
- CharV function 152
- CheckImageLoaded rule
 - SetFld function 352
- Class option 53
- clearing
 - BitClear function 136
- CLIPSPACES 187
- CodeInList function 153
- colons
 - use of 6
- commas
 - numeric constants 24
- comma-separated value files 310
- comment record processing
 - RecipBatch function 329
 - RecipName function 332
- comments
 - AddComment function 109
 - AddDocuSaveComment function 110
 - creating strings 300
 - lines 4

CompileWhenLoaded option 8
Complete function 154
CompressFlds function 155
compressing
 blank space 155
ConnectFlds function 157
Continue statements
 While...Wend statements 35
Control control group 8
control groups
 INI functions 77
control-z 4
converting
 character strings to integer values 256
 integer values to hexadecimal string values 193
coordinates
 ImageRect function 259
 SetImagePos function 356
copies
 counting recipient copies 330
 TotalPages function 382

CopyForm function 160
Count function 161
CountRec function 164
CreateIndex option 51
CreateTable option 51
custom field format 68
Cut function 165

D

DAL
 assignment statements 22
 calcs 2
 CALL statements 33
 CHAIN statements 33
 data flow statements 22
 DBUnloadDFD function 189
 debugger 39
 entering calculations in an external file 4
 examples 2, 48
 execution order 26
 flow control statements 22, 30
 format in external files 4
 GOTO statements 33
 IF statements 31
 image variable fields 24
 implicit conversion 28
 keywords 30
 labels 29
 numeric constants 24
 operators 25
 punctuation 26
 retaining variables 338
 RETURN statements 30
 runtime error messages 41
 runtime options 8
 source expression 23
 string constants 25
 target field 24
 using the Properties window 3
DAL control group 8
DAL rule
 ? function 103, 104

DAL scripts
 defined 2
 executing 320
 retrieving XML data 11

DALFunctions control group 8

DALLib option 9

DALLibraries control group 8

DALRun
 built-in function 10
 control group 9
 DBUnloadDFD function 189
 INI file 39

DALTriggers option 9

DALVAR built-in function 10

DashCode function 166

data flow statements 22

data storage statements 38

database functions
 accessing fields 56
 DB2/2 handler 52
 list of 50
 ODBC handler 51

databases
 handlers for Excel 53

date field format 68

Date function 169

Date2Date function 170

DateAdd function 171

DateCnv function 173

DateFmt rule
 ? function 104

DateFMT2To4Year option 61

DateFmt2To4Year option 8

dates
 century cut-off 61
 data storage statements 38
 formatting 59
 list of functions 58

Day function 175

DayName function
 defined 176
 WeekDay function 392

DaysInMonth function 177

DaysInYear function 178

DB2/2 handler 52

DBAdd function 179

DBCclose function
 defined 180
 memory tables 57
 record lengths 187

DBDelete function 181

DBFind function 182

DBFirstRec function 184

DBNextRec function 185

DBOpen function
 defined 186
 memory tables 57
 record lengths 187

DBPrepVars function 188

DBUnloadDFD function 189

DBUpdate function 190

DDT files
 DDTSourceName function 192
 FieldRule function 220
 storing information 6

DDTSourceName function 192

Debug_DAL_Rules option 9

Debug_Switches control group 9

Dec2Hex function 193

decimal target variables 23

DEFLIB directory 4

DeFormat function 194

DelBlankPages function 195

DelField function 196

DelForm function 198

Dellmage function 199

DelWIP function 202

descriptions
 retrieving 231

DestroyList function 12

DFD files
 DBUnloadDFD function 189

Dictionary Editor 103
DiffDate function 204
DiffDays function 205
DiffHours function 206
DiffMinutes function 207
DiffMonths function 208
DiffSeconds function 209
DiffTime function 210
DiffYears function 211
directing workflow 363, 364
disabling scripts 2
divide by zero 41
dividing year 173
Docusave
 AddDocusaveComment function 110
 adding comments 109
DocusaveScript option 43
dollar signs (\$) 12
dot operator 56
double quotes 25
drives
 FileDrive function 225
dummy pages 107
DumpDAL option 9
DupForm function
 CopyForm function 160
 defined 213

either required 221
email addresses 364
EmbedLogo function 214
EndSub function
 BeginSub function 36
ERRFILE.DAT file 343, 345
errors
 Beep function 134
 RPErrMsg function 343
 runtime error messages 41
Excel
 databases 53
exclusive OR operation 145
execution order 26
Exists function 215
exponential power
 using the POW function 316
exporting
 Complete function 154
Ext option
 defined 8
 LoadLib function 278
external files
 using 4
extract files
 CountRec function 164
 retrieving data 367
extracting a field's root name 340

E

EBCDIC
 AddDocusaveComment function 110

F

FAP units
 Logo function 279
 positioning images 357
field formats
 list of 68
 locating fields 70

FieldFormat function 216
FieldName function 217
FieldPrompt function 219
fields
 accessing database fields 56
 changing coordinates 157
 compressing blank space 155
 concatenating text 157
 ConnectFlds function 157
 date formats 59
 deleting 196
 extracting the root name 340
 functions 67
 JustField function 269
 locating 70
 moving horizontally 365
 renaming 125
 SpanField function 365
 specifying 24
 target fields 22
 using the Properties window 3
FieldType function 222
FieldX function
 defined 223
 Logo function 279
FieldY function
 defined 224
 Logo function 279
FileDrive function 225
FileExt function 226
FileName function 227
FilePath function 228
files
 DAL scripts 5
 entering calculations in an external file 4
 FileExt function 226
 FileName function 227
 FilePath function 228
 FullFileName function 234
filler pages 107, 195
Find function 229
flow control statements
 defined 22
 keywords 30
FlushDALSymbols option 9, 338
FlushSymbols option 8
fonts
 changing 354
form descriptions, retrieving 231
FORM PAGE NUM field 101
FORM PAGE NUM OF field 101
FORM.DAT file
 RecipientName function 331
 retrieving descriptions 231
Format function 230
formats
 DAL format in external files 4
 date 59
 numeric 69
 time 86
formatting functions
 fields 67
 string functions 84
FormDesc function 231
FormName function 232
forms
 AddForm_Propagate function 112
 CopyForm function 160
 DupForm function 213
 WhatForm function 393
FORMSET PAGE NUM field 101
FORMSET PAGE NUM OF field 101
FormsetID field
 SetWIPFld function 361
four-digit years 173
FrenchNumText function 233
FSISYS.INI file
 DAL script extensions 4
 executing DAL scripts 7
 GetINIBool function 240
 GetINIString function 242
 IgnoreInvalidImage option 111, 115
 INI functions 77
 LogEnabled option 240
 options for Docusave 43
 options for OnDemand 43
 PutINIBool function 325
 runtime options 8

FSIUSER.INI file
 executing DAL scripts 7
 GetINIBool function 240
 GetINIString function 242
 INI functions 77
 LogEnabled option 240
 options for Docusave 43
 options for OnDemand 43
 PutINIBool function 325
 PutINIString function 327

FullFileName function 234

functions

- mathematical 77, 79
- miscellaneous 80, 83, 88
- object 89
- overview 48
- string 84
- time 86
- where used 92

G

get field function 101

GetAttachVar function 235

GetData function 236
 and the SrchData function 236, 367

GetFormAttrib function 238

GetINIBool function
 defined 240

GetINIString function 242

GetListElem function 12

GetOvFlwSym function 244

GetValue function 245

GOTO statements
 defined 33
 runtime error messages 42
 While loops 36

GroupName function 246

groups

- PrinterGroup function 320
- WhatGroup function 394

GVM function

- defined 247

GVM variables

- HaveGVM function 252
- printing 318
- SetGVM function 355

H

HaveField function 248

HaveForm function 250

HaveGroup function 251

HaveGVM function 252

HaveImage function 253

HaveLogo function 254

HaveRecip function
 defined 255
 RecipientName function 331

Hex2Dec function 256

hexadecimal values

- date formats 61
- Date2Date function 170
- Dec2Hex function 193
- Hex2Dec function 256

host required 221

Hour function 257

I

IF rule 220, 221

IF statements
 defined 31
 runtime error messages 41

IgnoreInvalidImage option 111, 115

ImageName function 258

ImageRect function
 AddImage function 115
 defined 259

images

- adding 114
- checking 2
- PageImage function 306
- re-pagination 309
- repositioning 356
- retrieving coordinates 259
- variable fields 24
- WhatImage function 395

ImpFile_cd control group 111

implicit conversion 28

IncOvFlwSym function 261

INI files

- DAL options 8
- GetINIBool function 240
- GetINIString function 242
- LoadINIFile function 277
- PutINIBool function 325
- PutINIString function 327
- SaveINIFile function 346

INI functions 77

INIGroup control group 10

InlineLogo function 262

Input function 263

Insert function 264

inserting equipment 166

insertion text 69

Install option 51

INT function 265

integers

BitAnd function 135, 136

BiTest function 144

BitNot function 137

BitOr function 138

BitRotate function 139

BitSet function 141

BitShift function 142

BitXor function 145

Char function 151

CharV function 152

converting to hexadecimal string values 193

Dec2Hex function 193

Hex2Dec function 256

returning the remainder 296

target variables 23

interest rates

POW function 316

international

alphabetic field format 68

alphanumeric field format 68

uppercase alphabetic field format 68

uppercase alphanumeric field format 68

IsXMLError function 12

J

JCenter function 266

JLeft function 267

JRight function 268

JustField function 269

K

KeyID values 19

Keyword option 8

keywords 30

 BeginSub and EndSub 36

KickToWIP function 271

KickToWIP rule 221

L

labels

 in scripts 29

 runtime error messages 42

leading signs 302

leading spaces 305, 387

leap years

 DateAdd function 171

 DaysInMonth function 177

 DaysInYear function 178

 DiffYears function 211

 LeapYear function 272

 YearDay function 402

LeapYear function 272

Left function 273

LEN function

 defined 274

 Size function 362

Lib option 8

libraries

 LoadLib function 278

 of DAL scripts 5

limits

 significant numbers 79

line breaks

 MLEInput function 290, 293

line feeds 4

ListInList function 275

LoadCordFAP option 149

LoadExtractData rule 236

LoadINIFile function 277

LoadLib function

 DAL libraries 6

 defined 278

LoadXMLList function 12

LoadXMLList rule 11

localities 59, 61

locating fields 70

locating objects 89

log files

 RPLogMsg function 344

LOGFILE.DAT file 344

logical NOT operation 137

logical shift 142

Logo function 279

logos

 deleting 201

 in-lining 262

 locating 254

 renaming 335

Lower function 281

lowercase

 dates 59

M

- MailWIP function 282
- MajorVersion function 283
- master resource library
 - storing external script files 4
- MasterResource control group 9
- mathematical functions 79
- MAX function 284
- memory
 - GetINIString function 242
 - LoadINIFile function 277
 - LoadLib function 278
 - runtime error messages 41
 - SaveINIFile function 346
 - tables 57
- MEN.RES file
 - enabling the DAL debugger 40
 - executing a DAL script 7
- menus
 - executing a DAL script from 7
- messages
 - AFELog function 120
 - Ask function 129
 - Beep function 134
 - creating 299
- metadata 238, 323
- MIN function 286
- MinorVersion function 288
- minus signs 24
- Minute function 289
- miscellaneous functions 80, 83, 88
- MLEInput function
 - defined 290
 - MLETranslate function 293
- MLETranslate function
 - and MLEInput 290
 - defined 293
 - MLEInput function 293

- MOD function 296
- month abbreviations 61
- Month function 297
- MonthName function 298
- Move_It rule
 - ? function 103
- moving data to compress blank space 155
- MSG function 299
- Multicopy option 160, 213
- multi-line text area messages, creating 300
- multi-line text field format 68
- multi-line variable fields
 - MLEInput function 290, 293
 - MLETranslate function 293
- MYPAGE variables 307

N

- NAFILE.DAT files
 - embedding logos 214
- name
 - FileDrive function 225
 - FileExt function 226
 - FileName function 227
 - FilePath function 228
 - FullFileName function 234
- new line character
 - MLEInput function 290, 293
- NewFormatOnly option 377
- NL function 300
- NOT operation 137
- not required 221
- NUM function 301
- numeric constants 24
- numeric field format 68
- numeric formats 69
- Numeric function 302
- NumText function
 - defined 303
 - FrenchNumText function 233

O

- object functions
 - list of 89
 - locating objects 89
- occurrence
 - counts 71
 - PageImage function 306
- octothorpes (#) 12
- ODBC
 - DBUnloadDFD function 189
 - handler 51
- Ok buttons 299
- OldFormatOnly option 377
- OMR marks 195
- OnCreate option 20
- OnDemand, adding comments 109
- OnDemandScript option 43, 319, 320
- OnUpdate option 20
- operator required 221
- operators
 - defined 25
 - dot 56
 - source expressions 26
- options
 - setting using INI functions 77
- OR operation 138, 145
- OutMode option 43
- overflow
 - AddOVFlwSym function 119
 - AppendTxmUnique function 126
 - FieldRule function 220
 - GetOvFlwSym function 244
 - IncOvFlwSym function 261
 - ResetOvFlwSym function 337
- overflow record count
 - retrieving 386

P

- PAD function 305
- page numbering fields
 - @ function 101
- PageImage function 306
- PageInfo function 307
- pages
 - PageImage function 306
 - size 307
 - TotalPages function 382
- PaginateForm function 309
- paragraphs
 - importing 125
- parameters
 - punctuation 26
 - syntax of 48
- parentheses
 - specifying field names 24
- ParseListCount function 310
- ParseListItem function 312
- partial names
 - examples of 72, 131, 284, 286, 373
 - object functions 90
- PassWd option 51
- PathCreate function 314
- PathExist function 315
- paths
 - FileDrive function 225
 - FilePath function 228
 - FullFileName function 234
- percent signs (%) 12
- POW function 316
- Print function 317
- print functions 83
- print streams
 - adding comments 109
 - PrinterClass function 319
 - PrinterGroup function 320
- Print window 317
- Print_It function
 - defined 318
 - NL function 300

PrinterClass function 319
PrinterGroup function 320
PrinterID function 321
PrinterOutputSize function 322
printing
 Complete function 154
 in-lining logos 262
PrintViewOnly option 319, 320
procedures 48
prompts
 AFELog function 120
 creating 290
 DBDelete function 181
 FieldPrompt function 219
 Input function 263
 ODBC drivers 51
propagate
 AddForm_Propagate 112
Properties window
 comments 4
 entering DAL calcs 2, 3
PrtType option 319, 320
punctuation 26
purging WIP 202
PutFormAttrib function 323
PutINIBool function 325
PutINIString function 327

Q

Qualifier option 51
quotation marks
 @ function 101
 date formats 59
 field formats 68
 specifying field names 24
 string constants 25

R

RecipBatch function
 defined 329
RecipCopyCount function 330
recipients
 HaveRecip function 255
 page size 308
 TotalPages function 382
RecipName function
 defined 332
record lengths
 trailing spaces 187
records
 minimum number 164
Refresh function
 AddImage function 115
 defined 333
 DelLogo function 201
 Logo function 279
remainder
 returning 296
RenameLogo function
 defined 335
reserved keywords 22
ResetFld function
 defined 336
ResetOvFlwSym function 337
Retain function 338
retrieving
 a string with a new line sequence 300
 the overflow record count 386
 the SourceName field 192
return
 values 48
RETURN statements
 defined 30
 runtime error messages 42
WIPExit function 396

Right function 339
RootName function 340
Round function 341
rounding with the BankRound function 133
RouteWIP function 342
routing slips
 RouteWIP function 342
 SlipInsert function 364
RPErrMsg function 343
RPLogMsg function 344
RPWarningMsg function 345
RunMode control group 9, 149
runtime
 error messages 41
 options 39

S

SAMPCO sample resources 2, 48
SaveINIFile function 346
SaveWIP function 347
Script option 9, 20
ScriptFile option 7
scripts
 creating libraries 5
 defined 2
 disabling 2
 executing 320
 executing from a menu 7
 LoadLib function 278
 maximum size 3
 runtime error messages 41
 runtime options 8
 SlipInsert function 364
search criteria
 including spaces 236, 367
search masks
 CountRec function 164
searching
 character string list 275
Second function 348
semicolons
 formatting calculations 3
 use of 6
separators 59
sequence numbers
 HaveRecip function 255
Server option 51, 53
SetEdit function 350
SetFld function 352
SetFont function 354
SetGVM function 355
SetImagePos function
 AddImage function 115
 defined 356
SetOvFlwSym rule 119
SetProtect function 358
SetRecip function 359
SetRecipTb
 triggering the form name 384
 triggering the image name 385
SetRequiredFld function 360
setting the bit position 141
SetWIPFld function 361
shiftAmt value 139
shift-and-rotate operation 139
signatures 149
significant numbers 79
Size function 361, 362
SlipAppend function 363
SlipInsert function 364
source expressions
 defined 23
 operators 26
 punctuation 26
Source Name field
 DDTSourceName function 192
spaces
 in scripts 3, 4
 including 236, 367
 trailing 187

- SpanField function 365
- spreadsheets 310
- SrchData function 367
- standard export format
 - Complete function 154
- statements
 - data storage 38
 - operators 26
 - punctuation 26
 - separators 6
- STR function 369
- STRCompare function 370
- string functions 84
- strings
 - CodeInList function 153
 - comparing 370
 - constants 25
 - ListInList function 275
 - printing 318
 - retrieving 300
 - space and tab characters 3
 - target variables 23
- subroutines
 - BeginSub function 36
 - EndSub function 37
- SUM function 373
- SuppressBanner function
 - defined 376
- symbolic variable
 - Exists function 215
 - GetValue function 245
- symbols
 - runtime error messages 41
 - statement continuation 26
- syntax
 - errors 48
 - runtime error messages 41

T

- tab characters
 - evaluating scripts 3
 - in scripts 4
- Table function 377
- table only field format 68
- tables
 - setting up memory tables 57
- target fields 22, 24
- target variables
 - decimals 24
 - defined 22
 - integers 24
 - strings 24
- TbLkUp rule 220
- TblText rule 220
- testing a specified bit 144
- text
 - concatenating 157
 - creating a window for entering 290
 - searching for 148
- time
 - formats 86
 - functions 86
- Time function 379
- Time2Time function 380
- TimeAdd function 381
- Title option 9
- TLEs 43
- TotalPages function 382
- TotalSheets function 383
- trailing signs 302
- trailing spaces 305, 387
- translating new line characters 293
- Trigger2Archive control group 128
- TriggerFormName function 384
- TriggerImageName function 385
- triggering
 - form name 384
 - image name 385

TriggerRecsPerOvFlw 386
Trim function 387
two-digit year 173

U

Upper function 388
uppercase alphabetic field format 68
uppercase alphanumeric field format 69
uppercase dates 59
user ID
 assigning WIP 130
User option 51
UserID function 390
UserLvl function 391
using DAL 1

V

values
 absolute 105
 POW function 316
variable fields
 assign a calculation 2
 field formats 68
 functions 67
 locating fields 70
 mathematical functions 77, 79
 miscellaneous functions 80, 83, 88
 object functions 89
 resetting 336
variables
 deleting 8
 prefix names 56
 retaining 338
 target variables 22
 trailing spaces 387
VerifyKeyID
 control group 9, 20
 hook 19

version numbers
 MajorVersion function 283
 MinorVersion function 288

W

warnings
 Beep function 134
 RPWarningMsg function 345
WeekDay function 392
WhatForm function 393
WhatGroup function 394
WhatImage function 395
While...Wend statements 34
white space 3
wildcards 70
window
 creating 290
WIP
 DelWIP function 202
 KickToWIP function 271
 SaveWIP function 347
 setting WIP fields 361
WIPExit function 396
WIPFld function 397
WIPKey1 function 398
WIPKey2 function 399
WIPKeyID function 400
work-in-process, assigning 130

X

X or space field format 68
XDB database
 ? function 103
 file information 6
XML
 API functions 11
 built-in functions 12
 GetData function 236

- XML extract files 367
- XMLAttrName function 15
- XMLAttrValue function 15
- XMLFileExtract rule 11
- XMLFind function 13
- XMLFirst function 13
- XMLFirstAttrib function 14
- XMLFirstText function 14
- XMLGetCurName function 13
- XMLGetCurText function 14
- XMLNext function 13
- XMLNextAttrib function 14
- XMLNthAttrName function 15
- XMLNthAttrValue function 16
- XMLNthText function 15
- XOR operation 145
- XPath 16
- XPaths 236, 367
- XPATHW32 program 16

Y

- Y or N field format 69
- Year function 401
- YearDay function 402
- years
 - DiffYear function 211
 - forcing 2-digit 61
 - sizes 59

Z

- zero
 - runtime error messages 41

