Copyright © 2009, 2012, Oracle and/or its affiliates. All rights reserved.
The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license
agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial
property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain
interoperability with other independently created software or as specified by law, is prohibited.
The information contained in this document is subject to change without notice. If you find any problems in the documentation, please
report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license
agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or
mechanical, for any purpose.
If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States
Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial
computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific
supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation
and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent
applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software—Restricted Rights (June 1987). Oracle
USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.
The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall
be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such
applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.
The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible
for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If
you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is
not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third
party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not
responsible for any loss or damage of any sort that you may incur from dealing with any third party.
Oracle, JD Edwards, and PeopleSoft are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be
trademarks of their respective owners.
THIRD PARTY SOFTWARE NOTICES

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

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright © 2000-2009 The Apache Software Foundation. All rights reserved.

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

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright © 2009, Berkeley Software Distribution (BSD)

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

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE JDOM AUTHORS OR THE PROJECT CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright (C) 2000-2004 Jason Hunter & Brett McLaughlin. All rights reserved.

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

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Copyright © 2009 MIT

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

Copyright (c) 1995-2005 Jean-loup Gailly and Mark Adler
This software is based in part on the work of the Independent JPEG Group (http://www.ijg.org/).

This product includes software developed by the Dojo Foundation (http://dojotoolkit.org).

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright (c) 2005-2009, The Dojo Foundation. All rights reserved.

This product includes software developed by W3C.

Copyright © 2009 World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved. (http://www.w3.org/Consortium/Legal/)

This product includes software developed by Mathew R. Miller (http://www.bluecreststudios.com).

Copyright (c) 1999-2002 ComputerSmarts. All rights reserved.

This product includes software developed by Shaun Wilde and distributed via Code Project Open License (http://www.codeproject.com).

THIS WORK IS PROVIDED "AS IS", "WHERE IS" AND "AS AVAILABLE", WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES OR CONDITIONS OR GUARANTEES. YOU, THE USER, ASSUME ALL RISK IN ITS USE, INCLUDING COPYRIGHT INFRINGEMENT, PATENT INFRINGEMENT, SUITABILITY, ETC. AUTHOR EXPRESSLY DISCLAIMS ALL EXPRESS, IMPLIED OR STATUTORY WARRANTIES OR CONDITIONS, INCLUDING WITHOUT LIMITATION, WARRANTIES OR CONDITIONS OF MERCHANTABILITY, MERCHANTABILITY QUALITY OR FITNESS FOR A PARTICULAR PURPOSE, OR ANY WARRANTY OF TITLE OR NON-INFRINGEMENT, OR THAT THE WORK (OR ANY PORTION THEREOF) IS CORRECT, USEFUL, BUG-FREE OR FREE OF VIRUSES. YOU MUST PASS THIS DISCLAIMER ON WHENEVER YOU DISTRIBUTE THE WORK OR DERIVATIVE WORKS.

This product includes software developed by Chris Maunder and distributed via Code Project Open License (http://www.codeproject.com).

THIS WORK IS PROVIDED "AS IS", "WHERE IS" AND "AS AVAILABLE", WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES OR CONDITIONS OR GUARANTEES. YOU, THE USER, ASSUME ALL RISK IN ITS USE, INCLUDING COPYRIGHT INFRINGEMENT, PATENT INFRINGEMENT, SUITABILITY, ETC. AUTHOR EXPRESSLY DISCLAIMS ALL EXPRESS, IMPLIED OR STATUTORY WARRANTIES OR CONDITIONS, INCLUDING WITHOUT LIMITATION, WARRANTIES OR CONDITIONS OF MERCHANTABILITY, MERCHANTABILITY QUALITY OR FITNESS FOR A PARTICULAR PURPOSE, OR ANY WARRANTY OF TITLE OR NON-INFRINGEMENT, OR THAT THE WORK (OR ANY PORTION THEREOF) IS CORRECT, USEFUL, BUG-FREE OR FREE OF VIRUSES. YOU MUST PASS THIS DISCLAIMER ON WHENEVER YOU DISTRIBUTE THE WORK OR DERIVATIVE WORKS.
This product includes software developed by PJ Arends and distributed via Code Project Open License (http://www.codeproject.com).

THIS WORK IS PROVIDED "AS IS", "WHERE IS" AND "AS AVAILABLE", WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES OR CONDITIONS OR GUARANTEES. YOU, THE USER, ASSUME ALL RISK IN ITS USE, INCLUDING COPYRIGHT INFRINGEMENT, PATENT INFRINGEMENT, SUITABILITY, ETC. AUTHOR EXPRESSLY DISCLAIMS ALL EXPRESS, IMPLIED OR STATUTORY WARRANTIES OR CONDITIONS, INCLUDING WITHOUT LIMITATION, WARRANTIES OR CONDITIONS OF MERCHANTABILITY, MERCHANTABILITY QUALITY OR FITNESS FOR A PARTICULAR PURPOSE, OR ANY WARRANTY OF TITLE OR NON-INFRINGEMENT, OR THAT THE WORK (OR ANY PORTION THEREOF) IS CORRECT, USEFUL, BUG-FREE OR FREE OF VIRUSES. YOU MUST PASS THIS DISCLAIMER ON WHENVEVER YOU DISTRIBUTE THE WORK OR DERIVATIVE WORKS.

This product includes software developed by Erwin Tratar. This source code and all accompanying material is copyright (c) 1998-1999 Erwin Tratar. All rights reserved.

THIS SOFTWARE IS PROVIDED "AS IS" WITHOUT EXPRESS OR IMPLIED WARRANTY. USE IT AT YOUR OWN RISK! THE AUTHOR ACCEPTS NO LIABILITY FOR ANY DAMAGE/LOSS OF BUSINESS THAT THIS PRODUCT MAY CAUSE.

This product includes software developed by Sam Leffler of Silicon Graphics.

THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL SAM LEFFLER OR SILICON GRAPHICS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Copyright (c) 1988-1997 Sam Leffler
Copyright (c) 1991-1997 Silicon Graphics, Inc.

This product includes software developed by Guy Eric Schalnat, Andreas Dilger, Glenn Randers-Pehrson (current maintainer), and others. (http://www.libpng.org)

The PNG Reference Library is supplied "AS IS". The Contributing Authors and Group 42, Inc. disclaim all warranties, expressed or implied, including, without limitation, the warranties of merchantability and of fitness for any purpose. The Contributing Authors and Group 42, Inc. assume no liability for direct, indirect, incidental, special, exemplary, or consequential damages, which may result from the use of the PNG Reference Library, even if advised of the possibility of such damage.

This product includes software components distributed by the Cryptix Foundation.

THIS SOFTWARE IS PROVIDED BY THE CRYPTIX FOUNDATION LIMITED AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE CRYPTIX FOUNDATION LIMITED OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright © 1995-2005 The Cryptix Foundation Limited. All rights reserved.
This product includes software components distributed by Sun Microsystems.

This software is provided "AS IS," without a warranty of any kind. ALLEXPRESS OR IMPLIED CONDITIONS,
REPRESENTATIONS AND WARRANTIES, INCLUDING ANYIMPLIED WARRANTY OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS
LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING,
MODIFYING OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL,
CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE
THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS
BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.

This product includes software components distributed by Dennis M. Sosnoski.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS
OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN
IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright © 2003-2007 Dennis M. Sosnoski. All Rights Reserved

It also includes materials licensed under Apache 1.1 and the following XPP3 license

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright © 2002 Extreme! Lab, Indiana University. All Rights Reserved

This product includes software components distributed by CodeProject. This software contains material that is © 1994-2005 The
Ultimate Toolbox, all rights reserved.

This product includes software components distributed by Geir Landro.

Copyright © 2001-2003 Geir Landro (drop@destroydrop.com) JavaScript Tree - www.destroydrop.com/hjavascripts/tree/version 0.96
This product includes software components distributed by the Hypersonic SQL Group.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright © 1995-2000 by the Hypersonic SQL Group. All Rights Reserved.

This product includes software components distributed by the International Business Machines Corporation and others.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Copyright (c) 1995-2009 International Business Machines Corporation and others. All rights reserved.

This product includes software components distributed by the University of Coimbra.

University of Coimbra distributes this software in the hope that it will be useful but DISCLAIMS ALL WARRANTIES WITH REGARD TO IT, including all implied warranties of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. In no event shall University of Coimbra be liable for any special, indirect or consequential damages (or any damages whatsoever) resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of this software.

Copyright (c) 2000 University of Coimbra, Portugal. All Rights Reserved.

This product includes software components distributed by Steve Souza.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright © 2002, Steve Souza (admin@jamonapi.com). All Rights Reserved.

This product includes software developed by the OpenSymphony Group (http://www.opensymphony.com/).

Copyright © 2001-2004 The OpenSymphony Group. All Rights Reserved.
## 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  
7 Executing a DAL Script from a Menu  
8 Using INI Options  
10 Using Built-In Functions  
11 Checking KeyID Entries  
14 Grammar and Syntax  
14 Assignment Statements  
22 Flow Control Statements  
26 Using While...Wend Statements  
28 BeginSub and EndSub  
28 BeginSub  
29 EndSub  
30 Data Storage Statements  
31 Testing DAL Scripts  
32 Using the DAL Debugger in Documaker Workstation  
33 Runtime Error Messages  
35 DAL Script Examples

**Chapter 2, Function Reference**

41 Overview  
42 Bit/Binary Functions  
43 Database Functions  
44 ODBC Handler
45  DB2/2 Handler
46  Creating a Database Handler for an Excel Database
48  Associating Tables with Handlers
49  Accessing Database Fields
50  Setting Up Memory Tables
51  Date Functions
52  Date Formats
58  Documaker Server Functions
59  Documaker Workstation Functions
60  Docupresentment Functions
61  Field Functions
62  Field Formats
63  Numeric Formats
64  Locating Fields
68  File and Path Functions
69  Have Functions
70  INI Functions
71  Graphics Functions
72  Mathematical Functions
73  Miscellaneous Functions
74  Name Functions
75  Page Functions
76  Printer and Recipient Functions
77  Section Functions
78  String Functions
80  Time Functions
80  Time Formats
81  Using the Time Zone Functions
82  ICU Time Zones
88  WIP Functions
89  XML Functions
90  Using DAL XML Functions
91 XML Path Locator

94 Locating Objects

97 Where DAL Functions are Used

109 @

111 ?

113 ABS

114 AddAttachVAR

115 AddBlankPages

117 AddComment

118 AddDocusaveComment

119 AddForm

120 AddForm_Propagate

122 AddImage

125 AddImage_Propagate

127 AddOvFlwSym

128 AddresseeCount

129 AFELog

130 Always

131 Append

132 AppendText

134 AppendTxm

136 AppendTxmUnique

139 AppIdxRec

140 ApplyInserts

141 Ask

142 AssignWIP

143 Avg

145 BankRound

146 Beep

147 BitAnd

148 BitClear

149 BitNot

150 BitOr

151 BitRotate

153 BitSet
154 BitShift
156 BitTest
157 BitXor
158 BreakBatch
160 Call
161 Chain
162 CFind
163 ChangeLogo
165 Char
166 CharV
167 CodeInList
168 Complete
169 CompressFlds
171 ConnectFlds
174 CopyForm
175 Count
177 CountRec
178 Cut
179 DashCode
182 Date
183 Date2Date
184 DateAdd
186 DateCnv
188 Day
189 DayName
190 DaysInMonth
191 DaysInYear
192 DBAdd
193 DBClose
194 DBDelete
195 DBFind
197 DBFirstRec
198 DBNextRec
199 DBOpen
200 Creating Variable Length Records from Flat Files
201 DBPrepVars
202  DBUnloadDFD
203  DBUpdate
205  DDTSourceName
206  Dec2Hex
207  DeFormat
208  DelBlankPages
209  DelField
211  DelForm
212  DelImage
214  DelLogo
215  DelWIP
216  DestroyList
217  DeviceName
219  DiffDate
220  DiffDays
221  DiffHours
222  DiffMinutes
223  DiffMonths
224  DiffSeconds
225  DiffTime
226  DiffYears
228  DupForm
229  EmbedLogo
230  Exists
231  FieldFormat
232  FieldName
234  FieldPrompt
235  FieldRule
237  FieldType
238  FieldX
239  FieldY
240  FileDrive
241  FileExt
242  FileName
243  FilePath
244  Find
JLeft
JRight
JustField
KickToWIP
LeapYear
Left
LEN
ListInList
LoadINIFile
LoadLib
LoadXMLList
Logo
Lower
MailWIP
MajorVersion
MAX
MIN
MinorVersion
Minute
MLEInput
MLETranslate
MOD
Month
MonthName
MSG
NL
NUM
Numeric
NumText
PAD
PageImage
PageInfo
PaginateForm
ParseListCount
ParseListItem
PathCreate
339 PathExist
340 POW
341 Print
342 Print_It
343 PrinterClass
344 PrinterGroup
345 PrinterID
346 PrinterOutputSize
347 PutFormAttrib
349 PutINIBool
351 PutINIString
353 RecipBatch
354 RecipCopyCount
355 RecipientName
356 RecipName
357 Refresh
358 RemoveAttachVAR
359 RenameLogo
360 ResetFld
361 ResetOvFlwSym
362 Retain
363 Right
364 RootName
365 Round
366 RouteWIP
367 RPErrrorMsg
368 RPLogMsg
369 RPWarningMsg
370 SaveINIFile
371 SaveWIP
372 Second
373 SetDeviceName
375 SetEdit
377 SetFld
379 SetFont
380 SetFormDesc
381 SetGVM
382 SetImagePos
384 SetLink
385 SetLogo
386 SetProtect
387 SetRecip
388 SetRequiredFld
389 SetWIPFld
390 Size
391 SlipAppend
392 SlipInsert
393 SpanField
395 SrchData
397 STR
398 STRCompare
400 SUB
401 SUM
403 SuppressBanner
404 Table
406 Time
407 Time2Time
408 TimeAdd
409 TimeZone
410 TimeZone2TimeZone
412 TotalPages
413 TotalSheets
414 TriggerForm
416 TriggerFormName
417 TriggerImageName
418 TriggerRecsPerOvFlw
419 Trim
420 Upper
421 UniqueString
422 UserID
423 UserLvl
424 WeekDay
Chapter 3, Keyword Reference

Keyword Table

And
BeginSub
Break
Continue
Else
ElseIf
End
460  EndSub
461  Goto
462  If...End
464  Or
465  Return
466  While...Wend

467  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
- Checking KeyID Entries on page 11
- Grammar and Syntax on page 14
- Testing DAL Scripts on page 31
- Runtime Error Messages on page 33
- DAL Script Examples on page 35
The language you use for field calculations is called the Document Automation Language (DAL). The calculation itself is called a script. By using the proper script, you can make sure the data is processed in the manner you intend. This chapter explains calculation language and how to write scripts.

To assign a calculation to a field:

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

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

- DAL calc, if you want the system to recalculate the value of the field as soon as you highlight or enter any field. The system recalculates all Calc scripts for all fields when you highlight a new field.

- DAL script, if you want the system to recalculate the value in the field when you exit the field. The script is executed only when you exit the field containing the script reference and not during any other field actions.

- Disabled, if you do not want to run calculations during Section Check or during entry. This is a convenient way to disable the script without deleting it from the Properties window.

NOTE: The SAMPCO sample resources contain a great number of DAL examples and explanations. Be sure to check out this resource as you create DAL scripts for your company.
You enter calculations for variable fields on the Calculation tab of the field’s Properties window. Here is a sample calculation:

<table>
<thead>
<tr>
<th>Calculation</th>
<th>Return (@(&quot;Prem Basis1&quot;) * @(&quot;Prem/Ops Rate1&quot;)/100)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

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

**NOTE:** All space and tab characters inside a string constant are not considered white space, but rather part of the string.
ENTERING CALCULATIONS IN EXTERNAL FILES

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

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

To reference an external script file, you must use the CALL or CHAIN functions. The extension of the external script file is usually specified in your FSISYS.INI file as \texttt{DAL}.

If it is defined in your INI file, you do not have to specify an extension for the file name.

<table>
<thead>
<tr>
<th>Calculation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{Return(Call(“TestCalc”));}</td>
<td>Calls a calculation from an external file named TestCalc. Once completed, control returns to the script that initiated the function.</td>
</tr>
<tr>
<td>\texttt{Chain(“TestCalc”);}</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 (\texttt{\textbackslash r\textbackslash n}). You can end the file with a \texttt{CTRL+Z}; however, it is not required that you end the file with \texttt{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 (\texttt{*}) 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.
You can also create libraries of DAL scripts as structured named subroutines. The libraries which contain these named subroutines are standard ASCII files.

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

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

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

```plaintext
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 as CALL, FIND, IF, and so on. You can use upper- and lowercase letters in script names.

`BeginSub` and `EndSub` must be paired per script. You must have a space between `BeginSub` and the script name. For more information on these functions see `BeginSub` and `EndSub` on page 28.
NOTE: If you plan to use the XDB to update (separate) DDT file information, keep in mind that DAL scripts stored in the data section should follow the requirements specified for DDT data entry.

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

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

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

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

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

<table>
<thead>
<tr>
<th>Script name</th>
<th>Content</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPLETE.DAL</td>
<td>Complete ();</td>
<td>Completes each entry in WIP. This is the same result as if you chose the File, Complete option.</td>
</tr>
<tr>
<td>ASSIGN.DAL</td>
<td>AssignWIP (Fanelli);</td>
<td>Assigns each entry in WIP to the user ID <code>Fanelli</code>. This is the same result as if you chose the Formset, Assign option.</td>
</tr>
<tr>
<td>ASSIGN1.DAL</td>
<td>If WIPKey1()=&quot;Account&quot; then AssignWIP (Brown);end;</td>
<td>For each entry in WIP whose WIPKey1 equals <code>Account</code>, the script assigns the documents to the user ID <code>Brown</code>. This is the same result as if you chose the Formset, Assign Document option.</td>
</tr>
</tbody>
</table>
You can use several FSISYS.INI file control groups and options to control the way the system processes DAL functions and scripts. These options let you:

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

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

<table>
<thead>
<tr>
<th>Option</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control control group</td>
<td></td>
</tr>
<tr>
<td>FlushSymbols</td>
<td>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.</td>
</tr>
<tr>
<td>DateFmt2To4Year</td>
<td>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.</td>
</tr>
<tr>
<td>DAL control group</td>
<td></td>
</tr>
<tr>
<td>Ext</td>
<td>Enter a period and an extension. The default is <em>DAL</em>. Use this option to define the file extension used for external DAL scripts and file names.</td>
</tr>
<tr>
<td>DALFunctions control group</td>
<td></td>
</tr>
<tr>
<td>Keyword</td>
<td>Enter <code>DLLMOD-&gt;FunctionName</code>. 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).</td>
</tr>
<tr>
<td>DALLibraries control group</td>
<td></td>
</tr>
<tr>
<td>CompileWhenLoaded</td>
<td>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.</td>
</tr>
<tr>
<td>Lib</td>
<td>Use this option to specify the DAL library file to be loaded. You can specify multiple files. There is no default for this option.</td>
</tr>
</tbody>
</table>
The system also provides a number of specialized INI functions. For more information, see INI Functions on page 70.

<table>
<thead>
<tr>
<th>Option</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DALRun control group</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Script</strong></td>
<td>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 <code>DAL</code>.</td>
</tr>
<tr>
<td><strong>Title</strong></td>
<td>Enter a title. The default is <code>DALRUN - Document Automation Language Runtime</code></td>
</tr>
<tr>
<td><strong>RunMode control group</strong></td>
<td></td>
</tr>
<tr>
<td><strong>FlushDALSymbols</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Debug_Switches control group</strong></td>
<td></td>
</tr>
<tr>
<td><strong>DALLib</strong></td>
<td>Enter Yes to have the system create debug information related to the execution of library subroutines. The default is No.</td>
</tr>
<tr>
<td><strong>DebugDAL_Rules</strong></td>
<td>Enter Yes to create debug date related to the execution of each DAL function or procedure that is executed. The default is No.</td>
</tr>
<tr>
<td><strong>DumpDAL</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>VerifyKeyID control group</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Script</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>MasterResource control group</strong></td>
<td></td>
</tr>
<tr>
<td><strong>DALTriggers</strong></td>
<td>Enter the name of the DAL library file that contains your section trigger scripts (DAL triggers). The default is the name stored in the FormsetTriggers option in the MasterResource control group. If this option is omitted, the system looks for <code>SetRepTb</code>.</td>
</tr>
</tbody>
</table>
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 RACLI/RACCO), Documaker Studio, Image Editor, and various utilities such as ARCRET, ARCSPLIT, and DALRUN.

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

Here is an example:

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

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:

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

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

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

In this example, the GVM values, RunDate and PolicyNumber have not been loaded.
**CHECKING KEYID ENTRIES**

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

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

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

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

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

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

You must handle any error messages using the MSG function. See MSG on page 323 for more information.
To install the KeyID validation hook, include these INI options.

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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AFEProcedures control group</strong></td>
<td></td>
</tr>
<tr>
<td>AutoKeyID</td>
<td>Enter <code>TRNW32-&gt;TRNVerifyKeyID</code> as shown above to install the KeyID validation hook.</td>
</tr>
<tr>
<td><strong>VerifyKeyID control group</strong></td>
<td></td>
</tr>
<tr>
<td>Script</td>
<td>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.</td>
</tr>
<tr>
<td>OnCreate</td>
<td>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.</td>
</tr>
<tr>
<td>OnUpdate</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 != '9')
        msg("Invalid format found at position ", #l, ".");
        return("No");
    end

    #l -= 1;
    goto top:

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

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

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

- **Data Storage Statements**
  These statements return target variable data to the section variable fields.

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

### Assignment Statements

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

```
Target = Source expression
```

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

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

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

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”
  LAST_NAME = “Graham”
  LAST_NAME = “Graham”
  CompanyName = “Oracle”
  ```

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

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

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>Qualifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$myFloat</td>
<td>The $ denotes that this is a floating point number.</td>
</tr>
<tr>
<td>#myInteger</td>
<td>The # denotes that this is an numeric integer.</td>
</tr>
<tr>
<td>%myHandle</td>
<td>The % denotes that this is a numeric handle. No conversions should be done on this when used in DAL.</td>
</tr>
</tbody>
</table>

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

Source expression

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

Form set variable fields

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

\[ \$SubTotal = \text{sum} ("\text{Amount}\") \]

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

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

\[ @("\text{ThisField1}\") \]

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:

\[
\begin{align*}
\$My-Dec-Constant & = 3.14810 \\
\$Answer & = \$My-Dec-Constant \times 10.80
\end{align*}
\]

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:

\[
\begin{align*}
My-String-Constant & = \text{' Congratulations on your purchase. '}
\end{align*}
\]

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.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Assignment operator or logical test for equality.</td>
</tr>
<tr>
<td>+</td>
<td>Addition.</td>
</tr>
<tr>
<td>+ =</td>
<td>Value on the right is added to then assigned to the target variable on the left.</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction. Unary minus (negative)</td>
</tr>
<tr>
<td>- =</td>
<td>Value on the right is subtracted from then assigned to the target variable on the left.</td>
</tr>
</tbody>
</table>
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.

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

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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Order of Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>Highest priority—executed first</td>
</tr>
<tr>
<td>-</td>
<td>Second highest priority—executed after operations in parentheses</td>
</tr>
<tr>
<td>* /</td>
<td>Third highest priority.</td>
</tr>
<tr>
<td>+ - &amp;</td>
<td>Fourth priority</td>
</tr>
<tr>
<td>! !=</td>
<td>Fifth priority</td>
</tr>
<tr>
<td>AND OR</td>
<td>Sixth priority</td>
</tr>
<tr>
<td>=</td>
<td>Lowest priority</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Target variable</th>
<th>$AMOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source expression</td>
<td>@(&quot;BEG_BAL&quot;) + 100.00</td>
</tr>
<tr>
<td>Calculation</td>
<td>Takes the value in the section variable field named BEG_BAL adds 100.00 and places the result in the target decimal variable named $AMOUNT</td>
</tr>
<tr>
<td>Order of execution</td>
<td>Reads the expression from left to right</td>
</tr>
</tbody>
</table>
Chapter 1
Using DAL

\[ \$AMOUNT = \left( \@("PremBasis1") + \@("PremBasis2") \right) \times \@("Prem/OpsRate1")/100 \]

<table>
<thead>
<tr>
<th>Target variable</th>
<th>$AMOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source expression</td>
<td>( \left( @(&quot;PremBasis1&quot;) + @(&quot;PremBasis2&quot;) \right) \times @(&quot;Prem/OpsRate1&quot;)/100 )</td>
</tr>
<tr>
<td>Calculation</td>
<td>Takes the value in the section variable field named PremBasis1 adds the value in the section variable field named PremBasis2; multiples the total of these two fields by the value in the section variable field Prem/OpsRate1; then divides the total by 100 and places the result in the target decimal variable named AMOUNT.</td>
</tr>
<tr>
<td>Order of execution</td>
<td>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.</td>
</tr>
</tbody>
</table>

Implicit conversion

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

<table>
<thead>
<tr>
<th>Expression operands</th>
<th>Implicit conversion of operands</th>
<th>Internal result type</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRING op INTEGER</td>
<td>STRING op STRING</td>
<td>STRING</td>
</tr>
<tr>
<td>STRING op DECIMAL</td>
<td>STRING op STRING</td>
<td>STRING</td>
</tr>
<tr>
<td>STRING op STRING</td>
<td>STRING op STRING</td>
<td>STRING</td>
</tr>
<tr>
<td>INTEGER op INTEGER</td>
<td>INTEGER op INTEGER</td>
<td>INTEGER or DECIMAL</td>
</tr>
<tr>
<td>INTEGER op DECIMAL</td>
<td>DECIMAL op DECIMAL</td>
<td>DECIMAL</td>
</tr>
<tr>
<td>INTEGER op STRING</td>
<td>INTEGER op INTEGER or DECIMAL</td>
<td>INTEGER or DECIMAL</td>
</tr>
<tr>
<td>DECIMAL op INTEGER</td>
<td>DECIMAL op DECIMAL</td>
<td>DECIMAL</td>
</tr>
<tr>
<td>DECIMAL op DECIMAL</td>
<td>DECIMAL op DECIMAL</td>
<td>DECIMAL</td>
</tr>
<tr>
<td>DECIMAL op STRING</td>
<td>DECIMAL op DECIMAL</td>
<td>DECIMAL</td>
</tr>
</tbody>
</table>

* 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:

```plaintext
#val=$temp
```

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

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

In this example...

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

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

In this example...

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

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

Here is an example of implicit conversion differences:

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

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

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

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

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

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

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

**FLOW CONTROL STATEMENTS**

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

**Keywords**

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

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

These statements are explained in the following topics.
RETURN statements

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

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

```
RETURN (@("LAST_NAME") & ', ' & @("FIRST_NAME") & ' " & @("MIDDLE_INIT")")
RESULT: Takes the data in the section variable field LAST_NAME adds a comma; adds the data in the section variable field FIRST_NAME; adds the data in the section variable field MIDDLE_INIT and places this data in another section variable field.
RETURN (CALL('FirstFile'))
RESULT: Returns the result of the calculation generated by calling the script FirstFile.
```

IF statements

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

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

- Simple IF Statement

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

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

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

- Use of the keyword connector THEN is optional.

- IF Statement with ELSE Condition

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

```
IF (@("FirstAmount") < 1000.00) THEN
  $FinalAmount = @("FirstAmount") * .05;
ELSE
  $FinalAmount = @("FirstAmount") + 10.00;
END;
RETURN ($FinalAmount)
```
**CALCULATION:** If the value of the section variable field FirstAmount is less than 1000.00 then the amount is multiplied by .05 and entered in the target variable $FinalAmount.

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

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

Use of the keyword connector THEN is optional.

- **IF Statement with ELSEIF Condition**

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

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

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

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

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

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

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

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

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

```
GOTO SECTION_ONE:
```

**RESULT:** The control jumps to SECTION_ONE in a calculation.

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

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

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

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

CALL statements

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

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

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

CHAIN statements

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

```
CHAIN 'LastCalc'
```

or

```
CHAIN( 'LastCalc' )
```

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

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition</td>
<td>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.</td>
</tr>
<tr>
<td>Statements</td>
<td>One or more statements executed while the condition is true.</td>
</tr>
</tbody>
</table>

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.)

```plaintext
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.

```plaintext
Break
Break(levels)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Levels</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.)

```plaintext
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:

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

```plaintext
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.)

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

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

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

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

BEGINSub AND ENDSub

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

BeginSub

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

Syntax

BeginSub (Name)

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name associated with the subroutine</td>
<td>Yes</td>
</tr>
</tbody>
</table>

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.

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/X</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>&lt; DALRun &gt;</td>
</tr>
<tr>
<td></td>
<td>Script = file name</td>
</tr>
<tr>
<td></td>
<td>You can use any extension. The default is .DAL.</td>
</tr>
<tr>
<td>/INI</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Here are the INI options you can include in the INI file:</td>
</tr>
<tr>
<td></td>
<td>&lt; DALRun &gt;</td>
</tr>
<tr>
<td></td>
<td>Title = title string (an override to the window title)</td>
</tr>
<tr>
<td></td>
<td>Script = file name (the script to run)</td>
</tr>
<tr>
<td></td>
<td>&lt; DALFunctions &gt;</td>
</tr>
<tr>
<td></td>
<td>Keyword = DLLMOD-&gt;FunctionName</td>
</tr>
<tr>
<td></td>
<td>Keyword2 = DLLMOD-&gt;FunctionName2</td>
</tr>
<tr>
<td></td>
<td>(and so on)</td>
</tr>
<tr>
<td>/D</td>
<td>The debug switch starts the DAL Debugger. When on, the script executes in single step mode and registers this DAL function: DEBUG(&quot;message&quot;). The DEBUG function breaks execution, displays a message, and invokes the debugger in single step mode.</td>
</tr>
<tr>
<td>/T</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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 "Enable Debugger..." 502 "DBGW32->DBGEnableDebugger"
    "Enable DAL debugger." 0
    SEPARATOR
```

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

<table>
<thead>
<tr>
<th>Message</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out of memory</td>
<td>1</td>
<td>The calculation needs more memory than is available. Make more memory available to the program and try again.</td>
</tr>
<tr>
<td>Open failure on script file</td>
<td>2</td>
<td>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.</td>
</tr>
<tr>
<td>Syntax error</td>
<td>3</td>
<td>A calculation contains invalid information or does not use proper statement syntax.</td>
</tr>
<tr>
<td>Wrong number of parameters</td>
<td>4</td>
<td>A built-in function or procedure requires more parameters than are provided.</td>
</tr>
<tr>
<td>Wrong type of parameter</td>
<td>5</td>
<td>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.</td>
</tr>
<tr>
<td>Invalid or unknown symbol</td>
<td>6</td>
<td>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.</td>
</tr>
<tr>
<td>Invalid assignment statement</td>
<td>7</td>
<td>The assignment statement fails to provide a valid source expression or destination variable.</td>
</tr>
<tr>
<td>Cannot modify target</td>
<td>8</td>
<td>A statement attempted to change the value of an identifier that cannot be changed.</td>
</tr>
<tr>
<td>Unexpected internal error</td>
<td>9</td>
<td>A calculation caused an unexpected error or event that cannot be corrected.</td>
</tr>
<tr>
<td>Missing/mismatched parenthesis</td>
<td>10</td>
<td>The number of open parentheses does not match the number of close parentheses.</td>
</tr>
<tr>
<td>Invalid IF statement</td>
<td>11</td>
<td>An IF statement contains or fails to contain a keyword.</td>
</tr>
<tr>
<td>Unexpected end of script</td>
<td>12</td>
<td>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.</td>
</tr>
<tr>
<td>Invalid expression syntax</td>
<td>13</td>
<td>Generates due to a number of problems, such as: an expression fails to yield a result or encounters an unknown variable type.</td>
</tr>
<tr>
<td>Attempt to divide by zero</td>
<td>14</td>
<td>An attempt to divide a value by zero was found. Division by zero is undefined and must be avoided.</td>
</tr>
</tbody>
</table>
### Chapter 1
Using DAL

<table>
<thead>
<tr>
<th>Message</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No result value returned</td>
<td>15</td>
<td>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.</td>
</tr>
<tr>
<td>Statement label already used</td>
<td>16</td>
<td>Another label with the same name has been found within the script.</td>
</tr>
<tr>
<td>Unknown statement label</td>
<td>17</td>
<td>A GOTO statement names a label that does not occur within the script.</td>
</tr>
<tr>
<td>Invalid statement label</td>
<td>18</td>
<td>An invalid label was found.</td>
</tr>
<tr>
<td>Illegal label location</td>
<td>19</td>
<td>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.</td>
</tr>
<tr>
<td>Function out of place</td>
<td>20</td>
<td>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.</td>
</tr>
<tr>
<td>Illegal parameter value</td>
<td>21</td>
<td>A built-in function or procedure passed a parameter value that is not valid.</td>
</tr>
</tbody>
</table>
Here are some DAL script examples you can refer to as you create your own DAL scripts.

Preparing AFP or Metacode print streams for Docusave

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

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

```plaintext
< 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:

```plaintext
* 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:

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

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

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

Preparing AFP print streams for IBM’s OnDemand

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

- The AFP Conversion and Indexing Facility (ACIF), which is an IBM product, writes some AFP structures such as Tag Logical Element (TLEs) in an AFP print stream.
- Oracle Insurance’s comment support for AFP does not use TLEs. It was designed for OnDemand.
- The system uses the D3EEE 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:

```plaintext
< PrtType:AFP >
```
OnDemandScript = ONDEMAND.DAL

The ONDEMAND.DAL script file should contain this information:

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

OnDmdLib.DAL script library file

BeginSub OnDemand

* OnDemand Script is only valid for AFP print streams!
If (PrinterClass() != 'AFP') Then
    Return
End

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

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

* Add comment, ' ACIFINFO;DOCUMENT_NO:0000001'
#docnum += 1
AddComment (acifinfo & semi& docnum & colon &
Format(#docnum,'n',9999999))

* Add comment, 'MVS_FILENAME:PROD.EX.P.DCS.AFP.PREPOUT'
AddComment (mvsfile & colon & 'PROD.EX.P.DCS.AFP.PREPOUT')

* Add comment, 'EXPBPREP;PROCESS_DATE:mm-dd-yyyy'
AddComment (expbprep & semi & prodate & 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&\nFormat (Len (fldname), 'n', 999) & semi & fldname)
AddComment (recid & semi & idxdata & Format (#idxnum,'n',99) & semi&\nFormat (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&\n  Format (Len (fldname), 'n', 999) & semi & fldname)
  AddComment (recid & semi & idxdata & Format (#idxnum,'n',99) & semi&\n  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 42
- Database Functions on page 43
- Date Functions on page 51
- Documaker Server Functions on page 58
- Documaker Workstation Functions on page 59
- Field Functions on page 61
- File and Path Functions on page 68
- Have Functions on page 69
- INI Functions on page 70
- Graphics Functions on page 71
- Mathematical Functions on page 72
- Miscellaneous Functions on page 73
- Name Functions on page 74
- Page Functions on page 75
- Printer and Recipient Functions on page 76
- Section Functions on page 77
- String Functions on page 78
- Time Functions on page 80
- WIP Functions on page 88
- XML Functions on page 89
• Locating Objects on page 94
• Where DAL Functions are Used on page 97

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

Each category has a table listing the functions. The table lists and briefly describes each function. Use the table to quickly scan the available functions. Each function is discussed in detail in alphabetical order at the end of this chapter.
Functions and procedures and their associated parameters must be written in this syntax:

```
FUNCTION( parameters )
```

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

<table>
<thead>
<tr>
<th>Statement</th>
<th>This statement...</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF (FUNCTION( )) then ... END</td>
<td>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.</td>
</tr>
<tr>
<td>Y = FUNCTION( );</td>
<td>Demonstrates assigning another variable the result returned from a function.</td>
</tr>
<tr>
<td>Y = FUNCTION( FUNCTION2( ) );</td>
<td>Is similar to the last, except it also demonstrates the use of a function’s return value as a parameter to another function.</td>
</tr>
<tr>
<td>$VAL = 17.00 / FUNCTION( );</td>
<td>Demonstrates the use of a returned value as an operand in a mathematical expression.</td>
</tr>
</tbody>
</table>

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

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

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

Each function description identifies any required or optional return value.

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

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

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

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:

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

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”)
- **Qualifier** = Qualifier (no default)
- **User** = User ID (no default)
- **PassWd** = User password (no default)

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.

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

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*.

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

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.
Database Functions
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.

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
</DBTable:AppIdx>
```

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

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

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

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

**ACCESSING DATABASE FIELDS**

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

Here is an example:

Assume a table contains records with three fields:

• LOANTYPE
• PAYMENT
• DUEDATE

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

```
RECORD_LOANTYPE
RECORD_PAYMENT
RECORD_DUEDATE
```

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

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

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

```c
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:

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

or

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

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

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

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 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.

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

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

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format type</td>
<td>1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, or X. You must include a format type if you want to specify a separator, a year size, or a locale. The default is 1 (one). See <a href="#">Date format types on page 53</a> for a list of the various format types.</td>
</tr>
<tr>
<td>Separator</td>
<td>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 (/).</td>
</tr>
<tr>
<td>Year size</td>
<td>For the year size, you can specify either 2 (09) or 4 (2097) to indicate a two- or four-digit year. Use four-digit years. DAL functions use a four-digit year unless the format or the input data specifies otherwise. For example, if you enter 1/2, you specify date format 1 and a two-digit year, such as 02/17/09.</td>
</tr>
<tr>
<td>Case</td>
<td>(Optional) To return an uppercase date, such as FEBRUARY 17, 2009, include this character before the Locale: &gt; To return a lowercase date, such as february 17, 2009, include this character before the Locale: &lt; For a mixed case date, such as February 17, 2009, omit this parameter.</td>
</tr>
<tr>
<td>Locale</td>
<td>For DAL functions, you can enter an additional component to specify the locale. This is done with @xxx, where xxx indicates the locale. You must include the @, or the system ignores the locale code (xxx). US English (USD) is the default. See <a href="#">Locales on page 55</a>, for a list of locale codes.</td>
</tr>
</tbody>
</table>

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

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

*This format defaults to a two-digit year, but can be overridden to have four digits.*
Month abbreviations consist of the first three characters of the month’s name. Months with four-character names, such as June, are not abbreviated.

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

```
< Control >
DateFMT2To4Year =
```

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

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

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

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

<table>
<thead>
<tr>
<th>For this country</th>
<th>And this language</th>
<th>Use this code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argentina</td>
<td>Spanish</td>
<td>ARS</td>
</tr>
<tr>
<td>Australia</td>
<td>English</td>
<td>AUD</td>
</tr>
<tr>
<td>Austria</td>
<td>German</td>
<td>ATS</td>
</tr>
<tr>
<td>Belgium</td>
<td>Dutch</td>
<td>BED</td>
</tr>
<tr>
<td>Belgium</td>
<td>French</td>
<td>BEF</td>
</tr>
<tr>
<td>Bolivia</td>
<td>Spanish</td>
<td>BOB</td>
</tr>
<tr>
<td>Brazil</td>
<td>Portuguese</td>
<td>BRC</td>
</tr>
<tr>
<td>Canada</td>
<td>English</td>
<td>CAN</td>
</tr>
<tr>
<td>Canada</td>
<td>French</td>
<td>CAD</td>
</tr>
<tr>
<td>Chile</td>
<td>Spanish</td>
<td>CLP</td>
</tr>
<tr>
<td>Columbia</td>
<td>Spanish</td>
<td>COP</td>
</tr>
<tr>
<td>Denmark</td>
<td>Danish</td>
<td>DKK</td>
</tr>
<tr>
<td>Ecuador</td>
<td>Spanish</td>
<td>ECS</td>
</tr>
<tr>
<td>European Union</td>
<td>English</td>
<td>EUR</td>
</tr>
<tr>
<td>France</td>
<td>French</td>
<td>FRF</td>
</tr>
<tr>
<td>Finland</td>
<td>Finnish</td>
<td>FIM</td>
</tr>
<tr>
<td>Finland</td>
<td>Swedish</td>
<td>FMK</td>
</tr>
<tr>
<td>Germany</td>
<td>German</td>
<td>DEM</td>
</tr>
<tr>
<td>Guatemala</td>
<td>Spanish</td>
<td>GTQ</td>
</tr>
<tr>
<td>Iceland</td>
<td>Icelandic</td>
<td>ISK</td>
</tr>
<tr>
<td>Indonesia</td>
<td>Indonesian</td>
<td>IDR</td>
</tr>
<tr>
<td>Italy</td>
<td>Italian</td>
<td>ITL</td>
</tr>
<tr>
<td>Ireland</td>
<td>English</td>
<td>IEP</td>
</tr>
<tr>
<td>Liechtenstein</td>
<td>German</td>
<td>CHL</td>
</tr>
<tr>
<td>Luxembourg</td>
<td>French</td>
<td>FLX</td>
</tr>
<tr>
<td>Luxembourg</td>
<td>German</td>
<td>LUF</td>
</tr>
<tr>
<td>Mexico</td>
<td>Spanish</td>
<td>MXN</td>
</tr>
</tbody>
</table>
Here are some examples, using December 18, 2010:

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Format type 1</td>
<td>12/18/10</td>
</tr>
<tr>
<td>1-</td>
<td>Format type 1 with dashes (-) as the separator characters</td>
<td>12-18-10</td>
</tr>
<tr>
<td>1/2</td>
<td>Format type 1 with backslashes (/) as the separator characters and a two-digit year</td>
<td>12/18/10</td>
</tr>
<tr>
<td>14</td>
<td>Format type 1 with a four-digit year (no separator specified but the format type includes separators so the default separator (/) will be used)</td>
<td>12/18/10</td>
</tr>
<tr>
<td>B4</td>
<td>Format type B with a four-digit year (no separator specified and the format type does not include separators, so none will be included)</td>
<td>12182010</td>
</tr>
<tr>
<td>Example</td>
<td>Description</td>
<td>Result</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>4@CAD</td>
<td>Format type 4, with French Canadian as the locality. If you use &quot;4@CAD&quot; in a DAL function, the system returns the French Canadian translation of date format type 4 (Month D, YYYY with month spelled out). If you specify a locale, it must be the last component of the date format</td>
<td>décembre 18, 2010</td>
</tr>
</tbody>
</table>
The Documaker Server functions are summarized in the table below. Click on the function name to jump to a discussion of that function.

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

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddAttachVAR</td>
<td>Adds a string value as an attachment variable</td>
</tr>
<tr>
<td>GetAttachVAR</td>
<td>Returns the string value of an attachment variable</td>
</tr>
<tr>
<td>RemoveAttachVAR</td>
<td>Removes an attachment variable</td>
</tr>
</tbody>
</table>
Field functions retrieve or change data associated with variable fields defined on sections. The variable field functions are summarized in the table below. Click on the function name to jump to a discussion of that function.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>@</td>
<td>Returns the value contained in a field.</td>
</tr>
<tr>
<td>AppendText</td>
<td>Append text into a multi-line field from an external text file.</td>
</tr>
<tr>
<td>AppendTxm</td>
<td>Append text into a multi-line field from an external multi-line text area.</td>
</tr>
<tr>
<td>AppendTxmUnique</td>
<td>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.</td>
</tr>
<tr>
<td>CompressFlds</td>
<td>Compresses blank space by moving field data.</td>
</tr>
<tr>
<td>ConnectFlds</td>
<td>Repositions and aligns field text along a common horizontal coordinate so the field’s data appears concatenated.</td>
</tr>
<tr>
<td>DelField</td>
<td>Deletes a field from a section.</td>
</tr>
<tr>
<td>FieldFormat</td>
<td>Returns the format string associated with the field format type.</td>
</tr>
<tr>
<td>FieldPrompt</td>
<td>Returns the text of the prompt for a field.</td>
</tr>
<tr>
<td>FieldType</td>
<td>Returns the field format type assigned to a field.</td>
</tr>
<tr>
<td>FieldX</td>
<td>Returns the X coordinate of a field object.</td>
</tr>
<tr>
<td>FieldY</td>
<td>Returns the Y coordinate of a field object.</td>
</tr>
<tr>
<td>JustField</td>
<td>Justifies a variable field content by modifying its field coordinates.</td>
</tr>
<tr>
<td>MAX</td>
<td>Returns the maximum value found in a set of fields that share a naming method.</td>
</tr>
<tr>
<td>MIN</td>
<td>Returns the minimum value found in a set of fields that share a naming method.</td>
</tr>
<tr>
<td>NUM</td>
<td>Return the numeric value from a field regardless of the field’s format.</td>
</tr>
<tr>
<td>ResetFld</td>
<td>Clears a field of data.</td>
</tr>
<tr>
<td>SetFld</td>
<td>Assigns a value to a section field.</td>
</tr>
<tr>
<td>SetFont</td>
<td>Change the font on a field.</td>
</tr>
<tr>
<td>SetLink</td>
<td>Updates a hyperlink setting in a variable field, a graphic, or a text label.</td>
</tr>
<tr>
<td>SetProtect</td>
<td>Prevents a specified field from being altered.</td>
</tr>
<tr>
<td>SetRequiredFld</td>
<td>Changes the required option of a field to Required or Not Required.</td>
</tr>
</tbody>
</table>
Before you examine each field function individually, you should understand the available field formats and how to locate a specific field.

### FIELD FORMATS

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

<table>
<thead>
<tr>
<th>Format</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Alphabetic</td>
<td>Accepts only alphabetic characters (case sensitive)</td>
</tr>
<tr>
<td>A</td>
<td>Uppercase Alphabetic</td>
<td>Accepts only alphabetic characters and displays uppercase</td>
</tr>
<tr>
<td>B</td>
<td>Bar code</td>
<td>Accepts characters according to a bar code format string</td>
</tr>
<tr>
<td>C</td>
<td>Custom**</td>
<td>A custom formatted string</td>
</tr>
<tr>
<td>d</td>
<td>Date</td>
<td>Accepts date information according to a date format string</td>
</tr>
<tr>
<td>i</td>
<td>International Alphabetic</td>
<td>Accepts all alphabetic characters, including international characters, and is case sensitive</td>
</tr>
<tr>
<td>I</td>
<td>International Uppercase Alphabetic</td>
<td>Accepts all alphabetic characters, including international characters, and converts to uppercase</td>
</tr>
<tr>
<td>k</td>
<td>International Alphanumeric</td>
<td>Accepts all characters, including international characters, and is case sensitive</td>
</tr>
<tr>
<td>K</td>
<td>International Uppercase Alphanumeric</td>
<td>Accepts all characters, including international characters, and displays uppercase</td>
</tr>
<tr>
<td>m</td>
<td>X or space</td>
<td>Accepts an X or a space (used for a check box)</td>
</tr>
<tr>
<td>M</td>
<td>Multi-line text</td>
<td>No format</td>
</tr>
<tr>
<td>n</td>
<td>Numeric</td>
<td>Accepts numbers and uses a numeric format string</td>
</tr>
<tr>
<td>t</td>
<td>Table only</td>
<td>Accepts only information selected from a table</td>
</tr>
<tr>
<td>T</td>
<td>Time</td>
<td>Accepts only time</td>
</tr>
</tbody>
</table>
**Insertion text can be longer than a single character. Look at these examples:**

<table>
<thead>
<tr>
<th>Input Text</th>
<th>Format String</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>B105</td>
<td>1,97</td>
<td>B97105</td>
</tr>
<tr>
<td>First Street</td>
<td>6, (not 1st)</td>
<td>First (not 1st) Street</td>
</tr>
</tbody>
</table>

**NUMERIC FORMATS**

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

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“,”</td>
<td>Tells the system to automatically insert a comma in the specified position(s) of the field at data entry time.</td>
</tr>
<tr>
<td>“9”</td>
<td>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.</td>
</tr>
<tr>
<td>“.”</td>
<td>Tells the system to accept only a decimal point in the specified position at data entry time.</td>
</tr>
</tbody>
</table>
The following lists provides examples of various numeric formats:

- `ZZZZZZZ9.99`
- `+ZZZZZZZ9.99`
- `ZZZZZZZ9.99`
- `ZZZZZZZ9.99+`
- `ZZZZZZZ9.99DB`
- `ZZZZZZZ9.99CR`
- `ZZZZZZZ9.99`
- `$ZZZZZZZ9.99`
- `99999999999`
- `ZZZZZZZZZZZZZ`

### Component Description

**"Z"**

Tells the system to automatically suppress leading zeros in the specified positions of the field at data entry time.

Before version 10.0, the system would suppress zeros and insert blanks. In version 10.0 and in subsequent versions, the system will not print a blank character.

For example, if the field format was `($ZZZZZZ9.99` and you entered $255.98, the system would display ( $258.98). In version 10.0 and in subsequent versions, it shows ($258.98).

**"$"**

Tells the system to automatically insert a dollar sign in the specified position of the field at data entry time. The dollar sign may be used in a drifting manner or dollar fill. A single dollar sign in a field specifies that a currency system will always appear in the right most position before the first non-zero number. A dollar fill is specified by two dollar signs in the field format. A dollar fill specifies that leading zeros will be suppressed and replaced by the $ symbol.

**"*"**

Works much the same way as a dollar fill, but suppresses zeros with asterisks instead of dollar signs. An asterisk (*) must follow a dollar sign to a valid field format.

---

### LOCATING FIELDS

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

Field -> Section -> Form -> Group

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

Typically you will not have to specify all four components of the hierarchy to locate a given field for the DAL fields functions. By default, all field functions will search the current section which is the section that contains the script being executed. If the field you wish to reference occurs on the current section, then you do not have to specify any other information.
NOTE: You can also use the asterisk (*) as a wildcard, however, for optimal performance, avoid using wildcards (*) when searching for field, section, or form names.

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

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

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

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Section Name</th>
<th>Form Name</th>
<th>Group Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>omitted</em></td>
<td><em>omitted</em></td>
<td><em>omitted</em></td>
<td><em>omitted</em></td>
<td>In the absence of any of these parameters, the function will assume that you wish to use the current field.</td>
</tr>
<tr>
<td>“FLD”</td>
<td><em>omitted</em></td>
<td><em>omitted</em></td>
<td><em>omitted</em></td>
<td>Find FLD on the current section.</td>
</tr>
<tr>
<td>“FLD”</td>
<td>“IMG”</td>
<td><em>omitted</em></td>
<td><em>omitted</em></td>
<td>Find the first occurrence of IMG (a section) on the current form. If located, find FLD on that section.</td>
</tr>
<tr>
<td>“FLD”</td>
<td><em>omitted</em></td>
<td>“FRM”</td>
<td><em>omitted</em></td>
<td>Find the first occurrence of FRM (a form) in the current group. If located, find the first occurrence of FLD on that form. FLD may occur on any section on FRM since that parameter was omitted.</td>
</tr>
<tr>
<td>“FLD”</td>
<td><em>omitted</em></td>
<td><em>omitted</em></td>
<td>“GRP”</td>
<td>Find the first occurrence of FLD within the group, GRP. This field may be on any section on any form within that group.</td>
</tr>
</tbody>
</table>
Notice that many of these descriptions referred to the first occurrence of a particular object. This is the default search method unless an occurrence count is specified on the object name. For instance, if there are three occurrences of the field "MYFIELD" on a particular form, you would distinguish them as "MYFIELD\1", "MYFIELD\2", and "MYFIELD\3". (In practice you do not have to specify "\1" to identify the first occurrence except on those field functions that match on partial names.)

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

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

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Section Name</th>
<th>Form Name</th>
<th>Group Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“FLD”</td>
<td>“IMG”</td>
<td>“FRM”</td>
<td>omitted</td>
<td>Find the first occurrence of FRM in the current group. Find the first occurrence of IMG on that form. Find FLD on that section.</td>
</tr>
<tr>
<td>“FLD”</td>
<td>“IMG”</td>
<td>omitted</td>
<td>“GRP”</td>
<td>Find the first occurrence of IMG within the group, GRP. This section may occur on any form since that parameter was not specified. Then find FLD on that section.</td>
</tr>
<tr>
<td>“FLD”</td>
<td>“IMG”</td>
<td>“FRM”</td>
<td>“GRP”</td>
<td>Find the first occurrence of FRM within the group, GRP. Then find the first occurrence of IMG. Finally, locate FLD on that section.</td>
</tr>
</tbody>
</table>

Field Name | Section Name | Form Name | Group Name | Description |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>“FLD”</td>
<td>“IMG\2”</td>
<td>omitted</td>
<td>omitted</td>
<td>Find the second occurrence of IMG (a section) on the current form. If located, find FLD on that section.</td>
</tr>
<tr>
<td>“FLD\3”</td>
<td>omitted</td>
<td>“FRM\2”</td>
<td>omitted</td>
<td>Find the second occurrence of FRM (a form) in the current group. If located, find the third occurrence of FLD on that form. The third occurrence of FLD may occur on any section on FRM since that parameter was omitted.</td>
</tr>
<tr>
<td>“FLD\8”</td>
<td>omitted</td>
<td>omitted</td>
<td>“GRP”</td>
<td>Find the eighth occurrence of FLD within the group, GRP. This field may occur on any section or form within that group.</td>
</tr>
<tr>
<td>“FLD”</td>
<td>“IMG\5”</td>
<td>omitted</td>
<td>“GRP”</td>
<td>Find the fifth occurrence of IMG (a section) within the group, GRP. If located, find FLD on that section.</td>
</tr>
</tbody>
</table>
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.
The File and Path functions are summarized in the table below. Click on the function name to jump to a discussion of that function.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileDrive</td>
<td>Gets the drive component of a file name.</td>
</tr>
<tr>
<td>FileExt</td>
<td>Gets the extension component of a file name.</td>
</tr>
<tr>
<td>FileName</td>
<td>Gets the name component of a file name.</td>
</tr>
<tr>
<td>FilePath</td>
<td>Gets the path component of a file name.</td>
</tr>
<tr>
<td>FullFileName</td>
<td>Makes a full file name from a string containing the file name components.</td>
</tr>
<tr>
<td>PathCreate</td>
<td>Creates the subdirectory path you specify if it does not exist.</td>
</tr>
<tr>
<td>PathExist</td>
<td>Checks the path you specify to make sure it exists.</td>
</tr>
</tbody>
</table>
The Have functions are summarized in the table below. Click on the function name to jump to a discussion of that function.

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetINIBool</td>
<td>Retrieves from memory the Boolean value of an INI control group and option string.</td>
</tr>
<tr>
<td>GetINIString</td>
<td>Retrieves from memory an INI control group and option string.</td>
</tr>
<tr>
<td>INI</td>
<td>Retrieves and INI control group and option string.</td>
</tr>
<tr>
<td>LoadINIFile</td>
<td>Loads an INI file into cache memory.</td>
</tr>
<tr>
<td>PutINIBool</td>
<td>Store a Boolean value in an INI control group and option Boolean variable.</td>
</tr>
<tr>
<td>PutINIString</td>
<td>Store a string value in an INI control group and option string variable.</td>
</tr>
<tr>
<td>SaveINIFile</td>
<td>Saves the values from an INI control group and option into a file.</td>
</tr>
</tbody>
</table>

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

See Using INI Options on page 8 also for a list of the DAL-related INI control groups and options.
The graphics functions are summarized in the table below. These functions affect LOG files. Click on the function name to jump to a discussion of that function.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChangeLogo</td>
<td>Replaces an existing graphic on the section with a new graphic (LOG).</td>
</tr>
<tr>
<td>DelLogo</td>
<td>Deletes a graphic from a form.</td>
</tr>
<tr>
<td>InlineLogo</td>
<td>In-lines a graphic (LOG) into the print stream</td>
</tr>
<tr>
<td>Logo</td>
<td>Places a new graphic (LOG) at a specified position on the section.</td>
</tr>
</tbody>
</table>
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 Reference

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

**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 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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always</td>
<td>Used as a placeholder or stub.</td>
</tr>
<tr>
<td>Call</td>
<td>Suspends one calculation and executes another calculation file.</td>
</tr>
<tr>
<td>Chain</td>
<td>Calls another calculation language file.</td>
</tr>
<tr>
<td>CFind</td>
<td>Temporarily suspends one calculation and executes another calculation file.</td>
</tr>
<tr>
<td>Exists</td>
<td>Determines if a DAL symbolic variable exists.</td>
</tr>
<tr>
<td>GetValue</td>
<td>Returns a string that contains the contents of the DAL symbolic variable specified by the parameter.</td>
</tr>
<tr>
<td>LoadLib</td>
<td>Loads a file that contains a library of DAL scripts.</td>
</tr>
<tr>
<td>MajorVersion</td>
<td>Retrieves the major version number of the system being executed.</td>
</tr>
<tr>
<td>MinorVersion</td>
<td>Retrieves the minor version number of the system being executed.</td>
</tr>
<tr>
<td>Print_It</td>
<td>Prints a string on the console.</td>
</tr>
<tr>
<td>Retain</td>
<td>Retains DAL variables during transaction processing.</td>
</tr>
<tr>
<td>UniqueString</td>
<td>Returns a 45-character globally unique string.</td>
</tr>
</tbody>
</table>
The Name functions are summarized in the table below. Click on the function name to jump to a discussion of that function.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>FieldName</td>
<td>Returns the name of a field.</td>
</tr>
<tr>
<td>FormDesc</td>
<td>Retrieves a form description specified in a FORM.DAT file.</td>
</tr>
<tr>
<td>FormName</td>
<td>Returns a specified form’s name.</td>
</tr>
<tr>
<td>GroupName</td>
<td>Returns a specified group’s name.</td>
</tr>
<tr>
<td>ImageName</td>
<td>Returns a specified section’s name.</td>
</tr>
<tr>
<td>PageImage</td>
<td>Returns the name of a section on a given page number within the form set or form.</td>
</tr>
<tr>
<td>RecipientName</td>
<td>Returns from the FORM.DAT file the recipient name related to the specified section, form, or group.</td>
</tr>
<tr>
<td>RenameLogo</td>
<td>Renames a graphic (LOG).</td>
</tr>
<tr>
<td>RootName</td>
<td>Extracts and returns the root name, or the original part of the name, of a specified string.</td>
</tr>
<tr>
<td>SetFormDesc</td>
<td>Change the description of a form.</td>
</tr>
<tr>
<td>WhatForm</td>
<td>Returns the name of the form that includes the item you searched for.</td>
</tr>
<tr>
<td>WhatGroup</td>
<td>Returns the name of the group that includes the item you searched for.</td>
</tr>
<tr>
<td>WhatImage</td>
<td>Returns the name of the section that includes the item you searched for.</td>
</tr>
</tbody>
</table>
The Page functions are summarized in the table below. Click the function name to jump to a discussion of that function.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddBlankPages</td>
<td>Add blank or filler pages to the print stream</td>
</tr>
<tr>
<td>DelBlankPages</td>
<td>Removes blank or filler pages.</td>
</tr>
<tr>
<td>PageInfo</td>
<td>Gets information about the page of a form you specify.</td>
</tr>
<tr>
<td>PaginateForm</td>
<td>Applies section origins and re-paginates the form if necessary.</td>
</tr>
</tbody>
</table>
### Printer and Recipient Functions

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddComment</td>
<td>Adds a comment to the print stream.</td>
</tr>
<tr>
<td>AddDocusaveComment</td>
<td>Adds a comment to a Metacode or AFP print stream created specifically for Docusave.</td>
</tr>
<tr>
<td>BreakBatch</td>
<td>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.</td>
</tr>
<tr>
<td>DeviceName</td>
<td>Returns the current output device file name, such as the name of the current print stream output file.</td>
</tr>
<tr>
<td>IsPrintObject</td>
<td>Lets you know if the section (image), form, or group is printable, based on the current print recipient and the recipient copy count.</td>
</tr>
<tr>
<td>PrinterClass</td>
<td>Finds out the type of print stream the system is generating.</td>
</tr>
<tr>
<td>PrinterGroup</td>
<td>Retrieves the group name that is being used to generate the print stream.</td>
</tr>
<tr>
<td>PrinterID</td>
<td>Returns the printer ID assigned during a batch processing run.</td>
</tr>
<tr>
<td>PrinterOutputSize</td>
<td>Returns the approximate size of the current print output file during a batch print operation.</td>
</tr>
<tr>
<td>RecipBatch</td>
<td>Gets the name of the recipient batch file being processed. Used in banner or comment record processing.</td>
</tr>
<tr>
<td>RecipName</td>
<td>Gets the name of the recipient batch record for the transaction currently being printed. Used in banner or comment record processing.</td>
</tr>
<tr>
<td>SetDeviceName</td>
<td>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.</td>
</tr>
<tr>
<td>SuppressBanner</td>
<td>Suppresses the printing of a banner page.</td>
</tr>
</tbody>
</table>
The section (image) functions are summarized in the table below. Click on the function name to jump to a discussion of that function.

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>BankRound</td>
<td>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.</td>
</tr>
<tr>
<td>CFind</td>
<td>Finds and returns the position of a character (or string of characters) within another string of characters.</td>
</tr>
<tr>
<td>Char</td>
<td>Converts an integer into a single character.</td>
</tr>
<tr>
<td>CharV</td>
<td>Converts a single character into an integer value.</td>
</tr>
<tr>
<td>CodeInList</td>
<td>Searches for a string in a list of a strings.</td>
</tr>
<tr>
<td>Cut</td>
<td>Removes characters from a string at a specified position and returns the result.</td>
</tr>
<tr>
<td>DeFormat</td>
<td>Removes formatting from a string field and returns the result.</td>
</tr>
<tr>
<td>Find</td>
<td>Finds the position of a substring within a string and returns the result.</td>
</tr>
<tr>
<td>Format</td>
<td>Formats a string field and returns the result.</td>
</tr>
<tr>
<td>FrenchNumText</td>
<td>Converts a number into a string of words and returns the result (in French).</td>
</tr>
<tr>
<td>Insert</td>
<td>Inserts a substring into a string at a specified position and returns the result.</td>
</tr>
<tr>
<td>JCenter</td>
<td>Returns a string center justified.</td>
</tr>
<tr>
<td>JLeft</td>
<td>Returns a string left justified.</td>
</tr>
<tr>
<td>JRight</td>
<td>Returns a string right justified.</td>
</tr>
<tr>
<td>Left</td>
<td>Returns a specified number of left most characters.</td>
</tr>
<tr>
<td>LEN</td>
<td>Returns the current length of the string.</td>
</tr>
<tr>
<td>ListInList</td>
<td>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.</td>
</tr>
<tr>
<td>Lower</td>
<td>Converts all characters to lowercase and returns the result.</td>
</tr>
<tr>
<td>NL</td>
<td>Retrieves a string that contains a new line character sequence.</td>
</tr>
<tr>
<td>NumText</td>
<td>Converts a number into a string of words and returns the result (in English).</td>
</tr>
<tr>
<td>Function</td>
<td>Result</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PAD</td>
<td>Adds trailing spaces or characters and returns the result.</td>
</tr>
<tr>
<td>ParseListCount</td>
<td>Counts the indexed components within the formatted text</td>
</tr>
<tr>
<td>ParseListItem</td>
<td>Returns the indexed components from the formatted text.</td>
</tr>
<tr>
<td>Right</td>
<td>Returns a specified number of right most characters.</td>
</tr>
<tr>
<td>Round</td>
<td>Returns a number rounded to the nearest specified decimal point.</td>
</tr>
<tr>
<td>STRCompare</td>
<td>Compares two strings, considering case.</td>
</tr>
<tr>
<td>SUB</td>
<td>Returns a substring from a string at a specified position.</td>
</tr>
<tr>
<td>Trim</td>
<td>Removes end spaces and returns the result.</td>
</tr>
<tr>
<td>Upper</td>
<td>Converts all characters to uppercase and returns the result.</td>
</tr>
</tbody>
</table>
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.

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

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

TIME FORMATS

Times can be entered in several formats. The time formats are explained in this table:
The separators you can use include:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>99.99.99</td>
</tr>
<tr>
<td>,</td>
<td>99,99,99</td>
</tr>
<tr>
<td>-</td>
<td>99-99-99</td>
</tr>
<tr>
<td>b</td>
<td>99 99 99</td>
</tr>
<tr>
<td>:</td>
<td>99:99:99 (default)</td>
</tr>
</tbody>
</table>

**USING THE TIME-ZONE FUNCTIONS**

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

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

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

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

The ICU time zone functionality supports

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

  \texttt{continent/city or ocean/city}

  For example, \texttt{America/Los_Angeles} is an ID for Pacific Standard Time.

- Custom time zones based on Greenwich Mean Time (GMT), in this format:

  \texttt{"GMT [+-]h [[:]]mm "}
## ICU Time Zones

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

<table>
<thead>
<tr>
<th>Time Zones</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ACT</td>
<td>AET</td>
<td>Africa/Abidjan</td>
<td>Africa/Abidjan</td>
<td>Africa/Accra</td>
</tr>
<tr>
<td>Africa/Addis_Ababa</td>
<td>Africa/Algiers</td>
<td>Africa/Asmera</td>
<td>Africa/Bamako</td>
<td></td>
</tr>
<tr>
<td>Africa/Bangui</td>
<td>Africa/Banjul</td>
<td>Africa/Bissau</td>
<td>Africa/Blantyre</td>
<td></td>
</tr>
<tr>
<td>Africa/Brazzaville</td>
<td>Africa/Brazzaville</td>
<td>Africa/Cairo</td>
<td>Africa/Casablanca</td>
<td></td>
</tr>
<tr>
<td>Africa/Algiers</td>
<td>Africa/Algiers</td>
<td>Africa/Cairo</td>
<td>Africa/Casablanca</td>
<td></td>
</tr>
<tr>
<td>Africa/Addis_Ababa</td>
<td>Africa/Algiers</td>
<td>Africa/Asmera</td>
<td>Africa/Bamako</td>
<td></td>
</tr>
<tr>
<td>Africa/Bangui</td>
<td>Africa/Banjul</td>
<td>Africa/Bissau</td>
<td>Africa/Blantyre</td>
<td></td>
</tr>
<tr>
<td>Africa/Brazzaville</td>
<td>Africa/Brazzaville</td>
<td>Africa/Cairo</td>
<td>Africa/Casablanca</td>
<td></td>
</tr>
<tr>
<td>Africa/Algiers</td>
<td>Africa/Algiers</td>
<td>Africa/Cairo</td>
<td>Africa/Casablanca</td>
<td></td>
</tr>
<tr>
<td>Africa/Addis_Ababa</td>
<td>Africa/Algiers</td>
<td>Africa/Asmera</td>
<td>Africa/Bamako</td>
<td></td>
</tr>
<tr>
<td>Africa/Bangui</td>
<td>Africa/Banjul</td>
<td>Africa/Bissau</td>
<td>Africa/Blantyre</td>
<td></td>
</tr>
<tr>
<td>Africa/Brazzaville</td>
<td>Africa/Brazzaville</td>
<td>Africa/Cairo</td>
<td>Africa/Casablanca</td>
<td></td>
</tr>
<tr>
<td>Africa/Algiers</td>
<td>Africa/Algiers</td>
<td>Africa/Cairo</td>
<td>Africa/Casablanca</td>
<td></td>
</tr>
<tr>
<td>Africa/Addis_Ababa</td>
<td>Africa/Algiers</td>
<td>Africa/Asmera</td>
<td>Africa/Bamako</td>
<td></td>
</tr>
<tr>
<td>Africa/Bangui</td>
<td>Africa/Banjul</td>
<td>Africa/Bissau</td>
<td>Africa/Blantyre</td>
<td></td>
</tr>
<tr>
<td>Africa/Brazzaville</td>
<td>Africa/Brazzaville</td>
<td>Africa/Cairo</td>
<td>Africa/Casablanca</td>
<td></td>
</tr>
<tr>
<td>Africa/Algiers</td>
<td>Africa/Algiers</td>
<td>Africa/Cairo</td>
<td>Africa/Casablanca</td>
<td></td>
</tr>
<tr>
<td>Africa/Addis_Ababa</td>
<td>Africa/Algiers</td>
<td>Africa/Asmera</td>
<td>Africa/Bamako</td>
<td></td>
</tr>
<tr>
<td>Africa/Bangui</td>
<td>Africa/Banjul</td>
<td>Africa/Bissau</td>
<td>Africa/Blantyre</td>
<td></td>
</tr>
<tr>
<td>Africa/Brazzaville</td>
<td>Africa/Brazzaville</td>
<td>Africa/Cairo</td>
<td>Africa/Casablanca</td>
<td></td>
</tr>
<tr>
<td>Africa/Algiers</td>
<td>Africa/Algiers</td>
<td>Africa/Cairo</td>
<td>Africa/Casablanca</td>
<td></td>
</tr>
<tr>
<td>Africa/Addis_Ababa</td>
<td>Africa/Algiers</td>
<td>Africa/Asmera</td>
<td>Africa/Bamako</td>
<td></td>
</tr>
<tr>
<td>Africa/Bangui</td>
<td>Africa/Banjul</td>
<td>Africa/Bissau</td>
<td>Africa/Blantyre</td>
<td></td>
</tr>
<tr>
<td>Africa/Brazzaville</td>
<td>Africa/Brazzaville</td>
<td>Africa/Cairo</td>
<td>Africa/Casablanca</td>
<td></td>
</tr>
<tr>
<td>Africa/Algiers</td>
<td>Africa/Algiers</td>
<td>Africa/Cairo</td>
<td>Africa/Casablanca</td>
<td></td>
</tr>
<tr>
<td>Time Zones</td>
<td>Time Zones</td>
<td>Time Zones</td>
<td>Time Zones</td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
<td></td>
</tr>
<tr>
<td>America/Cuiaba</td>
<td>America/Curacao</td>
<td>America/Danmarkshavn</td>
<td>America/Dawson</td>
<td></td>
</tr>
<tr>
<td>America/Dawson_Creek</td>
<td>America/Denver</td>
<td>America/Detroit</td>
<td>America/Dominica</td>
<td></td>
</tr>
<tr>
<td>America/Edmonton</td>
<td>America/Eirunepe</td>
<td>America/El_Salvador</td>
<td>America/Ensenada</td>
<td></td>
</tr>
<tr>
<td>America/Fort_Wayne</td>
<td>America/Fortaleza</td>
<td>America/Glace_Bay</td>
<td>America/Godthab</td>
<td></td>
</tr>
<tr>
<td>America/Goose_Bay</td>
<td>America/Grand_Turk</td>
<td>America/Grenada</td>
<td>America/Guadeloupe</td>
<td></td>
</tr>
<tr>
<td>America/Guatemala</td>
<td>America/Guayaquil</td>
<td>America/Guyana</td>
<td>America/Halifax</td>
<td></td>
</tr>
<tr>
<td>America/Havana</td>
<td>America/Hermosillo</td>
<td>America/Indiana/Indianaapolis</td>
<td>America/Indiana/Knox</td>
<td></td>
</tr>
<tr>
<td>America/Indiana/Marengo</td>
<td>America/Indiana/Petersburg</td>
<td>America/Indiana/Vevey</td>
<td>America/Indiana/Vincennes</td>
<td></td>
</tr>
<tr>
<td>America/Indianapolis</td>
<td>America/Inuvik</td>
<td>America/Iqaluit</td>
<td>America/Jamaica</td>
<td></td>
</tr>
<tr>
<td>America/Jujuy</td>
<td>America/Juneau</td>
<td>America/Kentucky/Louisville</td>
<td>America/Kentucky/Monticello</td>
<td></td>
</tr>
<tr>
<td>America/Knox_IN</td>
<td>America/La_Paz</td>
<td>America/Lima</td>
<td>America/Los_Angeles</td>
<td></td>
</tr>
<tr>
<td>America/Louisville</td>
<td>America/Maceio</td>
<td>America/Managua</td>
<td>America/Manaus</td>
<td></td>
</tr>
<tr>
<td>America/Martinique</td>
<td>America/Mazatlan</td>
<td>America/Mendoza</td>
<td>America/Menominee</td>
<td></td>
</tr>
<tr>
<td>America/Merida</td>
<td>America/Mexico_City</td>
<td>America/Miquelon</td>
<td>America/Moncton</td>
<td></td>
</tr>
<tr>
<td>America/Monterrey</td>
<td>America/Montevideo</td>
<td>America/Montreal</td>
<td>America/Montserrat</td>
<td></td>
</tr>
<tr>
<td>America/Nassau</td>
<td>America/New_York</td>
<td>America/Nipigon</td>
<td>America/Nome</td>
<td></td>
</tr>
<tr>
<td>America/Noronha</td>
<td>America/North_Dakota/Center</td>
<td>America/North_Dakota/New_Salem</td>
<td>America/Panama</td>
<td></td>
</tr>
<tr>
<td>America/Pangnirtung</td>
<td>America/Paramaribo</td>
<td>America/Phoenix</td>
<td>America/Port- au-Prince</td>
<td></td>
</tr>
<tr>
<td>America/Port_of_Spain</td>
<td>America/Porto_Acre</td>
<td>America/Porto_Velho</td>
<td>America/Puerto_Rico</td>
<td></td>
</tr>
<tr>
<td>America/Rainy_River</td>
<td>America/Rankin_Inlet</td>
<td>America/Recife</td>
<td>America/Regina</td>
<td></td>
</tr>
<tr>
<td>America/Rio_Branco</td>
<td>America/Rosario</td>
<td>America/Santiago</td>
<td>America/Santo_Domingo</td>
<td></td>
</tr>
<tr>
<td>America/Sao_Paulo</td>
<td>America/Scoresby sund</td>
<td>America/Shiprock</td>
<td>America/St_Johns</td>
<td></td>
</tr>
<tr>
<td>America/St_Kitts</td>
<td>America/St_Lucia</td>
<td>America/St_Thomas</td>
<td>America/St_Vincent</td>
<td></td>
</tr>
<tr>
<td>America/Swift_Current</td>
<td>America/Tegucigalpa</td>
<td>America/Thule</td>
<td>America/Thunder_Bay</td>
<td></td>
</tr>
<tr>
<td>America/Tijuana</td>
<td>America/Toronto</td>
<td>America/Tortola</td>
<td>America/Vancouver</td>
<td></td>
</tr>
<tr>
<td>America/Virgin</td>
<td>America/Whitehorse</td>
<td>America/Winnipeg</td>
<td>America/Yakutat</td>
<td></td>
</tr>
<tr>
<td>Time Zones</td>
<td>America/Yellowknife</td>
<td>Antarctica/Cassey</td>
<td>Antarctica/Davis</td>
<td>Antarctica/DumontDUrville</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------</td>
<td>-------------------</td>
<td>-----------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>Antarctica/Mawson</td>
<td>Antarctica/McMurdo</td>
<td>Antarctica/Palmer</td>
<td>Antarctica/Rothera</td>
<td></td>
</tr>
<tr>
<td>Antarctica/South_Pole</td>
<td>Antarctica/Syowa</td>
<td>Antarctica/Vostok</td>
<td>Arctic/Longyearbyen</td>
<td></td>
</tr>
<tr>
<td>ART</td>
<td>Asia/Aden</td>
<td>Asia/Almaty</td>
<td>Asia/Amman</td>
<td></td>
</tr>
<tr>
<td>Asia/Anadyr</td>
<td>Asia/Aqtau</td>
<td>Asia/Aqtobe</td>
<td>Asia/Ashgabat</td>
<td></td>
</tr>
<tr>
<td>Asia/Ashkhabad</td>
<td>Asia/Baghdad</td>
<td>Asia/Bahrain</td>
<td>Asia/Baku</td>
<td></td>
</tr>
<tr>
<td>Asia/Bangkok</td>
<td>Asia/Beirut</td>
<td>Asia/Bishkek</td>
<td>Asia/Brunei</td>
<td></td>
</tr>
<tr>
<td>Asia/Calcutta</td>
<td>Asia/Choibalsan</td>
<td>Asia/Chongqing</td>
<td>Asia/Chungking</td>
<td></td>
</tr>
<tr>
<td>Asia/Colombo</td>
<td>Asia/Dacca</td>
<td>Asia/Damascus</td>
<td>Asia/Dhaka</td>
<td></td>
</tr>
<tr>
<td>Asia/Dili</td>
<td>Asia/Dubai</td>
<td>Asia/Dushanbe</td>
<td>Asia/Gaza</td>
<td></td>
</tr>
<tr>
<td>Asia/Harbin</td>
<td>Asia/Hong_Kong</td>
<td>Asia/Hovd</td>
<td>Asia/Irkutsk</td>
<td></td>
</tr>
<tr>
<td>Asia/Istanbul</td>
<td>Asia/Jakarta</td>
<td>Asia/Jayapura</td>
<td>Asia/Jerusalem</td>
<td></td>
</tr>
<tr>
<td>Asia/Kabul</td>
<td>Asia/Kamchatka</td>
<td>Asia/Karachi</td>
<td>Asia/Kashgar</td>
<td></td>
</tr>
<tr>
<td>Asia/Katmandu</td>
<td>Asia/Krasnoyarsk</td>
<td>Asia/Kuala Lumpur</td>
<td>Asia/Kuching</td>
<td></td>
</tr>
<tr>
<td>Asia/Kuwait</td>
<td>Asia/Macao</td>
<td>Asia/Macau</td>
<td>Asia/Magadan</td>
<td></td>
</tr>
<tr>
<td>Asia/Makassar</td>
<td>Asia/Manila</td>
<td>Asia/Muscat</td>
<td>Asia/Nicosia</td>
<td></td>
</tr>
<tr>
<td>Asia/Novosibirsk</td>
<td>Asia/Omsk</td>
<td>Asia/Oral</td>
<td>Asia/Phnom_Penh</td>
<td></td>
</tr>
<tr>
<td>Asia/Pontianak</td>
<td>Asia/Pyongyang</td>
<td>Asia/Qatar</td>
<td>Asia/Qyzylorda</td>
<td></td>
</tr>
<tr>
<td>Asia/Rangoon</td>
<td>Asia/Riyadh</td>
<td>Asia/Riyadh87</td>
<td>Asia/Riyadh88</td>
<td></td>
</tr>
<tr>
<td>Asia/Riyadh89</td>
<td>Asia/Saigon</td>
<td>Asia/Sakhalin</td>
<td>Asia/Samarkand</td>
<td></td>
</tr>
<tr>
<td>Asia/Seoul</td>
<td>Asia/Shanghai</td>
<td>Asia/Singapore</td>
<td>Asia/Taipei</td>
<td></td>
</tr>
<tr>
<td>Asia/Tashkent</td>
<td>Asia/Tbilisi</td>
<td>Asia/Tehran</td>
<td>Asia/Tel_Aviv</td>
<td></td>
</tr>
<tr>
<td>Asia/Thimbu</td>
<td>Asia/Thimphu</td>
<td>Asia/Tokyo</td>
<td>Asia/Ujung_Pandang</td>
<td></td>
</tr>
<tr>
<td>Asia/Ulaanbaatar</td>
<td>Asia/Ulan_Bator</td>
<td>Asia/Urumqi</td>
<td>Asia/Vientiane</td>
<td></td>
</tr>
<tr>
<td>Asia/Vladivostok</td>
<td>Asia/Yakutsk</td>
<td>Asia/Yekaterinburg</td>
<td>Asia/Yerevan</td>
<td></td>
</tr>
<tr>
<td>AST</td>
<td>Atlantic/Azores</td>
<td>Atlantic/Bermuda</td>
<td>Atlantic/Canary</td>
<td></td>
</tr>
<tr>
<td>Atlantic/Cape_Verde</td>
<td>Atlantic/Faeroe</td>
<td>Atlantic/Jan_Mayen</td>
<td>Atlantic/Madeira</td>
<td></td>
</tr>
<tr>
<td>Atlantic/Reykjavik</td>
<td>Atlantic/South_Georgia</td>
<td>Atlantic/St_Helena</td>
<td>Atlantic/Stanley</td>
<td></td>
</tr>
<tr>
<td>Time Zones</td>
<td>Australia/ACT</td>
<td>Australia/Adelaide</td>
<td>Australia/Brisbane</td>
<td>Australia/Broken_Hill</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------</td>
<td>--------------------</td>
<td>--------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>Australia/Canberra</td>
<td>Australia/Currie</td>
<td>Australia/Darwin</td>
<td>Australia/Hobart</td>
<td></td>
</tr>
<tr>
<td>Australia/LHI</td>
<td>Australia/Lindeman</td>
<td>Australia/Lord_Howe</td>
<td>Australia/Melbourne</td>
<td></td>
</tr>
<tr>
<td>Australia/North</td>
<td>Australia/NSW</td>
<td>Australia/Perth</td>
<td>Australia/Queensland</td>
<td></td>
</tr>
<tr>
<td>Australia/South</td>
<td>Australia/Sydney</td>
<td>Australia/Tasmania</td>
<td>Australia/Victoria</td>
<td></td>
</tr>
<tr>
<td>Australia/West</td>
<td>Australia/Yancowinna</td>
<td>BET</td>
<td>Brazil/Acre</td>
<td></td>
</tr>
<tr>
<td>Brazil/DeNoronha</td>
<td>Brazil/East</td>
<td>Brazil/West</td>
<td>BST</td>
<td></td>
</tr>
<tr>
<td>Canada/Atlantic</td>
<td>Canada/Central</td>
<td>Canada/East-Saskatchewan</td>
<td>Canada/Eastern</td>
<td></td>
</tr>
<tr>
<td>Canada/Mountain</td>
<td>Canada/Newfoundland</td>
<td>Canada/Pacific</td>
<td>Canada/Saskatchewan</td>
<td></td>
</tr>
<tr>
<td>Canada/Yukon</td>
<td>CAT</td>
<td>CET</td>
<td>Chile/Continental</td>
<td></td>
</tr>
<tr>
<td>Chile/EasterIsland</td>
<td>CNT</td>
<td>CST</td>
<td>CST6CDT</td>
<td></td>
</tr>
<tr>
<td>CTT</td>
<td>Cuba</td>
<td>EAT</td>
<td>ECT</td>
<td></td>
</tr>
<tr>
<td>EET</td>
<td>Egypt</td>
<td>Eire</td>
<td>EST</td>
<td></td>
</tr>
<tr>
<td>EST5EDT</td>
<td>Etc/GMT</td>
<td>Etc/GMT0</td>
<td>Etc/GMT+1</td>
<td></td>
</tr>
<tr>
<td>Etc/GMT+10</td>
<td>Etc/GMT+11</td>
<td>Etc/GMT+12</td>
<td>Etc/GMT+2</td>
<td></td>
</tr>
<tr>
<td>Etc/GMT+3</td>
<td>Etc/GMT+4</td>
<td>Etc/GMT+5</td>
<td>Etc/GMT+6</td>
<td></td>
</tr>
<tr>
<td>Etc/GMT+7</td>
<td>Etc/GMT+8</td>
<td>Etc/GMT+9</td>
<td>Etc/GMT-0</td>
<td></td>
</tr>
<tr>
<td>Etc/GMT-1</td>
<td>Etc/GMT-10</td>
<td>Etc/GMT-11</td>
<td>Etc/GMT-12</td>
<td></td>
</tr>
<tr>
<td>Etc/GMT-13</td>
<td>Etc/GMT-14</td>
<td>Etc/GMT-2</td>
<td>Etc/GMT-3</td>
<td></td>
</tr>
<tr>
<td>Etc/GMT-4</td>
<td>Etc/GMT-5</td>
<td>Etc/GMT-6</td>
<td>Etc/GMT-7</td>
<td></td>
</tr>
<tr>
<td>Etc/GMT-8</td>
<td>Etc/GMT-9</td>
<td>Etc/GMT0</td>
<td>Etc/Greenwich</td>
<td></td>
</tr>
<tr>
<td>Etc/UCT</td>
<td>Etc/Universal</td>
<td>Etc/UTC</td>
<td>Etc/Zulu</td>
<td></td>
</tr>
<tr>
<td>Europe/Amsterdam</td>
<td>Europe/Andorra</td>
<td>Europe/Athens</td>
<td>Europe/Belfast</td>
<td></td>
</tr>
<tr>
<td>Europe/Belgrade</td>
<td>Europe/Berlin</td>
<td>Europe/Bratislava</td>
<td>Europe/Brussels</td>
<td></td>
</tr>
<tr>
<td>Europe/Bucharest</td>
<td>Europe/Budapest</td>
<td>Europe/Chisinau</td>
<td>Europe/Copenhagen</td>
<td></td>
</tr>
<tr>
<td>Europe/Dublin</td>
<td>Europe/Gibraltar</td>
<td>Europe/Guernsey</td>
<td>Europe/Helsinki</td>
<td></td>
</tr>
<tr>
<td>Europe/Isle_of_Man</td>
<td>Europe/Istanbul</td>
<td>Europe/Jersey</td>
<td>Europe/Kaliningrad</td>
<td></td>
</tr>
<tr>
<td>Europe/Kiev</td>
<td>Europe/Lisbon</td>
<td>Europe/Ljubljana</td>
<td>Europe/London</td>
<td></td>
</tr>
</tbody>
</table>
## Time Zones

<table>
<thead>
<tr>
<th>Europe/Luxembourg</th>
<th>Europe/Madrid</th>
<th>Europe/Malta</th>
<th>Europe/Mariehamn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Europe/Minsk</td>
<td>Europe/Monaco</td>
<td>Europe/Moscow</td>
<td>Europe/Nicosia</td>
</tr>
<tr>
<td>Europe/Oslo</td>
<td>Europe/Paris</td>
<td>Europe/Prague</td>
<td>Europe/Riga</td>
</tr>
<tr>
<td>Europe/Rome</td>
<td>Europe/Samara</td>
<td>Europe/San_Marino</td>
<td>Europe/Sarajevo</td>
</tr>
<tr>
<td>Europe/Simferopol</td>
<td>Europe/Skopje</td>
<td>Europe/Sofia</td>
<td>Europe/Stockholm</td>
</tr>
<tr>
<td>Europe/Tallinn</td>
<td>Europe/Tirane</td>
<td>Europe/Tiraspol</td>
<td>Europe/Uzhgorod</td>
</tr>
<tr>
<td>Europe/Vaduz</td>
<td>Europe/Vatican</td>
<td>Europe/Vienna</td>
<td>Europe/Vilnius</td>
</tr>
<tr>
<td>Europe/Volgograd</td>
<td>Europe/Warsaw</td>
<td>Europe/Zagreb</td>
<td>Europe/Zaporozhye</td>
</tr>
<tr>
<td>Europe/Zurich</td>
<td>Factory</td>
<td>GB</td>
<td>GB-Eire</td>
</tr>
<tr>
<td>GMT</td>
<td>GMT+0</td>
<td>GMT0</td>
<td></td>
</tr>
<tr>
<td>Greenwich</td>
<td>Hongkong</td>
<td>HST</td>
<td>Iceland</td>
</tr>
<tr>
<td>IET</td>
<td>Indian/Antananarivo</td>
<td>Indian/Chagos</td>
<td>Indian/Christmas</td>
</tr>
<tr>
<td>Indian/Cocos</td>
<td>Indian/Comoro</td>
<td>Indian/Kerguelen</td>
<td>Indian/Mahe</td>
</tr>
<tr>
<td>Indian/Maldives</td>
<td>Indian/Mauritius</td>
<td>Indian/Mayotte</td>
<td>Indian/Reunion</td>
</tr>
<tr>
<td>Iran</td>
<td>Israel</td>
<td>IST</td>
<td>Jamaica</td>
</tr>
<tr>
<td>Japan</td>
<td>JST</td>
<td>Kwajalein</td>
<td>Libya</td>
</tr>
<tr>
<td>MET</td>
<td>Mexico/BajaNorte</td>
<td>Mexico/BajaSur</td>
<td>Mexico/General</td>
</tr>
<tr>
<td>Mideast/Riyadh87</td>
<td>Mideast/Riyadh88</td>
<td>Mideast/Riyadh89</td>
<td>MIT</td>
</tr>
<tr>
<td>MST</td>
<td>MST7/MDT</td>
<td>Navajo</td>
<td>NET</td>
</tr>
<tr>
<td>NST</td>
<td>NZ</td>
<td>NZ-CHAT</td>
<td>Pacific/Apiia</td>
</tr>
<tr>
<td>Pacific/Auckland</td>
<td>Pacific/Chatham</td>
<td>Pacific/Easter</td>
<td>Pacific/Efate</td>
</tr>
<tr>
<td>Pacific/Enderbury</td>
<td>Pacific/Fakaofo</td>
<td>Pacific/Fiji</td>
<td>Pacific/Funafuti</td>
</tr>
<tr>
<td>Pacific/Galapagos</td>
<td>Pacific/Gambier</td>
<td>Pacific/Guadalcanal</td>
<td>Pacific/Guam</td>
</tr>
<tr>
<td>Pacific/Honolulu</td>
<td>Pacific/Johnston</td>
<td>Pacific/Kirritimati</td>
<td>Pacific/Kosrae</td>
</tr>
<tr>
<td>Pacific/Kwajalein</td>
<td>Pacific/Majuro</td>
<td>Pacific/Marquesas</td>
<td>Pacific/Midway</td>
</tr>
<tr>
<td>Pacific/Nauru</td>
<td>Pacific/Niue</td>
<td>Pacific/Norfolk</td>
<td>Pacific/Noumea</td>
</tr>
<tr>
<td>Pacific/Pago_Pago</td>
<td>Pacific/Palau</td>
<td>Pacific/Pitcairn</td>
<td>Pacific/Ponape</td>
</tr>
<tr>
<td>Pacific/Port_Moresby</td>
<td>Pacific/Rarotonga</td>
<td>Pacific/Saipan</td>
<td>Pacific/Samoa</td>
</tr>
</tbody>
</table>
Time Functions

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

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

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

\[
\text{Return} \left( \text{TIME}\left( "2@ZAA", 13, 30, 5 \right) \right)
\]

or

\[
\text{Return} \left( \text{TIME}\left( 1, 13, 30, 5 \right) \right)
\]

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

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

<table>
<thead>
<tr>
<th>Time Zones</th>
<th>Pacific/Tahiti</th>
<th>Pacific/Tarawa</th>
<th>Pacific/Tongatapu</th>
<th>Pacific/Truk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pacific/Wake</td>
<td>Pacific/Wallis</td>
<td>Pacific/Yap</td>
<td>PLT</td>
<td></td>
</tr>
<tr>
<td>PNT</td>
<td>Poland</td>
<td>Portugal</td>
<td>PRC</td>
<td></td>
</tr>
<tr>
<td>PRT</td>
<td>PST</td>
<td>PST8PDT</td>
<td>ROC</td>
<td></td>
</tr>
<tr>
<td>ROK</td>
<td>Singapore</td>
<td>SST</td>
<td>Turkey</td>
<td></td>
</tr>
<tr>
<td>UCT</td>
<td>Universal</td>
<td>US/Alaska</td>
<td>US/Aleutian</td>
<td></td>
</tr>
<tr>
<td>US/Hawaii</td>
<td>US/Indiana-Starke</td>
<td>US/Michigan</td>
<td>US/Mountain</td>
<td></td>
</tr>
<tr>
<td>US/Pacific</td>
<td>US/Pacific-New</td>
<td>US/Samoa</td>
<td>UTC</td>
<td></td>
</tr>
<tr>
<td>VST</td>
<td>W-SU</td>
<td>WET</td>
<td>Zulu</td>
<td></td>
</tr>
</tbody>
</table>

When converting times
## 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.

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

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

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

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

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

**Scenario 1**
A Documaker program, such as GenData, loads an XML document and extracts the XML tree at the transaction level using the XMLFileExtract rule. This rule creates a list type DAL variable with a default name of %extract and pushes it onto the DAL stack. Then you can call other XML functions in a DAL script to access the XML tree and extract XML data.

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

- Add this in the AFGJOB.JDT file:
  ```
  ;XMLFileExtract;2;File=\deflib\test.xml
  ```
  The rule loads the XML file and creates a list type DAL variable to pass the XML tree to the XML API function.

- Add this in your DDT file:
  ```
  ;0;0;DALXMLSCRIPT;0;9;DALXMLSCRIPT;0;9;;DAL;Call("TEST.DAL");N;N;N;N;4792;19444;11010;
  ```
  `TEST.DAL` is the name of the DAL script file.

- Here is an example of the DAL script:
  ```
  %listH=XMLFind(%extract, "Forms", "Form");
  #rc=XMLFirst(%listH);
  if #rc=0
    return("Failed to XMLFirst");
  end
  aStr=XMLGetCurText(%listH);
  return(aStr);
  ```
  %listH denotes a list type DAL variable. #rc denotes an integer type DAL variable. aStr denotes a string type DAL variable.

**Scenario 2**
You can also load the XML document and create the XML tree at a specific section field by calling the LoadXMLList rule from a DAL script. You must set the calling procedure in the DDT file as shown in Scenario 1.

Here is an example of DAL script file:

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

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

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

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

### Axes

These axes apply:

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

### Function calls

You can use these function calls:

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

### Operators or signs

You can use these operators or signs:

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

### Expressions

You can use abbreviated syntax, as this table shows:

<table>
<thead>
<tr>
<th>For...</th>
<th>Use this abbreviation:</th>
</tr>
</thead>
<tbody>
<tr>
<td>child::*</td>
<td>*</td>
</tr>
<tr>
<td>child::para</td>
<td>para</td>
</tr>
<tr>
<td>child::chapter/child::para</td>
<td>chapter/para</td>
</tr>
<tr>
<td>child::para[position()==1]</td>
<td>para[1]</td>
</tr>
<tr>
<td>/child::chapter/child::para[position()==last()]</td>
<td>/chapter/para[last()]</td>
</tr>
<tr>
<td>child::text()</td>
<td>text()</td>
</tr>
<tr>
<td>child::node()</td>
<td>node()</td>
</tr>
<tr>
<td>child::para[attribute::type]</td>
<td>para[@type]</td>
</tr>
<tr>
<td>child::para[attribute::type=&quot;warning&quot;]</td>
<td>para[@type=&quot;warning&quot;]</td>
</tr>
</tbody>
</table>
The XMLFind function locates the XML path from the extract XML tree and returns a valid DAL variable result. It requires three input parameters, a list type DAL variable and two string type variables. They in turn pass in an XML tree, a node name from which the search starts, and XML path location for searching.

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

<table>
<thead>
<tr>
<th>For...</th>
<th>Use this abbreviation:</th>
</tr>
</thead>
<tbody>
<tr>
<td>child::para[@type=&quot;warning&quot;][position()=2]</td>
<td>para[@type=&quot;warning&quot;][2]</td>
</tr>
<tr>
<td>child::chapter[child::title]</td>
<td>chapter[title]</td>
</tr>
<tr>
<td>child::chapter[child::title=&quot;Introduction&quot;]</td>
<td>chapter[title=&quot;Introduction&quot;]</td>
</tr>
<tr>
<td>child::doc/descendant-or-self::node()/child::para</td>
<td>doc//para</td>
</tr>
<tr>
<td>attribute::*</td>
<td>@*</td>
</tr>
<tr>
<td>attribute::type</td>
<td>@type</td>
</tr>
<tr>
<td>/descendant-or-self::node()/child::para</td>
<td>//para</td>
</tr>
<tr>
<td>self::node()</td>
<td>.</td>
</tr>
<tr>
<td>self::node/descendant-or-self::node()/child::para</td>
<td>./para</td>
</tr>
<tr>
<td>parent::node()</td>
<td>..</td>
</tr>
<tr>
<td>parent::node/child::chapter</td>
<td>../chapter</td>
</tr>
<tr>
<td>parent::node/attribute::type</td>
<td>../@type</td>
</tr>
</tbody>
</table>

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

If you omit the second parameter, the search starts from the root. The return DAL variable *Result* can be either list type or string type, depending on XML path.
Here are some examples that result in different return values:

**Element list**

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

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

**Attribute list**

%attrListH=XMLFind(%extract, “Forms”, “Form/@type=’warning’”);

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

**Text list**

%TextListH= XMLFind(%extract, “Forms”, “Form/text()”);

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

**Text string**

aStr=XMLFind(%extract, Forms, “string(Form[2])”);

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

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

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

aStr=XMLFind(%extract, “Forms”, “name()”);

It returns the name of current context node.
LOCATING OBJECTS

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

item -> Section -> Form -> Group

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>Item Name</th>
<th>Section Name</th>
<th>Form Name</th>
<th>Group Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“ITEM”</td>
<td><em>omitted</em></td>
<td><em>omitted</em></td>
<td><em>omitted</em></td>
<td>Find the object on the current section.</td>
</tr>
<tr>
<td>“ITEM”</td>
<td>“IMG”</td>
<td><em>omitted</em></td>
<td><em>omitted</em></td>
<td>Find the first occurrence of IMG (a section) on the current form. If located, find ITEM on that section.</td>
</tr>
<tr>
<td>“ITEM”</td>
<td><em>omitted</em></td>
<td>“FRM”</td>
<td><em>omitted</em></td>
<td>Find the first occurrence of FRM (a form) in the current group. If located, find the first occurrence of ITEM on that form. The item may occur on any section on FRM since that parameter was omitted.</td>
</tr>
<tr>
<td>“ITEM”</td>
<td><em>omitted</em></td>
<td><em>omitted</em></td>
<td>“GRP”</td>
<td>Find the first occurrence of ITEM within the group, GRP. This item may be on any section on any form within that group.</td>
</tr>
</tbody>
</table>
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 \ to identify the first occurrence except with those functions that match on partial names.)

<table>
<thead>
<tr>
<th>Item Name</th>
<th>Section Name</th>
<th>Form Name</th>
<th>Group Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“ITEM”</td>
<td>“IMG”</td>
<td>“FRM”</td>
<td><em>omitted</em></td>
<td>Find the first occurrence of FRM in the current group. Find the first occurrence of IMG on that form. Find ITEM on that section.</td>
</tr>
<tr>
<td>“ITEM”</td>
<td>“IMG”</td>
<td><em>omitted</em></td>
<td>“GRP”</td>
<td>Find the first occurrence of IMG within the group, GRP. This section may occur on any form since that parameter was not specified. Then find ITEM on that section.</td>
</tr>
<tr>
<td>“ITEM”</td>
<td>“IMG”</td>
<td>“FRM”</td>
<td>“GRP”</td>
<td>Find the first occurrence of FRM within the group, GRP. Then find the first occurrence of IMG. Finally, find ITEM on that section.</td>
</tr>
</tbody>
</table>

Locating Sections

| “IMG” | *omitted* | *omitted* | Find the occurrence of the section on the current form. |
| “IMG” | “FRM”    | *omitted* | Find the occurrence of the section on the form named. The form is assumed to be in the current group. |
| “IMG” | *omitted* | “GRP”    | Locate the section within the named group. |
| “IMG” | “FRM”    | “GRP”    | Locate the form in the specified group. Then locate the section on that form. |

Locating Forms

| “FRM” | *omitted* | Locate the form within the current group. |
| “FRM” | “GRP”    | Locate the form in the specified group. |

Locating Groups

| “GRP” | Locate the specified group. |
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.

<table>
<thead>
<tr>
<th>Item Name</th>
<th>Section Name</th>
<th>Form Name</th>
<th>Group Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ITEM&quot;</td>
<td>&quot;IMG\2&quot;</td>
<td>&quot;omitted&quot;</td>
<td>&quot;omitted&quot;</td>
<td>Find the second occurrence of IMG (a section) on the current form. If located, find ITEM on that section.</td>
</tr>
<tr>
<td>&quot;ITEM\3&quot;</td>
<td>&quot;omitted&quot;</td>
<td>&quot;FRM\2&quot;</td>
<td>&quot;omitted&quot;</td>
<td>Find the second occurrence of FRM (a form) in the current group. If located, find the third occurrence of ITEM on that form.</td>
</tr>
<tr>
<td>&quot;ITEM&quot;</td>
<td>&quot;IMG\5&quot;</td>
<td>&quot;omitted&quot;</td>
<td>&quot;GRP&quot;</td>
<td>Find the fifth occurrence of IMG (a section) within the group, GRP. If located, find ITEM on that section.</td>
</tr>
</tbody>
</table>

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.
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.

<table>
<thead>
<tr>
<th>Function/Procedure</th>
<th>Affects form entry (Documaker Workstation)</th>
<th>Affects form processing (Documaker Server)</th>
</tr>
</thead>
<tbody>
<tr>
<td>@</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>ABS</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>AddAttachVAR</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>AddBlankPages</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>AddComment</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>AddDocusaveComment</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>AddForm</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>AddForm_Propagate</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>AddImage</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>AddImage_Propagate</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>AddOvFlwSym</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>AFELog</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>AppendText</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>AppendTxm</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>AppendTxmUnique</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ApplIdxRec</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ApplyInserts</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Ask</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>AssignWIP</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Avg</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>BankRound</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

* While these XML-related functions affect both Documaker Workstation and Documaker Server, Documaker Server can have a default variable that refers to the transaction loaded in the XML extract. No such variable would exist automatically within Documaker Workstation.
<table>
<thead>
<tr>
<th>Function/Procedure</th>
<th>Affects form entry (Documaker Workstation)</th>
<th>Affects form processing (Documaker Server)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beep</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>BitAnd</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>BitClear</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>BitNot</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>BitOr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>BitRotate</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>BitSet</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>BitShift</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>BitTest</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>BitXor</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>BreakBatch</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>CFind</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ChangeLogo</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>CodeInList</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Complete</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>CompressFlds</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ConnectFlds</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>CopyForm</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Count</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>CountRec</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Cut</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DashCode</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Date</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Date2Date</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DateAdd</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

* While these XML-related functions affect both Documaker Workstation and Documaker Server, Documaker Server can have a default variable that refers to the transaction loaded in the XML extract. No such variable would exist automatically within Documaker Workstation.
<table>
<thead>
<tr>
<th>Function/Procedure</th>
<th>Affects form entry (Documaker Workstation)</th>
<th>Affects form processing (Documaker Server)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DateCnv</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Day</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DayName</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DaysInMonth</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DaysInYear</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DBAdd</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DBClose</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DBDelete</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DBFind</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DBFirstRec</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DBNextRec</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DBOpen</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DBPrepVars</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DBUnloadDFD</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DBUpdate</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DDTSourceName</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Dec2Hex</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DeFormat</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DelBlankPages</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DelField</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DelForm</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DelImage</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DelLogo</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DelWIP</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DestroyList *</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

* While these XML-related functions affect both Documaker Workstation and Documaker Server, Documaker Server can have a default variable that refers to the transaction loaded in the XML extract. No such variable would exist automatically within Documaker Workstation.
<table>
<thead>
<tr>
<th>Function/Procedure</th>
<th>Affects form entry (Documaker Workstation)</th>
<th>Affects form processing (Documaker Server)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeviceName</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DiffDate</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DiffDays</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DiffHours</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DiffMinutes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DiffMonths</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DiffSeconds</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DiffTime</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DiffYears</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DupForm</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>EmbedLogo</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Exists</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>FieldFormat</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>FieldName</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>FieldPrompt</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>FieldRule</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>FieldType</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>FieldX</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>FieldY</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>FileDrive</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>FileExt</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>FileName</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>FilePath</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Find</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Format</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

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

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

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

<table>
<thead>
<tr>
<th>Function/Procedure</th>
<th>Affects form entry (Documaker Workstation)</th>
<th>Affects form processing (Documaker Server)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IncOvFlwSym</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>InlineLogo</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Input</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Insert</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>INT</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>IsPrintObject</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>IsXMLLError *</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>JCenter</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>JLeft</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>JRight</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>JustField</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>KickToWIP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>LeapYear</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Left</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>LEN</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ListInList</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>LoadINIFile</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>LoadLib</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>LoadXMLList *</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Logo</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Lower</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MailWIP</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>MajorVersion</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MAX</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MIN</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

* While these XML-related functions affect both Documaker Workstation and Documaker Server, Documaker Server can have a default variable that refers to the transaction loaded in the XML extract. No such variable would exist automatically within Documaker Workstation.
<table>
<thead>
<tr>
<th>Function/Procedure</th>
<th>Affects form entry (Documaker Workstation)</th>
<th>Affects form processing (Documaker Server)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinorVersion</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Minute</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MLEInput</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>MLETranslate</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>MOD</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Month</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MonthName</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MSG</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>NL</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>NUM</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Numeric</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>NumText</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>PAD</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>PageImage</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>PageInfo</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>PaginateForm</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ParseListCount</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ParseListItem</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>PathCreate</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>PathExist</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>POW</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Print</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Print_It</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>PrinterClass</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>PrinterGroup</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

* While these XML-related functions affect both Documaker Workstation and Documaker Server, Documaker Server can have a default variable that refers to the transaction loaded in the XML extract. No such variable would exist automatically within Documaker Workstation.
<table>
<thead>
<tr>
<th>Function/Procedure</th>
<th>Affects form entry (Documaker Workstation)</th>
<th>Affects form processing (Documaker Server)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrinterID</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>PrinterOutputSize</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>PutFormAttrib</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>PutINIBool</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>PutINIString</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>RecipBatch</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>RecipCopyCount</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>RecipientName</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>RecipName</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Refresh</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>RemoveAttachVAR</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>RenameLogo</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ResetFld</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ResetOvFlwSym</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Retain</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Right</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>RootName</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Round</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>RouteWIP</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>RPEErrorMsg</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>RPLLogMsg</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>RPWarningMsg</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>SaveINIFile</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SaveWIP</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Second</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

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

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

* While these XML-related functions affect both Documaker Workstation and Documaker Server, Documaker Server can have a default variable that refers to the transaction loaded in the XML extract. No such variable would exist automatically within Documaker Workstation.
<table>
<thead>
<tr>
<th>Function/Procedure</th>
<th>Affects form entry (Documaker Workstation)</th>
<th>Affects form processing (Documaker Server)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimeAdd</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TimeZone</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TimeZone2TimeZone</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TotalPages</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>TotalSheets</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TriggerFormName</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>TriggerImageName</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>TriggerRecsPerOvFlw</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Trim</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Upper</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>UniqueString</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>UserID</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>UserLvl</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>WeekDay</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>WhatForm</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>WhatGroup</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>WhatImage</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>WIPExit</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>WIPFld</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>WIPKey1</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>WIPKey2</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>WIPKeyID</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>XMLAttrName *</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>XMLAttrValue *</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>XMLFind *</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

* While these XML-related functions affect both Documaker Workstation and Documaker Server, Documaker Server can have a default variable that refers to the transaction loaded in the XML extract. No such variable would exist automatically within Documaker Workstation.
<table>
<thead>
<tr>
<th>Function/Procedure</th>
<th>Affects form entry (Documaker Workstation)</th>
<th>Affects form processing (Documaker Server)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMLFirst *</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>XMLFirstAttrib *</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>XMLFirstText *</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>XMLGetCurName *</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>XMLGetCurText *</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>XMLNext *</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>XMLNextAttrib *</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>XMLNextText *</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>XMLNthAttrName *</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>XMLNthAttrValue *</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>XMLNthText *</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Year</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>YearDay</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

* While these XML-related functions affect both Documaker Workstation and Documaker Server, Documaker Server can have a default variable that refers to the transaction loaded in the XML extract. No such variable would exist automatically within Documaker Workstation.
Use this function to return the current value contained in a section field. The @ function is also called the get field function. The @ symbol is used because it is easy to recognize in script statements and it reduces the amount of typing required.

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

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

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

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

**Syntax**

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

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

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

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

**Example**

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

For the third example, assume the current form is the third page of the form set being processed. For the fourth example, assume the section Header3 is on the second page of the form ABC.
<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return(@( ))</td>
<td>1234.23</td>
<td>Returns the value in the current field.</td>
</tr>
<tr>
<td>Return(@(&quot;MyField&quot;))</td>
<td>1234.23</td>
<td>Returns the value in the named field, located on the current section.</td>
</tr>
<tr>
<td>Return ( @(&quot;Formset Page Num&quot;) )</td>
<td>3</td>
<td>Returns the value in the field named “Formset Page Num” on the current section.</td>
</tr>
<tr>
<td>Return ( @(Form Page Num&quot;), &quot;Header3&quot;,&quot;ABC&quot;) )</td>
<td>2</td>
<td>Returns the value in the field named “Form Page Num” in section, “Header3” on form “ABC”.</td>
</tr>
</tbody>
</table>

See also
- Field Functions on page 61
- Field Formats on page 62
- Locating Fields on page 64
- NUM on page 325
Use this function to retrieve data from a record in the extract file. This function only uses
the specified entry (LookUpName) in the XDB database to determine the:

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

**Syntax**

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

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

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

Assume you have these entries defined in the XDB:

<table>
<thead>
<tr>
<th>Name</th>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Conditional</th>
<th>Required</th>
<th>Rule</th>
<th>Mask</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>PosRec</td>
<td>PolRec</td>
<td>0 0</td>
<td>9 10</td>
<td>No</td>
<td>No</td>
<td>MOVE_IT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1_f_name</td>
<td>POL_REC</td>
<td>10</td>
<td>16</td>
<td>No</td>
<td>No</td>
<td>MOVE_IT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1_m_initial</td>
<td>POL_REC</td>
<td>30</td>
<td>1</td>
<td>No</td>
<td>No</td>
<td>MOVE_It</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1_l_name</td>
<td>POL_REC</td>
<td>55</td>
<td>25</td>
<td>No</td>
<td>No</td>
<td>MOVE_It</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Issue_Date</td>
<td>POL_REC</td>
<td>80</td>
<td>8</td>
<td>No</td>
<td>No</td>
<td>DATE Tâm</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>F_Name</td>
<td>POL_REC</td>
<td>20</td>
<td>25</td>
<td>No</td>
<td>No</td>
<td>MOVE_It</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>9</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

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

**Example 1**
Assume the Driver field has this script and uses the DAL rule.

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

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

Morris V. Vanelli
Example 2  In this example, assume there are ten fields (driver01, driver02, and so on) on the section, the first field includes this script and it uses the DAL rule.

    Call (*drivers.dal*)

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

* Determine number of ‘pol_rec’ records exist in the transaction.

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

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

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

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

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

Example 3  In this example, assume the License Issued field has this script and uses the DAL rule.

    Return ( ?("issue_date") );

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

    September 21, 1957

See also  FieldRule on page 235
 GetData on page 253
 Documaker Server Functions on page 58
 Field Formats on page 62
 Locating Fields on page 64
ABS

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

Syntax

ABS (Number)

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

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

Example

Here are some examples:

(Assume the current field contains the number 250.)

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

See also Mathematical Functions on page 72
**AddAttachVAR**

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

Syntax

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter the name of the attachment variable.</td>
</tr>
<tr>
<td>Value</td>
<td>Enter the value you want to add.</td>
</tr>
<tr>
<td>DSIqueue</td>
<td>(Optional) Enter one (1) for input or two (2) for output. The default is two (2).</td>
</tr>
</tbody>
</table>

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

See also

- [Docupresentment Functions on page 60](#)
- [GetAttachVAR on page 252](#)
- [RemoveAttachVAR on page 358](#)
**AddBlankPages**

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

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

**Syntax**

```plaintext
AddBlankPages (FAP)
```

**Parameter**  
**Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAP</td>
<td>Enter the name of the FAP file you want the system to use as a filler page. The default is blank.</td>
</tr>
</tbody>
</table>

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:

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

Here is an example of the BANNER.DAL file:

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

**NOTE:** See *Documaker Administration Guide* for more information on using banner processing.
Here is a table which shows when blank pages will be added, based on the duplex setting of the two current pages and the duplex setting of the next page. Blank means a blank page will be added, As is means no blank page is needed and the form will be left as is.

<table>
<thead>
<tr>
<th>If the current page is</th>
<th>Unknown</th>
<th>Front</th>
<th>Back</th>
<th>None</th>
<th>Short</th>
<th>Rolling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unknown</td>
<td>Blank</td>
<td>Blank</td>
<td>As is</td>
<td>Blank</td>
<td>Blank</td>
<td>Blank</td>
</tr>
<tr>
<td>None</td>
<td>Blank</td>
<td>Blank</td>
<td>As is</td>
<td>Blank</td>
<td>Blank</td>
<td>Blank</td>
</tr>
<tr>
<td>Front</td>
<td>Blank</td>
<td>Blank</td>
<td>As is</td>
<td>Blank</td>
<td>Blank</td>
<td>As is</td>
</tr>
<tr>
<td>Short</td>
<td>Blank</td>
<td>Blank</td>
<td>As is</td>
<td>Blank</td>
<td>Blank</td>
<td>As is</td>
</tr>
<tr>
<td>Rolling (Front)</td>
<td>Blank</td>
<td>Blank</td>
<td>As is</td>
<td>Blank</td>
<td>Blank</td>
<td>As is</td>
</tr>
<tr>
<td>Back</td>
<td>As is</td>
<td>As is</td>
<td>Blank</td>
<td>As is</td>
<td>As is</td>
<td>As is</td>
</tr>
<tr>
<td>Rolling (Back)</td>
<td>As is</td>
<td>As is</td>
<td>Blank</td>
<td>As is</td>
<td>As is</td>
<td>As is</td>
</tr>
</tbody>
</table>

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

The API to call from custom code is as follows:

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

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

See also  
DelBlankPages on page 208  
SuppressBanner on page 403  
Page Functions on page 75  
Miscellaneous Functions on page 73  
Creating a DAL Script Library on page 5
**AddComment**

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

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

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

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

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

### Syntax

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

### Parameter Description

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

### Example

Here are some examples:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddComment('This is an example')</td>
<td>1 or 0</td>
<td>Adds the comment, “This is an example”, to the print stream.</td>
</tr>
<tr>
<td>* Add a comment to PCL print stream Comment = AppIdxRec(); AddComment(comment, 1);</td>
<td>1 or 0</td>
<td>Adds a comment containing the archive record ID. The second parameter (1) indicates that the string is to be added as an ASCII string.</td>
</tr>
</tbody>
</table>

### See also

Printer and Recipient Functions on page 76
**AddDocusaveComment**

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

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

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

**Syntax**

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

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comment</td>
<td>Enter the string to be written as a comment in the print stream.</td>
</tr>
</tbody>
</table>
| Convert   | (Optional) Choose from these options:  

- 0 - (zero) convert the string to EBCDIC  
- 1 - convert the string to ASCII  
- 2 - do not convert the string.  

For Docusave, you will always want EBCDIC comments. The default is zero (0). |

**Example**

Here are some examples:

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

**See also**

- Printer and Recipient Functions on page 76
- DAL Script Examples on page 35
**AddForm**

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

**Syntax**

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

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form</td>
<td>Enter the name of a form in the specified group.</td>
</tr>
<tr>
<td>Insert</td>
<td>Enter the name of a form <em>after</em> which the new form should be inserted. The default is to append after the last form in the group.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of a group to contain the specified form. The default is the current group.</td>
</tr>
</tbody>
</table>

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

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

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

**Example**

Here are some examples:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddForm( “Form1”)</td>
<td>1 or 0</td>
<td>Add the named form after the last form in the current group.</td>
</tr>
<tr>
<td>AddForm( “Form”, “Form\1”, GRP”)</td>
<td>1 or 0</td>
<td>Insert the named form after the first occurrence of that form within the named group.</td>
</tr>
</tbody>
</table>

**See also**

AddForm_Propagate on page 120
CopyForm on page 174
DupForm on page 228
TriggerForm on page 414
WIP Functions on page 88
**AddForm_Propagate**

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

**Syntax**

\[\text{AddForm\_Propagate (Form, Insert, Group)}\]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form</td>
<td>Enter the name of a form in the specified group.</td>
</tr>
<tr>
<td>Insert</td>
<td>Enter the name of a form after which the new form should be inserted. The default is to append after the last form in the group.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of a group to contain the specified form. The default is the current group.</td>
</tr>
</tbody>
</table>

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

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

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

Keep in mind...

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

**Example**

Here are some examples:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Results</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddForm_Propagate( &quot;0002EA&quot;)</td>
<td>1 (success) or 0 (failed)</td>
<td>Add the named form, 0002EA, after the last form in the current group.</td>
</tr>
<tr>
<td>AddForm_Propagate ( &quot;C22510WGIM&quot;, &quot;C22510WGIM \1&quot;, &quot;Sales&quot;)</td>
<td>1 (success) or 0 (failed)</td>
<td>Insert the named form, C22510WGIM, after the first occurrence of specified form, C22510WGIM, within the named group, Sales. See sample output.</td>
</tr>
</tbody>
</table>
Original form:  C22510WGIM

<table>
<thead>
<tr>
<th>Employer:</th>
<th>Oracle Insurance</th>
<th>form name =  C22510WGIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee:</td>
<td>J. Stewart</td>
<td>section name = GENRCHDR</td>
</tr>
<tr>
<td>Date of Loss:</td>
<td>12/11/10</td>
<td></td>
</tr>
<tr>
<td>File Number:</td>
<td>12345</td>
<td></td>
</tr>
<tr>
<td>State Case Num:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Samford and Son

Sincerely,

Workers’ Compensation Unit

cc: J. Stewart

Added form:  C22510WGIM\2

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

<table>
<thead>
<tr>
<th>Employer:</th>
<th>Oracle Insurance</th>
<th>form name =  C22510WGIM\2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee:</td>
<td>J. Stewart</td>
<td>section name = GENRCHDR</td>
</tr>
<tr>
<td>Date of Loss:</td>
<td>12/11/10</td>
<td></td>
</tr>
<tr>
<td>File Number:</td>
<td>12345</td>
<td></td>
</tr>
<tr>
<td>State Case Num:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Samford and Son

Sincerely,

Workers’ Compensation Unit

See also  AddForm on page 119
CopyForm on page 174
DupForm on page 228
TriggerForm on page 414
WIP Functions on page 88
## AddImage

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

### Syntax

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

### Parameter Description

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

```plaintext
AddImage ( "myFAP", "mainImage", , , 1,1)
```

This example omits the Form and Group parameters, but does specify the Flag parameter as well as the Pagination parameter. Note: If you enter zero (0) or omit this parameter, the function works as it prior to version 11.2. The default is zero (0). |

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

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

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

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

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

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

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

**Example**

Here are some examples:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddImage (&quot;IMG1&quot;)</td>
<td>1 - if successfully added. 0 - if not added</td>
<td>Insert the named section, IMG1, on a new page after the current page.</td>
</tr>
<tr>
<td>AddImage(&quot;NEW1&quot;, &quot;IMG\3&quot;, &quot;GRP&quot;)</td>
<td>1 - if successfully added. 0 - if not added</td>
<td>Insert the named section, NEW1, after the third occurrence of IMG, within GRP. This section is placed on a new page after the third occurrence of the specified section.</td>
</tr>
<tr>
<td>AddImage (&quot;IMG1&quot;, 1)</td>
<td>1 - if successfully added. 0 - if not added</td>
<td>Insert the named section, IMG1, after the current section on the same page.</td>
</tr>
<tr>
<td>AddImage(&quot;NEW1&quot;, &quot;IMG\3&quot;, 1)</td>
<td>1 - if successfully added. 0 - if not added</td>
<td>Insert the named section, NEW1, after the third occurrence of IMG on the same page.</td>
</tr>
</tbody>
</table>

See also DellImage on page 212
ADDIMAGE_PROPAGATE

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

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

Syntax

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

Parameter Description

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

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

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

Keep in mind...

- Global multi-line variable field data is not propagated to the added form.
- The system does not move sections already defined for a page. Therefore, you can overlay existing sections on the page.
- Make sure you do not unintentionally overlay an existing section. You can move the new section using the ImageRect and SetImagePos procedures.
- If you use this procedure to add sections to forms and you also import and export those forms, be sure to set the IgnoreInvalidImage option in the ImpFile_CD control group. Otherwise, users will receive an error message. For detailed instructions, see the Documaker Workstation Administration Guide.
When adding a section, there is no way for you to specify what section options or recipients you want included on the new section. So, the AddImage_Propagate procedure takes the missing information from an associated section. The system will, however, exclude the In-lined, Copy on Overflow, Duplex Front, Duplex Back, and Caused by Overflow settings. These options are not normally associated with a section being added via DAL.

Example

Here are some examples:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddImage_Propagate (&quot;IMG1&quot;)</td>
<td>1 - if successfully added. 0 - if not added.</td>
<td>Insert the named section, IMG1, on a new page after the current page.</td>
</tr>
<tr>
<td>AddImage_Propagate(&quot;NEW1&quot;, &quot;IMG\3&quot;, &quot;GRP&quot;)</td>
<td>1 - if successfully added. 0 - if not added.</td>
<td>Insert the named section, NEW1, after the third occurrence of IMG, within GRP. This section is placed on a new page after the third occurrence of the specified section.</td>
</tr>
<tr>
<td>AddImage_Propagate (&quot;IMG1&quot;, 1)</td>
<td>1 - if successfully added. 0 - if not added.</td>
<td>Insert the named section, IMG1, after the current section on the same page.</td>
</tr>
<tr>
<td>AddImage_Propagate(&quot;NEW1&quot;, &quot;IMG\3&quot;, 1)</td>
<td>1 - if successfully added. 0 - if not added.</td>
<td>Insert the named section, NEW1, after the third occurrence of IMG on the same page.</td>
</tr>
</tbody>
</table>

See also

AddImage on page 122
WIP Functions on page 88
ADD0VFLWSYM

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 AFGJOBJ.DJT file.

Syntax

AddOvFlwSym (Form, Symbol, MaxRecords)

Parameter Description

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

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

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

Example

Here are some examples:

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

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

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

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

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

Here is another example:

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

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

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

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

See also

GetOvFlwSym on page 262
IncOvFlwSym on page 280
ResetOvFlwSym on page 361
Documaker Server Functions on page 58
**ADDRESSEE COUNT**

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

**Syntax**

```
AddresseeCount (RecipName)
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RecipName</td>
<td>Enter the name of the recipient for which you want to know the number of addressees. This name should be a recipient name associated with the document set.</td>
</tr>
</tbody>
</table>

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

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

**Example**

Here are some examples.

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Cnt = AddresseeCount( &quot;Memo&quot; )</td>
<td>6</td>
<td>The #Cnt variable contains 6 to reflect the number of unique addressees associated with the memo recipient documents for the document set.</td>
</tr>
<tr>
<td>#Cnt = AddresseeCount( &quot;Agent&quot; )</td>
<td>0</td>
<td>The document set is not distributed to the Agent recipient.</td>
</tr>
<tr>
<td>#Cnt = AddresseeCount( &quot;Insured&quot; )</td>
<td>0</td>
<td>The Insured recipient does not support Addressees; therefore the returned value is zero (0).</td>
</tr>
</tbody>
</table>

**See also**

- GetAddresseeValues on page 250
- HaveRecip on page 274
- Have Functions on page 69
**AFELog**

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

**Syntax**

```
AFELog (String)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Enter a valid string.</td>
</tr>
</tbody>
</table>

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:

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

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

*Input a custom message to be written to the AFELOG file*

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

**See also**

WIP Functions on page 88
**Always**

Use this function to return TRUE (Always).

**Syntax**

```
Always()
```

There are no parameters for this function.

This function is typically used as a placeholder or stub.

**Example**

```
Always()
```

**See also** [Miscellaneous Functions on page 73](#)
The Append function is obsolete and is no longer supported. Use one of these functions instead:

<table>
<thead>
<tr>
<th>To</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Append text into a multi-line field from an external multi-line text area.</td>
<td>AppendTxm</td>
</tr>
<tr>
<td>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.</td>
<td>AppendTxmUnique</td>
</tr>
</tbody>
</table>

See also [Field Functions on page 61](#)
**APPENDTEXT**

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

### Syntax

```plaintext
APPENDTEXT (File, Field, Section, Form, Group)
```

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

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

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

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

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

### Example

Here are some examples:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPENDTEXT (&quot;MyFile.txt&quot;)</td>
<td>1 or 0</td>
<td>The current field receives the text from the file named, MyFile.Txt. The named file does not specify a path, therefore the system tries to locate the file where form sections are normally located.</td>
</tr>
<tr>
<td>APPENDTEXT (&quot;C:\MyFile.txt&quot;, &quot;MyField&quot;)</td>
<td>1 or 0</td>
<td>MyField will be located on the current section. If found, the field receives the text from the file named, MyFile.Txt. The named file specifies a path, therefore the system looks for the file in that location.</td>
</tr>
<tr>
<td>APPENDTEXT (&quot;MyFile.txt&quot;, &quot;MyField&quot;,&quot;MyField&quot;,&quot;MyForm&quot;)</td>
<td>1 or 0</td>
<td>The field, MyField, will be located on the form, MyForm. Since a section was not specified, it may occur on any section on that form. Once located, the text from the specified file is appended to the field.</td>
</tr>
</tbody>
</table>

### See also

- Field Functions on page 61
- Field Formats on page 62
Locating Fields on page 64
Refresh on page 357
**APPENDTXM**

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

**Syntax**

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

**Parameter Description**

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

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

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

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

Here are some examples:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>#rc = AppendTxm(&quot;Message&quot;, &quot;Name_Line&quot;); Refresh ();</td>
<td>1 or 0</td>
<td>The text in the first text area on the section named Message is appended to the multi-line text field, called Name_Line. The system then refreshes the display.</td>
</tr>
<tr>
<td>#rc = AppendTxm(&quot;.\mstrres\messages\msg1&quot;, &quot;Name_Line&quot;, &quot;Mailer&quot;); Refresh ();</td>
<td>1 or 0</td>
<td>The path, .\mstrres\messages\msg1, is appended to the multi-line text field, called Name_Line, which is on the section named Mailer. The system then refreshes the display.</td>
</tr>
<tr>
<td>#rc = AppendTxm(&quot;message&quot;, &quot;Address1&quot;, &quot;Name_Line&quot;); Refresh ();</td>
<td>1 or 0</td>
<td>The fields in the text area are inserted before the variable field named Address1, in the tabbing sequence. The system then refreshes the display.</td>
</tr>
</tbody>
</table>

See also

Field Functions on page 61
Field Formats on page 62
Locating Fields on page 64
AppendTxmUnique on page 136
Refresh on page 357
**APPENDTXMUNIQUE**

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

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

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

This procedure only works on multi-line text fields.

**Syntax**  
```plaintext
APPENDTXMUNIQUE (FAP, InsertFld, Field, Section, Form, Group)
```

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

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

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

The system locates the section in the directory where sections (FAP files) are typically found. Use the Refresh procedure to make sure all appended text appears in the field.
This procedure is similar to the AppendTxm procedure. For instance, suppose your paragraph section looks like this, where X1 is a reference to the Name field and X2 is a reference to the City field.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Here are some examples:

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

See also

- Field Functions on page 61
- Locating Objects on page 94
- Field Formats on page 62
- Locating Fields on page 64
- AppendTxm on page 134
- Refresh on page 357
**AppIdxRec**

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

**Syntax**

```plaintext
AppIdxRec ()
```

There are no parameters for this function.

**Example**

Here are some examples. Assume that...

- The rundate is 01/10/2009
- The sub-string of extract record being processed is:
  
  SCO1234567HEADERREC00000...

- The FORM.DAT file contains the following:
  
  ;SAMPCO;LB1;Libby;;R;;letter|D<INSURED(1),COMPANY(1),AGENT(1)>;

Also assume these INI options exist:

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comment = AppIdxRec()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Print_It ( Comment )</td>
<td>1 or 0</td>
<td></td>
</tr>
</tbody>
</table>

**See also**

- [Documaker Server Functions on page 58](#)
- [Print_It on page 342](#)
- [DAL Script Examples on page 35](#)
**APPLYINSERTS**

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

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

**Syntax**

```
ApplyInserts()
```

There are no parameters for this procedure.

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

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

**Example**

Here is an example:

```
ApplyInserts()
```

**See also** Section Functions on page 77
**Ask**

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

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

**Syntax**

\[ \text{Ask (Msgline1, Msgline2, Msgline3, Title, Defans)} \]

**Parameter Description**

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

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:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>#result = Ask (&quot;Are you sure you made the correct entry?&quot;, , &quot;Sample Message&quot;)</td>
<td>1, if the user answers Yes 0, if the user answers No</td>
<td>“Sample Message” is the title of the entry box. “Are you sure you made the correct entry?” is the message the user answers.</td>
</tr>
<tr>
<td>#result = Ask (&quot;This is line 1&quot;, &quot;This is line 2&quot;, &quot;Is this line 3?&quot;, &quot;Please Respond&quot;)</td>
<td>1, if the user answers Yes 0, if the user answers No</td>
<td>“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.</td>
</tr>
</tbody>
</table>

**See also** Documaker Workstation Functions on page 59
ASSIGNWIP

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

Syntax

\texttt{ASSIGNWIP (UserID)}

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserID</td>
<td>Enter a valid user ID.</td>
</tr>
</tbody>
</table>

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

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

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

Example

Here is an example:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssignWIP (MVV)</td>
<td>User ID for the work-in-process is changed to \textit{MVV}, the form set is saved in the WIP directory, and you return to the main menu.</td>
<td>The user ID in the WIP database is set to \textit{MVV} for the current WIP.</td>
</tr>
<tr>
<td>AssignWIP ()</td>
<td>The Assign window appears. Use this window to make the assignment.</td>
<td>If you omit the user ID, the system instead displays the Assign window, just as if you had selected the Formset, Assign Document option.</td>
</tr>
</tbody>
</table>

See also

- WIP Functions on page 88
- Documaker Workstation Administration Guide
- Documaker Workstation User Guide
**Avg**

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

**Syntax**

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

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

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

**Example**

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

- Myfield1
- Myfield2
- Myfield20

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

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

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

**Example**

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Section</th>
<th>Form</th>
<th>Group</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyField1</td>
<td>IMG A</td>
<td>FRM A</td>
<td>GRP</td>
<td>100.24</td>
</tr>
<tr>
<td>MyField2</td>
<td>IMG A</td>
<td>FRM A</td>
<td>GRP</td>
<td>200.16</td>
</tr>
</tbody>
</table>
Assume the current field is MyField1, on the first section of the first form. Reference the previous table for field values.

<table>
<thead>
<tr>
<th>Field</th>
<th>Section</th>
<th>Form</th>
<th>Group</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyField1</td>
<td>IMG B</td>
<td>FRM A</td>
<td>GRP</td>
<td>98.60</td>
</tr>
<tr>
<td>MyField2</td>
<td>IMG B</td>
<td>FRM A</td>
<td>GRP</td>
<td>* no value yet *</td>
</tr>
<tr>
<td>MyField1</td>
<td>IMG A</td>
<td>FRM B</td>
<td>GRP</td>
<td>0.00</td>
</tr>
<tr>
<td>MyField2</td>
<td>IMG A</td>
<td>FRM B</td>
<td>GRP</td>
<td>* no value yet *</td>
</tr>
<tr>
<td>MyField1</td>
<td>IMG B</td>
<td>FRM B</td>
<td>GRP</td>
<td>70.77</td>
</tr>
<tr>
<td>MyField2</td>
<td>IMG B</td>
<td>FRM B</td>
<td>GRP</td>
<td>* no value yet *</td>
</tr>
</tbody>
</table>

Function Result Explanation

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

See also
Mathematical Functions on page 72
Field Formats on page 62
Locating Fields on page 64
**BankRound**

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

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

**Syntax**

```
BankRound(Value)
```

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

**Example**

Here are some examples that compare BankRound with Round:

<table>
<thead>
<tr>
<th>With BankRound</th>
<th>Returns</th>
<th>Whereas, with Round</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>BankRound(123.425)</td>
<td>123.42</td>
<td>Round(123.425)</td>
<td>123.43</td>
</tr>
<tr>
<td>BankRound(123.435)</td>
<td>123.44</td>
<td>Round(123.435)</td>
<td>123.44</td>
</tr>
</tbody>
</table>

**See also**

- String Functions on page 78
- Round on page 365
**Beep**

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

### Syntax

```
Beep (Integer)
```

### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>Choose from these options:</td>
</tr>
<tr>
<td></td>
<td>0 - Warning sound</td>
</tr>
<tr>
<td></td>
<td>1 - Message sound</td>
</tr>
<tr>
<td></td>
<td>2 - Error sound</td>
</tr>
<tr>
<td></td>
<td>The default is two (2).</td>
</tr>
</tbody>
</table>

This procedure emits the sound specified by the parameter.

### Example

Here are some examples:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beep ()</td>
<td>Emits error sound.</td>
<td>Defaults to 2.</td>
</tr>
<tr>
<td>Beep (0)</td>
<td>Emits warning sound.</td>
<td>The operating system emits the installed option for the warning sound.</td>
</tr>
</tbody>
</table>

See also  [Documaker Workstation Functions on page 59](#)
**BitAnd**

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

**Syntax**

```plaintext
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:

<table>
<thead>
<tr>
<th>Value1 bit</th>
<th>Value2 bit</th>
<th>Result bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Example**

Here is an example:

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

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

**See also**

- BitClear on page 148
- BitNot on page 149
- BitOr on page 150
- BitRotate on page 151
- BitSet on page 153
- BitShift on page 154
- BitTest on page 154
- BitXor on page 157
- Bit/Binary Functions on page 42
**BitClear**

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

**Syntax**

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

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

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

![Binary representation](image)

**Example**

Here is an example:

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

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

**See also**

BitAnd on page 147
BitNot on page 149
BitOr on page 150
BitRotate on page 151
BitSet on page 153
BitShift on page 154
BitTest on page 156
BitXor on page 157
Bit/Binary Functions on page 42
**BitNot**

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

**Syntax**

```
BitNot(value1)
```

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

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

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

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

<table>
<thead>
<tr>
<th>Value1 bit</th>
<th>Result bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Example**

Here is an example:

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

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

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

**See also**

- BitAnd on page 147
- BitClear on page 148
- BitOr on page 150
- BitRotate on page 151
- BitSet on page 153
- BitShift on page 154
- BitTest on page 156
- BitXor on page 157
- Bit/Binary Functions on page 42
**BitOr**

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

**Syntax**

```plaintext
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:

<table>
<thead>
<tr>
<th>Value1 bit</th>
<th>Value2 bit</th>
<th>Result bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Example**

Here is an example:

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

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

**See also**

- BitAnd on page 147
- BitClear on page 148
- BitNot on page 149
- BitRotate on page 151
- BitSet on page 153
- BitShift on page 154
- BitTest on page 156
- BitXor on page 157
- Bit/Binary Functions on page 42
**BitRotate**

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

**Syntax**

\[ \text{BitRotate}(\text{value1, shiftAmt}) \]

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>Value1 bits</th>
<th>Shift</th>
<th>Result value bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 (0110)</td>
<td>1</td>
<td>12 (1100)</td>
</tr>
<tr>
<td>6 (0110)</td>
<td>2</td>
<td>24 (0001 1000)</td>
</tr>
<tr>
<td>6 (0110)</td>
<td>3</td>
<td>48 (0011 0000)</td>
</tr>
<tr>
<td>6 (0110)</td>
<td>4</td>
<td>96 (0110 0000)</td>
</tr>
<tr>
<td>6 (0110)</td>
<td>-1</td>
<td>3 (0011)</td>
</tr>
<tr>
<td>6 (0110)</td>
<td>-2</td>
<td>-2147483647 (1000 0000 0000 0000 0000 0000 0000 0001)</td>
</tr>
<tr>
<td>6 (0110)</td>
<td>-3</td>
<td>-1073741824 (1100 0000 0000 0000 0000 0000 0000 0000)</td>
</tr>
<tr>
<td>6 (0110)</td>
<td>-4</td>
<td>1610612736 (0110 0000 0000 0000 0000 0000 0000 0000)</td>
</tr>
</tbody>
</table>

**Example**

Here is an example:

\[
\begin{align*}
  z &= \text{BitRotate}(6, -8) \\
  z &= 100663296 (0000 0110 0000 0000 0000 0000 0000 0000)
\end{align*}
\]

**See also**

BitAnd on page 147

BitClear on page 148
BitNot on page 149
BitOr on page 150
BitSet on page 153
BitShift on page 154
BitTest on page 156
BitXor on page 157
Bit/Binary Functions on page 42
**BitSet**

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

**Syntax**

`BitSet(Value1, BitPos)`

The parameters specify the numeric value and the bit position on which the operation is performed. The specified bit is set to a 1 in the value provided. If the bit was already on, the value is unchanged. Specifying a negative or zero bit position does not result in any change to the value.

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

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

**Example**

Here is an example:

```
y = 6  (6 is 0110 in binary)
z = BitSet(x, 1)
z = 7  (7 is 0111 in binary)

y = 6  (6 is 0110 in binary)
z = BitSet(x, 4)
z = 15 (15 is 1110 in binary)
```

**See also**

- BitAnd on page 147
- BitClear on page 148
- BitNot on page 149
- BitOr on page 150
- BitRotate on page 151
- BitShift on page 154
- BitTest on page 156
- BitXor on page 157
- Bit/Binary Functions on page 42
**BitShift**

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

**Syntax**

\[ \text{BitShift}(\text{Value1}, \text{ShiftAmt}) \]

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

This is a logical shift, as opposed to a shift-and-rotate operation. This means bits shifted off the end of a value are considered lost.

**NOTE:** See the BitRotate on page 151 function for shift-and-rotate.

A positive `ShiftAmt` value causes the bit pattern in `Value1` to be shifted left the number of bits specified by `ShiftAmt`. Bits vacated by the shift operation are zero-filled.

A negative `ShiftAmt` value causes the bit pattern in `Value1` to be shifted right the number of bits specified by `ShiftAmt`. Bits vacated by the shift operation are zero-filled.

Note that integer values have 32 bits. Attempting to shift more than 31 bit positions will result in a zero (0) being returned, as all bits are cleared.

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

<table>
<thead>
<tr>
<th>Value1 bits</th>
<th>Shift</th>
<th>Result value bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 (0110)</td>
<td>1</td>
<td>12 (1100)</td>
</tr>
<tr>
<td>6 (0110)</td>
<td>2</td>
<td>24 (0001 1000)</td>
</tr>
<tr>
<td>6 (0110)</td>
<td>3</td>
<td>48 (0011 0000)</td>
</tr>
<tr>
<td>6 (0110)</td>
<td>4</td>
<td>96 (0110 0000)</td>
</tr>
<tr>
<td>6 (0110)</td>
<td>-1</td>
<td>3 (0011)</td>
</tr>
<tr>
<td>6 (0110)</td>
<td>-2</td>
<td>1 (0001)</td>
</tr>
<tr>
<td>6 (0110)</td>
<td>-3</td>
<td>0 (0000)</td>
</tr>
<tr>
<td>6 (0110)</td>
<td>-4</td>
<td>0 (0000)</td>
</tr>
</tbody>
</table>

**Example**

Here is an example:

\[ z = \text{BitShift}(6, 8) \]
\[ z = 1536 \quad (1536 \text{ is } 0110 \ 0000 \ 0000 \text{ in binary}) \]

**See also**

- BitAnd on page 147
- BitClear on page 148
- BitNot on page 149
BitOr on page 150
BitRotate on page 151
BitSet on page 153
BitTest on page 156
BitXor on page 157
Bit/Binary Functions on page 42
**BitTest**

Use this function to return TRUE (1) if the specified bit in a value is a 1; otherwise return FALSE (0).

**Syntax**

```
BitTest(Value1, BitPos)
```

Parameters specify the numeric value and the bit position on which the operation is performed. The specified bit is tested for a 1 value. If the bit is a 1, then 1 is returned. If the bit is zero (0), then zero (0) is returned. Specifying a negative or zero bit position will result in zero (0) being returned.

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

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

**Example**

Here is an example:

```
y = 6  (6 is 0110 in binary)
z = BitTest(x,1)
z = 0  (bit 1 was not on)
y = 6  (6 is 0110 in binary)
z = BitTest(x,2)
z = 1  (bit 2 was on)
```

**See also**

BitAnd on page 147  
BitClear on page 148  
BitNot on page 149  
BitOr on page 150  
BitRotate on page 151  
BitSet on page 153  
BitShift on page 154  
BitXor on page 157  
Bit/Binary Functions on page 42
**BitXor**

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

**Syntax**

```
BitXor(Value1, Value2)
```

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

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

The following table shows the result of a bitwise XOR (exclusive OR) operation:

<table>
<thead>
<tr>
<th>Value1 bit</th>
<th>Value2 bit</th>
<th>Result bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Example**

Here is an example:

\[
x = 3 \quad (3 \text{ is 0011 in binary})
y = 6 \quad (6 \text{ is 0110 in binary})
\]

\[
z = \text{BitXor}(x,y)
z = 5 \quad (5 \text{ is 0101 in binary})
\]

**See also**

- BitAnd on page 147
- BitClear on page 148
- BitNot on page 149
- BitOr on page 150
- BitRotate on page 151
- BitSet on page 153
- BitShift on page 154
- BitTest on page 156
- Bit/Binary Functions on page 42
**BreakBatch**

Use this function to tell the Documaker Server to break the output print stream file for the current recipient batch after processing the current recipient, including post transaction banner processing.

**Syntax**

```
BreakBatch()
```

This procedure is typically called in the transaction banner DAL script. You must use the SetDeviceName function to specify a new device name. Otherwise, the new file has the same name as the old file and overwrites its contents.

After the GenPrint program finishes processing the current transaction, it closes the current output device file. This includes executing any post-batch banner processes. It then continues processing the recipient batch.

If you have assigned a new output device file name using the SetDeviceName function, the system will create and start writing to a new print stream file with that name. The best place to call the BreakBatch function is in the post-transaction banner DAL script.

Here is an example of DAL script logic that might appear in a post-transaction banner DAL script. This example requires that a pre-transaction banner DAL script save the current recipient name in a variable called `CurrRecip`, as shown here:

```
CurrRecip = RecipName()
```

The post-transaction banner DAL script would then include the following:

```
IF TotalSheets(CurrRecip) > 16000
  #COUNTER += 1
  CurFile = DeviceName()
  Drive = FileDrive(CurFile)
  Path = FilePath(CurFile)
  Ext = FileExt(CurFile)
  RecipBatch = RecipBatch()
  NewFile = FullFileName(Drive,Path,RecipBatch & #COUNTER,Ext)
  SetDeviceName(NewFile)
  BreakBatch()
END
```

**NOTE:** See FileDrive, FileExt, FileName, FilePath, and FullFileName for information on using DAL functions to manipulate file names.

**Keep in mind...**

- These print drivers are supported: AFP, MET, PCL5, PCL6, and PST.
- These print drivers are not supported: EPT, GDI, HTML, PDF, RTF, and XML.
- All platforms are supported, but note that while UniqueString is supported on z/OS, z/OS does not support long file names.
- Both multi-step and single-step processing are supported.
The only DAL function actually involved in splitting the print stream is BreakBatch. The others make it easier to implement this functionality. For example, since you need to name the new print stream, you use the SetDeviceName procedure. To find the name of the current device, you use the DeviceName function. If you need to create unique file names, you can use the UniqueString function.

**NOTE:** The BreakBatch and SetDeviceName functions are not applicable in Entry since it does not use the batch printing engine. The other functions, DeviceName and UniqueString, are applicable to both Entry and Documaker Server.

See also  
- Printer and Recipient Functions on page 76  
- DeviceName on page 217  
- SetDeviceName on page 373  
- UniqueString on page 421
**CALL**

Use this function to temporarily suspend one calculation and execute another calculation file. A CALL statement must begin with `CALL`.

**Syntax**

```
CALL (File)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>Enter the name of the calculation file you want the system to execute.</td>
</tr>
</tbody>
</table>

The calculation file that is called must contain a RETURN statement if the original calculation expects a returned value.

**Example**

Here is an example:

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

This tells the system to call the calculation file TestCalc. After the calculations in TestCalc are completed, processing returns to the current script. In this example, TestCalc is not expected to return a value.

**See also** [Miscellaneous Functions on page 73](#)
**CHAIN**

Use this function to calls another calculation language file. A Chain statement must begin with *CHAIN*. There is no limit to the number of Chain statements you can use.

**Syntax**

CHAIN (Script)

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Script</td>
<td>Enter the name of the DAL script file. You can omit the extension.</td>
</tr>
</tbody>
</table>

**Example**

Here are some examples:

CHAIN 'LastCalc'

or

CHAIN('LastCalc')

These examples permanently call the calculation file named LastCalc. Processing does not return to the current script. No statements from the original script will be evaluated after the Chain statement.

**See also** [Miscellaneous Functions on page 73](#)
**CFind**

Use this function to search a text string and return the first position of any character found within a specified set of characters. The search is not case sensitive.

**Syntax**

```
CFind (String, Charset, Integer)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Enter a valid string. This is the string that is searched. The default is the value of the current field text.</td>
</tr>
<tr>
<td>Charset</td>
<td>Enter a set of one or more characters, any of which may be found in the target string.</td>
</tr>
<tr>
<td>Integer</td>
<td>Enter zero (0) for a left to right search. Enter one (1) for a right to left search. The default is zero (0).</td>
</tr>
</tbody>
</table>

The system returns a zero (0) if none of the search characters are found in the text string. The default search order is left to right. You can also specify a right to left search order. Both search methods returns the position relative to the first (left-hand) character of the string parameter.

**Example**

Here are some examples:

(Assume the current field contains the text *Your Name.*)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return(CFind (&quot;This is the answer&quot;, &quot;ws&quot;))</td>
<td>4</td>
<td>Searching from left to right, s was first found at position 4.</td>
</tr>
<tr>
<td>Return(CFind (&quot;This is the answer&quot;, &quot;ws&quot;, 1))</td>
<td>16</td>
<td>Searching from right to left, w was first found at position 16.</td>
</tr>
<tr>
<td>Return(CFind (&quot;n&quot;))</td>
<td>6</td>
<td>The first occurrence of an n in the current field <em>Your Name</em> is at position 6. Note the search is not case sensitive.</td>
</tr>
<tr>
<td>Return(CFind (&quot;xz&quot;))</td>
<td>0</td>
<td>Neither x nor z is contained in the current text field.</td>
</tr>
</tbody>
</table>

**See also** String Functions on page 78
**CHANGELOG**

Use this procedure/function to replace a bitmap graphic (LOG file) on a section with a different graphic.

**Syntax**

```
ChangeLogo (LOGFile, Graphic, Section, Form, Group)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOGFile</td>
<td>Enter the name of a file that contains a valid graphic.</td>
</tr>
<tr>
<td>Graphic</td>
<td>Enter the name of the current graphic in a section.</td>
</tr>
<tr>
<td>Section</td>
<td>Enter the name of a section that contains the graphic you specified. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of a form that contains the section or graphic. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of a group to use to locate the object. The default is the current group.</td>
</tr>
</tbody>
</table>

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

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

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

The Graphic parameter tells the system to look for the name that appears in the Name field on the Graphic Options in Studio or Image Editor. If there is no entry in this field, this procedure will not work correctly.

**NOTE:** When you use this procedure with Documaker Workstation, you must follow this procedure with the Refresh procedure. The ChangeLogo procedure does not redraw the section after it changes the graphic.

When you use the ChangeLogo procedure with Documaker, you must include the CheckImageLoaded rule as one of the section level rules for the section or else set the LoadCordFAP option in the RunMode control group to Yes in your FSISYS.INI file.
Here are some examples:

(Assume the section has a graphic named `sign`.)

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ChangeLogo (&quot;johndoe&quot;, &quot;sign&quot;)</code></td>
<td>1 or 0</td>
<td>Replaces the existing graphic contained by <code>sign</code> with a new graphic.</td>
</tr>
<tr>
<td><code>ChangeLogo (, &quot;sign&quot;, &quot;IMG&quot;)</code></td>
<td>1 or 0</td>
<td>Locate the specified section on the current form. If found replace the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>existing graphic contained by <code>sign</code> with an empty graphic.</td>
</tr>
</tbody>
</table>

See also

- DelLogo on page 214
- HaveLogo on page 272
- InlineLogo on page 282
- RenameLogo on page 359
- Logo on page 303
- Refresh on page 357
- Graphics Functions on page 71
**Char**

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

**Syntax**

```
Char (Integer)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>An integer value that ranges zero (0) to 255.</td>
</tr>
</tbody>
</table>

**Example**

Here is an example:

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

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

**See also**

- CharV on page 166
- String Functions on page 78
**CharV**

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

**Syntax**

```
CharV (String)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>A character string. If the string contains more than one character, only the first character is converted. The remaining characters are ignored.</td>
</tr>
</tbody>
</table>

**Example**

In this example, assume the variable, `char_to_convert`, contains the single character: “@”.

```python
#_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: “@( )”.

```python
#_the_integer = CharV(the_string)
```

The integer variable, `_the_integer`, is set the value: 64. The remaining characters are ignored.

**See also**

- Char on page 165
- String Functions on page 78
**CODEINLIST**

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

**Syntax**

CodeInList (String, List)

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Enter the string you want to search for. Keep in mind the system considers spaces when matching strings and that the strings must match exactly.</td>
</tr>
<tr>
<td>List</td>
<td>Enter the name of the list of strings. Use commas to separate each string entry you want to search for.</td>
</tr>
</tbody>
</table>

The function returns a number that indicates which string entry was found. For instance, if the third string entry was found, the function returns a three (3).

**Example**

Here is an example:

- CodeInList( "ABC",  "ABC,AB,DE,A,GFHI,ABCD" )returns 1
- CodeInList( "AB",   "ABC,AB,DE,A,GFHI,ABCD" )returns 2
- CodeInList( "DE",   "ABC,AB,DE,A,GFHI,ABCD" )returns 3
- CodeInList( "A",    "ABC,AB,DE,A,GFHI,ABCD" )returns 4
- CodeInList( "GFHI", "ABC,AB,DE,A,GFHI,ABCD" )returns 5
- CodeInList( "ABCD", "ABC,AB,DE,A,GFHI,ABCD" )returns 6
- CodeInList( "XYZ",  "ABC,AB,DE,A,GFHI,ABCD" )returns 0
- CodeInList( ",",    "ABC,AB,DE,A,GFHI,ABCD" )returns 0
- CodeInList( "ABC",  "" ) returns 0
- CodeInList( ",",    "" ) returns 1

If you omit the first parameter, you get the data from the current field. If you omit the second parameter, you receive this error message:

Wrong number of parameters

Here is another example:

Assume that GetValue(col_name1) results in the string: EE. And the variable col_name1_codes contains the string: EEacb,XXEE,EE,AEEAC.

#rc = CodeInList( GetValue(col_name1), col_name1_codes ) returns 3

Keep in mind...

- The search *is not* case sensitive. This means that A will match a.
- Spaces *are* considered. This means the system will find no matches in these examples:

  - CodeInList("Steel",  " Steel,Aluminum")
  - CodeInList("Steel",  "Steel ,Aluminum")
  - CodeInList("Steel",  "Aluminum,Steel ")

  and will return zero (0) each time.

See also  **String Functions on page 78**
**COMPLETE**

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

**Syntax**

```
Complete (PrintFlag, ExportFlag, ExportType, ExportFile)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrintFlag</td>
<td>Indicates whether the system should print the form set. The default is No.</td>
</tr>
<tr>
<td>ExportFlag</td>
<td>Indicates whether the system should export the work-in-process data to a file. The default is No.</td>
</tr>
<tr>
<td>ExportType</td>
<td>TD, SI, and so on. Indicates the type of export file. The default is TD.</td>
</tr>
<tr>
<td>Exportfile</td>
<td>The file name for the Standard Export file, if specified in the INI options.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete ()</td>
<td>Completes the work-in-process.</td>
<td>Performs the processes as specified by archive INI options.</td>
</tr>
</tbody>
</table>
| Complete
(EXPORT.TXT) | Completes the work-in-process and writes the data to a file named EXPORT.TXT. | Performs the processes as specified by archive INI options and writes the data to a file named EXPORT.TXT. |

**See also**

- WIP Functions on page 88
- Documaker Workstation Administration Guide
- Documaker Workstation User Guide
**COMPRESSIONFLDS**

Use this function/procedure to compress blank space by moving field data. This function moves field data from one field to a prior named field to compress the space between the fields. Typically you use this function to compress vertical space, as in address lines, but the fields do not have to be vertical relative to each other. You can compress any field.

**NOTE:** The data moves between the fields; the actual location of each physical field remains the same.

CompressFLDS can be used as a procedure or as a function.

**Syntax**

```
CompressFLDS (FieldList, Section, Form, Group)
```

**Parameter** | **Description**
--- | ---
FieldList | Enter a list of the fields you want to compress, separated by commas. Here is an example:
FIELD1, FIELD2, FIELD3
Section | (Optional) Enter the name of a section that contains the fields you specified. The default is the current section.
Form | (Optional) Enter the name of a form that contains the section and/or field you specified. The default is the current form.
Group | (Optional) Enter the name of the form group that contains the form, section, or fields. The default is the current group.

**NOTE:** When using this function in Documaker Server processing, make sure the fields exist on the section. Some implementations that use versions of the system prior to version 11.0 do not load FAP files in all cases, and fields will not be created when data mapping did not place any data into the field.

Keep in mind...

- Each subsequent field with data is mapped into the first available empty field which you included in the list.
- Fields are defined in FAP sections with a tabbing order. This tabbing order typically matches the order in which field level rules are processed during Documaker Server processing. Unlike the SetAddr rules, the CompressFLDS function can compress fields in any order, and the field spaces do not have to be compressed up following the tabbing order.
- The last movement of that field determines the final location of a given field’s data.
- Always specify a set of unique field names. Do not attempt to name a field more than once within a field list as this can produce unpredictable results.
- This function does not work with bar code or multi-line text fields.
Example

For this example, assume the following fields and data:

<table>
<thead>
<tr>
<th>This field</th>
<th>Contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIELD_A</td>
<td>ABCDEFG</td>
</tr>
<tr>
<td>FIELD_B</td>
<td>is empty</td>
</tr>
<tr>
<td>FIELD_C</td>
<td>is empty</td>
</tr>
<tr>
<td>FIELD_D</td>
<td>TUVWXYZ</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>This field</th>
<th>Contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIELD_A</td>
<td>ABCDEFG</td>
</tr>
<tr>
<td>FIELD_B</td>
<td>TUVWXYZ</td>
</tr>
<tr>
<td>FIELD_C</td>
<td>is empty</td>
</tr>
<tr>
<td>FIELD_D</td>
<td>is empty</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>This field</th>
<th>Contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIELD_A</td>
<td>is empty</td>
</tr>
<tr>
<td>FIELD_B</td>
<td>is empty</td>
</tr>
<tr>
<td>FIELD_C</td>
<td>ABCDEFG</td>
</tr>
<tr>
<td>FIELD_D</td>
<td>TUVWXYZ</td>
</tr>
</tbody>
</table>

See also Field Functions on page 61
**CONNECTFLDS**

Use this function/procedure to move fields (change field coordinates) in such a way as to make the field’s text appear to be concatenated. This function does not literally concatenate the fields but instead repositions and aligns field text along a common horizontal coordinate so the field’s data appears concatenated. It does not move fields vertically.

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

**Syntax**

```
ConnectFlds (FieldList, Section, Form, Group)
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FieldList</td>
<td>A list of the fields you want to connect, preceded by a movement flag and separated with commas. Here is an example: &quot;FIELD1, FIELD2, FIELD3&quot; If a field name is not preceded by a movement flag or if it is preceded by the F movement flag, which indicates it is a fixed field, the field is not moved. The first field you name in the parameter must be a fixed field. The rest of the field names in your list indicate fields you want moved adjacent to the fixed field. Each field you name is moved according to the use described by the movement flag that precedes its name.</td>
</tr>
<tr>
<td>Section</td>
<td>(Optional) Enter the name of a section that contains the fields you specified. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>(Optional) Enter the name of a form that contains the section and/or field you specified. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>(Optional) Enter the name of the form group that contains the form, section, or fields. The default is the current group.</td>
</tr>
</tbody>
</table>

In the FieldList parameter you must specify a fixed field and at least one field to move (visually concatenate) to the left or right side of the fixed field. You can specify multiple fields to move.

**NOTE:** This function does not move fields vertically. Fields are only moved horizontally. You should set the vertical alignment of fields when you create the section.

By default, each concatenation will be placed the distance of one space character from the fixed field, unless the parameter indicates otherwise. You can include these movement flags in the FieldList parameter:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>Tells the system to move the specified field so it appears to be appended to the left of the fixed field.</td>
</tr>
<tr>
<td>R</td>
<td>Tells the system to append the specified field to the right of the fixed field.</td>
</tr>
</tbody>
</table>
Here is an example:

"F=FIELD1,RNO=FIELD2"

Here, the contents of FIELD2 are placed immediately adjacent to the end of the contents of FIELD1 without an intervening space.

Keep in mind...

• As each field is appended to the fixed field, the fixed rectangle grows. By growing the fixed rectangle, additional fields that append move based upon where the prior appended field ended.

• If a field specified for appending does not contain any data or is not valid, then no space, or space holder, is included in the concatenation.

• If a field contains centered or right justified data padded with spaces then the results may appear to be incorrect. This function calculates the width of a field based upon the entire contents and will not remove spaces, or any other white space characters, in the fields.

• Naming a field to move more than once in the first parameter can cause unpredictable results.

• The last movement of a field will determine the final location of a field’s data.

• During any movement operation, the field being moved cannot also be named as the fixed field.

• This function does not work with bar code or multi-line text fields.

• This function does not handle rotated fields.

Example

For the following examples, make these assumptions:

<table>
<thead>
<tr>
<th>This field</th>
<th>Contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIELD1</td>
<td>ABC</td>
</tr>
<tr>
<td>FIELD2</td>
<td>DEF</td>
</tr>
<tr>
<td>FIELD3</td>
<td>XYZ</td>
</tr>
</tbody>
</table>

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:

```sql
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:

```sql
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:

```sql
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:

```sql
ConnectFlds("F=FIELD1,R=FIELD2,R=FIELD2")
```

You get this result:

```
ABC     DEF
```

In this case, FIELD2 is defined to move twice. Since the operations are sequential, the field first moved adjacent to FIELD1. This movement expanded the fixed rectangle used by subsequent movements. When the field was named again, it moved relative to the newly expanded rectangle, resulting in the field appearing farther to the right, a distance equal to the size of the text in the field plus the width of two spaces.

See also [Field Functions on page 61](#)
**COPYFORM**

Use this procedure/function to locate a form and copy that form and its field contents (data) into a new form. With this procedure, you can also specify another form and group as the insertion point for the new form.

**NOTE:** When you use the AddForm procedure, the only data duplicated is the global data that propagates into the fields. When you use the DupForm procedure, only those forms with the Multicopy option checked can be duplicated. With the CopyForm procedure, any form within the document can be copied.

**Syntax**

CopyForm (Form, Group, InsAtForm, InsAtGroup)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form</td>
<td>Enter the name of the form you want to copy</td>
</tr>
<tr>
<td>Group</td>
<td>(Optional) Enter the name of the group if the form is not in the current group.</td>
</tr>
<tr>
<td>InsAtForm</td>
<td>Enter the name of the form after which you want the system to insert the form it copies.</td>
</tr>
<tr>
<td>InsAtGroup</td>
<td>(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.</td>
</tr>
</tbody>
</table>

If you do not specify an insertion point, the system appends the new form to the end of the form group of the original form.

If the procedure is successful in copying the form, it returns a non-zero value, otherwise zero (0) is returned. This procedure can fail for these reasons:

- Could not locate the form or form group specified
- Lack of available memory

You can use this procedure in scripts hosted by AFEMain or other Entry-related applications and also in batch applications using the GenData program.

**See also**

AddForm on page 119
AddForm_Propagate on page 120
DupForm on page 228
TriggerForm on page 414
WIP Functions on page 88
**COUNT**

Use this function to count the number of fields that have values and have names that begin with common characters. The result of the operation is returned.

**Syntax**

```
Count (PartialField, Section, Form, Group)
```

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

This function returns the number of fields that have values that begin with the specified partial field name.

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

```
Myfield1
Myfield2
Myfield20
```

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

Note that zero (0) is a valid field value. A field that has never been given a value is excluded from the count.

**NOTE:** As a general rule, include the PartialField parameter. Fields in a section must have unique names. Using the default will probably not give the expected result, unless you created the form and understand the naming conventions.

**Example**

Here are some examples:

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Section</th>
<th>Form</th>
<th>Group</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyField1</td>
<td>IMG A</td>
<td>FRM A</td>
<td>GRP</td>
<td>100.24</td>
</tr>
<tr>
<td>MyField2</td>
<td>IMG A</td>
<td>FRM A</td>
<td>GRP</td>
<td>200.16</td>
</tr>
<tr>
<td>Field</td>
<td>Section</td>
<td>Form</td>
<td>Group</td>
<td>Value</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>-------</td>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>MyField1</td>
<td>IMG B</td>
<td>FRM A</td>
<td>GRP</td>
<td>98.60</td>
</tr>
<tr>
<td>MyField2</td>
<td>IMG B</td>
<td>FRM A</td>
<td>GRP</td>
<td>* no value yet *</td>
</tr>
<tr>
<td>MyField1</td>
<td>IMG A</td>
<td>FRM B</td>
<td>GRP</td>
<td>0.00</td>
</tr>
<tr>
<td>MyField2</td>
<td>IMG A</td>
<td>FRM B</td>
<td>GRP</td>
<td>* no value yet *</td>
</tr>
<tr>
<td>MyField1</td>
<td>IMG B</td>
<td>FRM B</td>
<td>GRP</td>
<td>70.77</td>
</tr>
<tr>
<td>MyField2</td>
<td>IMG B</td>
<td>FRM B</td>
<td>GRP</td>
<td>* no value yet *</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count( )</td>
<td>1</td>
<td>Without any other information, the function will assume the current field and section. There will only be one value included in the count.</td>
</tr>
<tr>
<td>Return(Count ( &quot;Myfield2&quot;))</td>
<td>1</td>
<td>Again, there is only one field included in this result.</td>
</tr>
<tr>
<td>Return(Count ( &quot;MyField&quot;))</td>
<td>2</td>
<td>In this example, the current section contains two fields that begin with the name &quot;MyField&quot;.</td>
</tr>
<tr>
<td>Return(Count ( &quot;MyField&quot;, &quot;IMG B&quot;))</td>
<td>1</td>
<td>Although two fields on IMG B have a matching name, only one field actually has a value.</td>
</tr>
<tr>
<td>Return(Count ( &quot;MyField&quot;, &quot;FRM A&quot;))</td>
<td>3</td>
<td>No section is specified in this example, so the entire form is searched. Four fields match the name criteria, but only three have values.</td>
</tr>
<tr>
<td>Return(Count ( &quot;MyField&quot;, &quot;IMG B&quot;, &quot;GRP&quot;))</td>
<td>2</td>
<td>This example specifies a section and group, but no form. There are four fields that match the name criteria, but only two have values.</td>
</tr>
<tr>
<td>Return(Count ( &quot;MyField&quot;, &quot;GRP&quot;))</td>
<td>5</td>
<td>This example names the group without a form or section. Eight fields meet the naming criteria, but only five fields actually have values.</td>
</tr>
</tbody>
</table>

See also
- Mathematical Functions on page 72
- Field Formats on page 62
- Locating Fields on page 64
**CountRec**

Use this function to count the number of records in an extract file transaction that match a search mask parameter. In addition, you can also make sure that at least a minimum number of records match the search mask parameter.

**Syntax**

```
CountRec (SearchMask, MinNumber)
```

**Parameter** | **Description**
--- | ---
SearchMask | The search mask you want to use for the search.
MinNumber | (Optional) Number of records that must exist in the transaction. Set this parameter to 1 if you want to know if a record exists that matches the search mask.

This function returns the total number of records found, the MinNumber of records if they exist, or zero (0) if no records match the search mask or there are less than the MinNumber of records.

**Example**

Lets assume there are five records in a transaction with the following values in the applicable columns.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address1</td>
<td>AA</td>
</tr>
<tr>
<td>Address2</td>
<td>BB</td>
</tr>
<tr>
<td>Address3</td>
<td>BB</td>
</tr>
<tr>
<td>Address4</td>
<td>BB</td>
</tr>
<tr>
<td>Address5</td>
<td>CC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CountRec (&quot;1,Address&quot;)</td>
<td>5</td>
<td>The function returns five (5) because there are five records that match the search mask in the transaction.</td>
</tr>
<tr>
<td>CountRec (&quot;1,Address,31,BB&quot;, 2)</td>
<td>2</td>
<td>The function returns two (2) because there are at least two records that match the search mask in the transaction.</td>
</tr>
<tr>
<td>CountRec (&quot;1,Address&quot;, 6)</td>
<td>0</td>
<td>The function returns zero (0) because there are less than six records in the transaction that match the search mask.</td>
</tr>
<tr>
<td>CountRec (&quot;1,Address,31,AA&quot;, 2)</td>
<td>0</td>
<td>The function returns a zero (0) because there are less than two records that match the search mask.</td>
</tr>
</tbody>
</table>

**See also**

Documaker Server Functions on page 58
Field Formats on page 62
Locating Fields on page 64
**Cut**

Use this function to remove characters from a string at a specified position and return the result.

**Syntax**

\[
\text{Cut (String, Position, Length)}
\]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Enter a valid string. The default is the value of the current field text.</td>
</tr>
<tr>
<td>Position</td>
<td>Enter the position within the first parameter to begin cutting. The default is one (1).</td>
</tr>
<tr>
<td>Length</td>
<td>Enter the length to cut from text. The default is zero (0).</td>
</tr>
</tbody>
</table>

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.*)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return(Cut ( ))</td>
<td>Your Name</td>
<td>No length is specified for the cut function; therefore the field remains the same.</td>
</tr>
<tr>
<td>Return(Cut ( , , 5))</td>
<td>Name</td>
<td>Five characters are cut from the current field beginning at position 1.</td>
</tr>
<tr>
<td>Return(Cut (&quot;Complete all the blanks.&quot;, 10, 4))</td>
<td>Complete the blanks</td>
<td>Goes to position 10 to begin the cut and removes four characters.</td>
</tr>
<tr>
<td>Return(Cut (&quot;Complete all the blanks.&quot;, 9))</td>
<td>all the blanks</td>
<td>Defaults to position 1 to begin the cut and cuts nine characters.</td>
</tr>
</tbody>
</table>

See also  [String Functions on page 78](#)
**DashCode**

Use this function to build a value to assign to a series of fields from the binary value of an integer. This is sometimes called a dash code. A dash code is a type of OMR mark that is read by certain mail, binding, or inserting equipment.

A dash code is a series of horizontal lines aligned in a column — each usually around 1/2 to one inch in length — that are typically on the left or right edge of the paper. The marks are usually expected to be in a uniform (fixed) position. Here is an example of a dash code:

```
   __________
   |         |
   |         |
   |_________|
```

Dash codes can be used, for instance, to represent the beginning or end of a set of pages that are associated in some way. The marks might indicate sequencing, first page, last page, staple requirements, additional pages to be inserted at a given point, the envelope size, or binding requirements.

**NOTE:** The exact meaning, order, and position of each mark depends on the finishing equipment you are using. Check the specifications that came with your equipment and assign the values appropriately.

### Syntax

`DashCode(Value, Bits, RootName, Section, Form, Group, OnString, OffString, Direction, AltLens)`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>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.</td>
</tr>
<tr>
<td>Bits</td>
<td>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.</td>
</tr>
<tr>
<td>RootName</td>
<td>This parameter identifies the initial portion of a series of field names that are to be the repository for the OnString and OffString filled values. The bit number referenced will be appended to each name to form the final name expected to be found on the resulting section. For instance, if <code>MVALUE_</code> is passed as the RootName, the first fill value is assigned to <code>MVALUE_1</code>, the second to <code>MVALUE_2</code>, <code>MVALUE_3</code>, and so on, until the maximum number of bits specified are all mapped. If all 32 bits are mapped, the last field would be <code>MVALUE_32</code>. The associated fields will be filled to their defined length. In most dash code (bar code) type situations, you will want all the fields to be the same length.</td>
</tr>
</tbody>
</table>
The return value indicates the number of fields assigned. A return value of zero (0) means that no fields were found.
Example  Here are some examples:

#val = 11 (which is 1011 in binary)

    DASHCODE(#val, 4, "BFLD");

Assuming that BFLD is a root field name and matching fields are located on the current section, the following assignments are made. Further assume that each field is five characters in length.

    BFLD1 is assigned "_____"
    BFLD2 is assigned "_____"
    BFLD3 is assigned "     " (five spaces)
    BFLD4 is assigned "_____"

    DASHCODE(#val, 4, "BFLD", , , "A", "B");

This example uses the parameters to supply different OnString and OffString parameters.

    BFLD1 is assigned "AAAA"
    BFLD2 is assigned "AAAA"
    BFLD3 is assigned "BBBB"
    BFLD4 is assigned "AAAA"

    DASHCODE(#val, 4, "BFLD", , , "A", "B", 1);

Note the Direction parameter was used to reverse the order of the bits interpretation.

    BFLD1 is assigned "AAAA"
    BFLD2 is assigned "BBBB"
    BFLD3 is assigned "AAAA"
    BFLD4 is assigned "AAAA"

    DASHCODE(#val, 4, "AB", "XYZ", 0, "1,2,3,5");

In this example, the last parameter applies differing lengths to the fields you are mapping. This example also uses alternate OnString and OffString parameters and uses text greater than one character. In this case, the string may be truncated or repeated as necessary to fill the field length.

    BFLD1 is assigned “A”
    BFLD2 is assigned “AB”
    BFLD3 is assigned “XYZ”
    BFLD4 is assigned “ABAB” or “ABABA”

Note that the last example indicates two possible results. During Documaker Workstation entry, the field length is considered paramount and cannot be overridden. During batch operations, it is possible for the data length to override the field length.

See also  Bit/Binary Functions on page 42
**DATE**

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

**Syntax**

\[
\text{Date (Format, Day, Month, Year)}
\]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format</td>
<td>Enter a date format. The default is format 1 (MM/DD?YY).</td>
</tr>
<tr>
<td>Day</td>
<td>Enter an integer day value. The default is based on the current day.</td>
</tr>
<tr>
<td>Month</td>
<td>Enter an integer month value. The default is based on the current month.</td>
</tr>
<tr>
<td>Year</td>
<td>Enter an integer year value. The default is based on the current year.</td>
</tr>
</tbody>
</table>

The system returns a date string that contains a formatted date value. If you omit any of the Day, Month, or Year parameters, the system uses a value based on the current date.

**NOTE:** To change to some date formats, make sure the variable field’s Type field (on the field’s Properties window) is set to alphanumeric.

**Example**

Here are some examples:

(Assume the current date is 07/01/10.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return(Date( ))</td>
<td>07/01/2010</td>
<td>No parameters entered, defaults to current date in date format 1.</td>
</tr>
<tr>
<td>Return(Date(&quot;44 &quot;))</td>
<td>July 1, 2010</td>
<td>Date format 4 selected, with a four-digit year length. Defaults to the current date in the selected format.</td>
</tr>
<tr>
<td>Return(Date (,18,5,2009))</td>
<td>05/18/2010</td>
<td>Defaults to date format 1 using the given values.</td>
</tr>
<tr>
<td>Return(Date(&quot;12 &quot;,18,5))</td>
<td>10/138</td>
<td>Date format I selected with a two-digit year length. Enters the given date values in the selected format.</td>
</tr>
</tbody>
</table>

**See also**

- Date Functions on page 51
- Date Formats on page 52
- Using INI Options on page 8
**Date2Date**

Use this function to convert a date from one format to a new format.

**Syntax**

\[ \text{Date2Date (Date, Format, NewFormat)} \]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>Enter a date string. The system assumes your entry to be in the format specified in the Format parameter. The default is the current date.</td>
</tr>
<tr>
<td>Format</td>
<td>Enter a date format string that describes the contents of the Date parameter. The default is date format 1 (MM/DD/YY).</td>
</tr>
<tr>
<td>NewFormat</td>
<td>Enter the date format you want to convert to. The default is date format 1.</td>
</tr>
</tbody>
</table>

This function converts a date string from one format to another. The new value is formatted according to the `NewFormat` parameter.

**Example**

Here are some examples:

(Assume the current date is 07/01/09 and the variable field called, `arc_date`, contains the hexadecimal value, `BC6792D0`)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \text{Return(Date2Date ()} ]</td>
<td>07/01/2009</td>
<td>No parameters entered—defaults to current date in date format 1.</td>
</tr>
<tr>
<td>[ \text{Return(Date2Date(&quot;02/01/09&quot;, &quot;1&quot;, &quot;44&quot;)} ]</td>
<td>February 1, 2009</td>
<td>Changes the given date (02/01/09) from date format “1” to date format “4”, with a four-digit year.</td>
</tr>
<tr>
<td>[ \text{Return(Date2Date(&quot;09/138&quot;, &quot;G&quot;)} ]</td>
<td>05/18/09</td>
<td>Changes the given date (09/138) from date format G to the default date format 1.</td>
</tr>
<tr>
<td>[ \text{Return(Date2Date (@ (&quot;arc date&quot;), &quot;X&quot;, &quot;4&quot; } ]</td>
<td>BB273650</td>
<td>Returns the current date in a eight character hexadecimal representation.</td>
</tr>
<tr>
<td>[ \text{Return(Date2Date (@ (&quot;arc date&quot;), &quot;X&quot;, &quot;4&quot; } ]</td>
<td>February 29, 2008</td>
<td>Converts the hexadecimal date to month name DD, YYY without leading zeros.</td>
</tr>
</tbody>
</table>

**See also**

- Date Functions on page 51
- Date Formats on page 52
- Using INI Options on page 8
**DATEADD**

Use this function to add a specified number of days, months, and/or years to a date.

**Syntax**

```
DATEADD (Date, Format, Days, Months, Years)
```

**Parameter** | **Description**
---|---
Date | Enter a date string. The system assumes your entry to be in the format specified in the Format parameter. The default is the current date.
Format | Enter a date format string that describes the contents of the Date parameter. The default is date format 1 (MM/DD/YY).
Days | Enter the number of days. The default is zero (0).
Months | Enter the number of months. The default is zero (0).
Years | Enter the number of years. The default is zero (0).

This function adds a specified number of days, months, and years to a given date. The result is formatted according to the Format parameter.

The Days, Months, and Years parameters can be negative or positive. If you enter a negative parameter, the system subtracts the specified days, months, or years.

You do not have to divide the values into components. For example, you can add 300 days and 40 months to a date. The result reflects the appropriate year, month, and day.

**NOTE:** This function tells the system to add days, months, and years—*in that order*. For instance, if you tell the system to add one day and one year to the date 02/28/2007, the result is 03/01/2008—not 02/29/2008.

To get 02/29/2008 as the result, you would use two calculations, first adding the year, then adding the day.

**Example**

Here are some examples (assume the current date is 07/01/09):

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Return(DateAdd( , &quot;1&quot;, 10))</code></td>
<td>07/11/2009</td>
<td>Defaults to the current date which is specified as Date( ) and adds 10 days.</td>
</tr>
<tr>
<td><code>Return(DateAdd(02/01/09, , , 44))</code></td>
<td>10/01/2012</td>
<td>Uses the given date (02/01/09) and adds 44 months. (Note that if you enter &quot;44&quot; as a string, it is automatically converted to an integer.)</td>
</tr>
<tr>
<td><code>Return(DateAdd(&quot;09/138&quot;, &quot;I&quot;, , -3))</code></td>
<td>06/139</td>
<td>The given date (09/138) using date format I is May 18, 2009. Subtracting three years results in the date May 18, 2006. Because 2008 is a leap year, the correct day of the year (counting consecutively from January 1) is 139. The resulting date is returned in the same date format.</td>
</tr>
</tbody>
</table>
See also

Date Functions on page 51
Date Formats on page 52
**DATECnv**

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

**Syntax**

\[
\text{DATECnv (Date, Format, DivideYear, Century)}
\]

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>Enter a date string. The system assumes your entry to be in the format specified in the Format parameter. The default is the current date.</td>
</tr>
<tr>
<td>Format</td>
<td>Enter a date format string that describes the contents of the Date parameter. The default is date format 1 (MM/DD/YY).</td>
</tr>
<tr>
<td>DivideYear</td>
<td>A dividing year value used to determine if the date value belongs to the specified century or the next. The default is the current year plus 40.</td>
</tr>
<tr>
<td>Century</td>
<td>The century to assign if the date falls in the dividing year. Otherwise, the result is this century plus one. The default is the current century.</td>
</tr>
</tbody>
</table>

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 \((\text{current year} + 40) \mod 100\).

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: \((1999 + 40) \mod 100 = 39\).

**NOTE:** In this case, \(\mod\) 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, \(2035 \mod 100 = 35\).

**Function Result Explanation**

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return(DateCnv())</td>
<td>07/01/1999</td>
<td>Defaults to the current date and format 1. Since 99 is greater than 39, this date assumes the current century.</td>
</tr>
<tr>
<td>Return(DateCnv(&quot;07/01/00&quot;))</td>
<td>07/01/2000</td>
<td>Since 00 is not greater than 39, this date assumes the next century.</td>
</tr>
<tr>
<td>Return(DateCnv(&quot;50/138&quot;, &quot;I&quot;, 50))</td>
<td>2050/138</td>
<td>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.</td>
</tr>
<tr>
<td>Return(DateCnv(&quot;99/138&quot;, &quot;I&quot;, 50))</td>
<td>1999/138</td>
<td>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.</td>
</tr>
</tbody>
</table>
See also

Date Functions on page 51
Date Formats on page 52
Using INI Options on page 8
**Day**

Use this function to get the day portion of a date as an integer.

**Syntax**

```plaintext
Day(Date, Format, Locale)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>Enter a date string. The system assumes your entry to be in the format specified in the Format parameter. The default is the current date.</td>
</tr>
<tr>
<td>Format</td>
<td>Enter a date format string that describes the contents of the Date parameter. The default is date format 1 (MM/DD/YY).</td>
</tr>
<tr>
<td>Locale</td>
<td>(Optional) Enter the locale code. If you omit this parameter, the system checks the Locale INI option. If the Locale INI option offers no value, the system defaults to USD (United States/English).</td>
</tr>
</tbody>
</table>

The system determines the day portion of the given date based on the format you specify in the Format parameter.

**Example**

Here are some examples:

(Assume the current date is 07/01/09.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return(Day( ))</td>
<td>1</td>
<td>Defaults to the current date and enters the integer 1.</td>
</tr>
<tr>
<td><code>datestring = DateAdd( , , 15); Return(Day(datestring))</code></td>
<td>16</td>
<td>First the DateAdd function defaults to the current date and adds 15 days which results in a date of July 16, 2009. This date is returned to the target variable <code>datestring</code>. The date is then used by the Day function and the integer value of 16 is returned.</td>
</tr>
<tr>
<td>Return(Day(&quot;09/138&quot;, &quot;I&quot;))</td>
<td>18</td>
<td>The given date (09/138) in date format I is May 18, 2009. Therefore, the integer value of 18 is returned.</td>
</tr>
</tbody>
</table>

See also

- Date Functions on page 51
- Locales on page 55
- Date Formats on page 52
- DateAdd on page 184
**DayName**

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

**Syntax**

```
DayName (DayOfWeek, Locale)
```

**Parameter** | **Description**
--- | ---
**DayOfWeek** | Enter an integer to designate the day of the week.
1 - Sunday
2 - Monday
3 - Tuesday
4 - Wednesday
5 - Thursday
6 - Friday
7 - Saturday
The default is the current day of the week.

**Locale** | (Optional) Enter the locale code. If you omit this parameter, the system checks the Locale INI option. If the Locale INI option offers no value, the system defaults to USD (United States/English).

This function is typically used with the WeekDay function. The WeekDay function determines the day of the week number from a given date.

**Example**

Here are some examples:

(Assume the current date is Saturday, January 3, 2009.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return(DayName( ))</td>
<td>Saturday</td>
<td>Defaults to the current day of the week and returns Saturday.</td>
</tr>
<tr>
<td>DayName(WeekDay (&quot;09/33&quot;, &quot;T&quot;))</td>
<td>Monday</td>
<td>First the WeekDay function determines the day of the week number for the given date and format. DayName then uses this number to return the correct day name: Monday.</td>
</tr>
<tr>
<td>DayName (WeekDay( DateAdd(&quot;&quot;,1)))</td>
<td>Friday</td>
<td>First the DateAdd function uses the current date and subtracts one day. WeekDay then determines the number for the day of the week. DayName then determines that the given date is Friday, January 2, 2009 and returns the day name: Friday.</td>
</tr>
<tr>
<td>Return(DayName (&quot;ZAA&quot;))</td>
<td>Saterdag</td>
<td>It returns the name of the current day of the week based and translates that name into Afrikaans.</td>
</tr>
</tbody>
</table>

See also

- **Date Functions on page 51**
- **Locales on page 55**
- **Using INI Options on page 8**
- **DateAdd on page 184**
- **WeekDay on page 424**
**DaysInMonth**

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

**Syntax**

```
DaysInMonth (Month, Year)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Month</td>
<td>Enter a month number from 1 to 12, with January being 1 and December being 12. The default is the current month.</td>
</tr>
<tr>
<td>Year</td>
<td>Enter a year. The default is the current year.</td>
</tr>
</tbody>
</table>

The year value is only used when the month number is 2 (February). The result for February is different if the given year is a leap year. This function is typically used with the Month function. The Month function extracts the month number from a given date.

**Example**

Here are some examples:

(Assume the current date is 07/01/09.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DaysInMonth ()</td>
<td>31</td>
<td>Defaults to the current date and returns the value 31 since July has 31 days.</td>
</tr>
<tr>
<td>DaysInMonth (Month (&quot;04/15/2009&quot;) )</td>
<td>30</td>
<td>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.</td>
</tr>
<tr>
<td>DaysInMonth(2, 2008)</td>
<td>29</td>
<td>The year 2008 was a leap year, February had 29 days. Therefore the integer 29 is returned.</td>
</tr>
</tbody>
</table>

**See also**

Date Functions on page 51

Month on page 321
**DaysInYear**

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

**Syntax**

```yaml
DaysInYear (Year)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
<td>Enter the year. The default is the current year.</td>
</tr>
</tbody>
</table>

This function returns 365 or 366, depending on whether the year parameter is a leap year. This function is typically used with the Year function. The Year function extracts the year number from a given date.

**Example**

Here are some examples:

(Assume the current date is 07/01/2008.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DaysInYear ( )</td>
<td>366</td>
<td>2008 is a leap year, therefore the returned value is 366.</td>
</tr>
<tr>
<td>DaysInYear (09)</td>
<td>365</td>
<td>The year 2009 is not a leap year and has 365 days.</td>
</tr>
<tr>
<td>DaysInYear (Year(&quot;2009/09/09&quot;, &quot;34&quot;))</td>
<td>365</td>
<td>First the Year function extracts the year number (2009) from the given date using the format specified. The DaysInYear function then determines that the given year has 365 days and returns the integer 365.</td>
</tr>
</tbody>
</table>

**See also**

- Date Functions on page 51
- Year on page 448
**DBAdd**

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

**Syntax**

`DBAdd (Table, PrefixVariable)`

**Parameter** | **Description**
--- | ---
Table | Enter the name of an open table.
PrefixVariable | (Optional) Enter the name of a DAL variable to associate with the record fields of the table. The default is Table.

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

Unlike for the DBFirstRec and DBNextRec procedures, the PrefixVariable parameter and the associated fields should have already been defined. For some database handlers, these column names are case sensitive. Columns not required can be left blank.

The actual variable names appended with a prefix are taken from the DFD file. The DFD file is determined by your entry in the Table parameter or by using the column names found in the table if there is no DFD file associated with that table.

Possible causes for failure to add the record include:

- A required column was left blank
- Database specific failure

**Example**

Here is an example:

```
RECORD.Company=“Oracle”;
RECORD.Lob=“Util”;
RECORD.Rundate=DATE();
DBAdd( “APPIDX” , “RECORD” )
```

Possible causes for failure to add the record include:

- A required column was left blank
- Database specific failure

**Table**: Assuming the table APPIDX has the columns Company, Lob, and Rundate, a new record will be added to the table whose values in those columns are Oracle, Util, and the current date, respectively.

**See also** Database Functions on page 43
**DBClose**

Use this procedure/function to close a database table.

Syntax

\[ \text{DBClose} (\text{Table}) \]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>Enter the name of the table you want to close.</td>
</tr>
</tbody>
</table>

The system closes the table and returns one (1) if the table was successfully closed. If the table cannot be closed, it may be because...

- The table was not open, such as if it had already been closed
- There was a database-specific failure

Example

Here is an example:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBClose(&quot;APPIDX&quot;)</td>
<td>1 or 0</td>
<td>Closes the table named APPIDX.</td>
</tr>
</tbody>
</table>

See also

- Creating Variable Length Records from Flat Files on page 200
- Setting Up Memory Tables on page 50
- DBOpen on page 199
- Database Functions on page 43
**DBDelete**

Use this procedure/function to delete all records which match the key criteria from the database table.

**Syntax**

```
DBDelete (Table, KeyName1, KeyValue1, KeyName2, KeyValue2, ...)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>Enter the name of an open table.</td>
</tr>
<tr>
<td>KeyName,</td>
<td>Each KeyName refers to the name of a column to search. For some database handlers, this may be a case-sensitive comparison.</td>
</tr>
<tr>
<td>KeyValue, ...</td>
<td>Each KeyValue is the value of the corresponding KeyName for which to search. For some database handlers, this may be a case-sensitive comparison.</td>
</tr>
<tr>
<td></td>
<td>At least one KeyName/KeyValue pair are required.</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBDelete (&quot;APPIDX&quot;,</td>
<td>1 or 0</td>
<td>Assuming Company and Lob are valid key components for the APPIDX table, the procedure will delete all records with the value SAMPCO in the column named Company and the value Util in the column named Lob.</td>
</tr>
<tr>
<td>&quot;Company&quot;, &quot;SAMPCO&quot;, &quot;Lob&quot;,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;Util&quot;)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See also [Database Functions on page 43](#)
**DBFind**

Use this procedure/function to retrieve the first record from a database table which satisfies the key criteria.

Syntax  

```
DBFind (Table, Variable, KeyName1, KeyValue1, KeyName2, KeyValue2,...)
```

Parameter | Description
--- | ---
Table | Enter the name of an open table.
Variable | Enter the name of a DAL variable to associate with the record fields retrieved by the procedure. The default is Table.
KeyName, KeyValue, ... | Each KeyName specifies a column to search. For some database handlers, it may be a case-sensitive 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 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:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBFind( &quot;APPIDX&quot;, &quot;RECORD&quot;, &quot;Company&quot;,&quot;Oracle&quot;, &quot;Lob&quot;, &quot;DM&quot; )</td>
<td>1 or 0</td>
<td>Assuming that the APPIDX table has columns named <em>Company</em> and <em>Lob</em>, and that these columns are a key, the first record containing &quot;Oracle&quot; and &quot;DM&quot; in the appropriate column will be retrieved and associated with the prefix variable RECORD.</td>
</tr>
</tbody>
</table>

See also

Database Functions on page 43

DBPrepVars on page 201
**DBFirstRec**

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

**Syntax**

```
DBFirstRec (Table, PrefixVariable)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>Enter the name of an open table.</td>
</tr>
<tr>
<td>PrefixVariable</td>
<td>Enter the name of a DAL variable to associate with the record fields retrieved by the procedure. The default is Table.</td>
</tr>
</tbody>
</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:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBFirstRec (&quot;APPIDX&quot;, &quot;RECORD&quot;)</td>
<td>1 or 0</td>
<td>Retrieves the first record from the APPIDX table and associates the columns with the prefix variable RECORD.</td>
</tr>
</tbody>
</table>

**See also**

- Database Functions on page 43
- DBNextRec on page 198
**DBNextRec**

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

**Syntax**

```
DBNextRec (Table, PrefixVariable)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>Enter the name of an open table.</td>
</tr>
<tr>
<td>PrefixVariable</td>
<td>Enter the name of a DAL variable to associate with the record fields retrieved by the procedure. The default is Table.</td>
</tr>
</tbody>
</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:

```
<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBNextRec (&quot;APPIDX&quot;, &quot;RECORD&quot;)</td>
<td>1 or 0</td>
<td>Will retrieve the next record from the table APPIDX and associate the field columns with the prefix variable RECORD.</td>
</tr>
</tbody>
</table>
```

**See also**

- [Database Functions on page 43](#)
- [DBFirstRec on page 197](#)
**DBOpen**

Use this procedure/function to open the specified database table in the mode you request. The DBOpen procedure supports having multiple:

- Simultaneous ODBC connection via different ODBC drivers. See Database Functions on page 43 for more information.
- Tables open in the same database.

**Syntax**

`DBOpen (Table, Handler, DFDFile, Mode, Truncate)`

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>Enter the name of the table you want to open.</td>
</tr>
<tr>
<td>Handler</td>
<td>Enter the name of the database handler to associate with the table. If you omit Handler, DBOPEN looks in the DBTable:TableName control group for the DBHandler option. If this option is not present, DBOPEN defaults to the ODBC handler.</td>
</tr>
<tr>
<td>DFDFile</td>
<td>Enter the name of the format file to associate with the table. If omitted, the Handler tries to query the information from the database. Note that this may not be supported by all databases.</td>
</tr>
<tr>
<td>Mode</td>
<td>Enter a string which specifies the mode in which to open the file. Your options are READ, WRITE, FAIL_IF_EXISTS, and CREATE_IF_NEW. These may be combined by separating them with an ampersand (&amp;), as in &quot;READ&amp;WRITE&amp;FAIL_IF_EXISTS&quot;. You can include spaces between the tokens. If omitted, the open mode defaults to READ &amp; WRITE &amp; CREATE_IF_NEW.</td>
</tr>
<tr>
<td>Truncate</td>
<td>Include this parameter to remove all records from a database table. This lets you use dynamic tables with DAL where the tables are created on a fly, records added, and then deleted.</td>
</tr>
</tbody>
</table>

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:

```c
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:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBOpen (&quot;APPIDX&quot;, &quot;ODBC&quot;, &quot;READ&quot;)</td>
<td>1 or 0</td>
<td>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.</td>
</tr>
<tr>
<td>DBOpen(&quot;MYTABLE&quot;,&quot;ODBC&quot;,&quot;D:\deflib\mytable.dfd&quot;,&quot;READ&amp;WRITE&amp;TRUNCATE&quot;)</td>
<td></td>
<td>This DAL statement removes all rows from the table named MYTABLE.</td>
</tr>
</tbody>
</table>

See also

Setting Up Memory Tables on page 50

DBClose on page 193

Database Functions on page 43
**DBPrepVars**

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

**Syntax**

```
DBPrepVars (Table, PrefixVariable)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>Enter the name of an open table.</td>
</tr>
<tr>
<td>PrefixVariable</td>
<td>PrefixVariable is the name of the DAL variable to associate with the record fields retrieved by the procedure. The default is Table.</td>
</tr>
</tbody>
</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:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBPrepVars(&quot;APPIDX&quot;,&quot;RECORD&quot;);</td>
<td>1 or 0</td>
<td>Creates the DAL variables for the APPIDX table. Each column name is appended with the prefix variable RECORD.</td>
</tr>
</tbody>
</table>

**See also** [Database Functions on page 43](#)
DBUnloadDFD

Use this procedure/function to streamline the use of DAL with ODBC and memory tables by creating DFD files and using only memory tables. You can use the DALRUN program to create the DFD files based on a DAL script since it is a one-time operation. You only need to run the script again after table layout changes.

Syntax

DBUnloadDFD (TableName, DFDName)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableName</td>
<td>Enter the name of the table opened with DBOpen procedure.</td>
</tr>
<tr>
<td>DFDName</td>
<td>Enter the name of the output file. The system overwrite this file if it exists.</td>
</tr>
</tbody>
</table>

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:

```c
#rc = DBOpen("MYTABLE","ODBC");
if #rc = 0
  * display error
end
#rc = DBUnloadDFD("MYTABLE","aaa.dfd");
if #rc = 0
  * display error
end
```

This script unloads a DFD file named AAA.DFD which describes the table named MYTABLE in the current directory.

See also

Database Functions on page 43
DBUPDATE

Use this procedure/function to update the database table record which satisfies the key criteria.

Syntax

```
DBUpdate (Table, Variable, KeyName1, KeyValue1, KeyName2, KeyValue2,...)
```

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:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBFirstRec(&quot;APPIDX&quot;, &quot;RECORD&quot;); RECORD.RUNDATE = DATE(); DBUpdate(&quot;APPIDX&quot;, &quot;RECORD&quot;,&quot;UNIQUE_ID&quot;,RECORD.UNIQUE_ID)</td>
<td>1 or 0</td>
<td>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).</td>
</tr>
</tbody>
</table>

See also

Database Functions on page 43
DBFind on page 195
DBFirstRec on page 197
DBNextRec on page 198
DDTSourceName

Use this function to return the contents of the Source Name field in the DDT file you are currently processing. This function is only applicable during Documaker Server processing.

**NOTE:** As of version 11.0, DDT fields are physically stored inside FAP files.

**Syntax**

```
DDTSourceName ( )
```

There are no parameters for this function.

**Example**

Here is an example:

```
MYROOT = RootName(DDTSourceName( ))
```

**See also**

Documaker Server Functions on page 58
**Dec2Hex**

Use this function to return the hexadecimal equivalent of an integer value.

**Syntax**

\[ \text{Dec2Hex (Value1, Digits)} \]

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value1</td>
<td>This parameter specifies a integer value to be converted into a hexadecimal string value. If the parameter is not specified as an integer, it will be converted to an integer before performing the operation. The largest hexadecimal value supported is FFFFFFFF. Keep in mind, however, that hexadecimal values are considered <strong>unsigned</strong> 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.</td>
</tr>
<tr>
<td>Digits</td>
<td>This parameter defaults to zero (0) and means the resulting hexadecimal value will not have leading zeros. You can set this parameter from one (1) to eight (8) to control the minimum number of hexadecimal digits returned in the string. If you set the minimum too small to represent the value, it will be ignored.</td>
</tr>
</tbody>
</table>

**Example**

Here is an example:

\[ y = 1000 \]
\[ z = \text{Dec2Hex}(y) \]
\[ \text{Result is } z = 3E8 \]

\[ y = 254220 \]
\[ z = \text{Dec2Hex}(y,8) \]
\[ \text{Result is } z = 0003E10C \]

\[ y = -2 \]
\[ z = \text{Dec2Hex}(y) \]
\[ \text{Result is } z = FFFFFFFE \]

**See also**

- Hex2Dec on page 275
- Bit/Binary Functions on page 42
DeFormat

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

Syntax

DeFormat (String, FieldType, Format)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Enter a valid string of formatted text. The default is the value of current field text.</td>
</tr>
<tr>
<td>FieldType</td>
<td>Enter the field type indicator used to format the first parameter. The default is the value of current field type.</td>
</tr>
<tr>
<td>Format</td>
<td>Enter the format of the first parameter. This is the field format entered in the Properties window. The default is the value of current field format.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeFormat (&quot;1,234.89&quot;, &quot;n&quot;)</td>
<td>&quot;1234.89&quot;</td>
<td>Deformat removes commas but retains decimal points for numeric fields.</td>
</tr>
<tr>
<td>DeFormat (&quot;ABC.123.DEF&quot;, &quot;C&quot;, &quot;3,.123.&quot; )</td>
<td>&quot;ABCDEF&quot;</td>
<td>Deformat removes the custom format characters (.123.) after the third character, which were previously added to the string.</td>
</tr>
<tr>
<td>DeFormat (&quot;$$$$$11,980.00 &quot;, &quot;n&quot;)</td>
<td>11980.00</td>
<td>Deformat removes the “$” characters and commas but retains decimal points for a numeric field.</td>
</tr>
</tbody>
</table>

See also

Field Formats on page 62

String Functions on page 78
DELBLANKPAGES

Use this procedure to remove blank or filler pages in a form set. For instance, you can use this rule to remove blank pages reserved for OMR marks when creating PDF files.

Syntax

```
DELBLANKPAGES ( )
```

There are no parameters for this procedure.

Example

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

This will cause blank pages to be removed from each transaction. To do this, you need these INI settings:

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

Here is an example of the BANNER.DAL file:

```
BeginSub PreBatch
  DelBlankPages()
EndSub
```

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

The API to call from custom code is as follows:

```
DWORD __VMMAPI FAPDelBlankPages(VMMHANDLE objectH, /* formset or form handle */
```

See also the Documaker Administration Guide for information on using banner processing.

See also

AddBlankPages on page 115
Page Functions on page 75
SuppressBanner on page 403
Miscellaneous Functions on page 73
Creating a DAL Script Library on page 5
**DelField**

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

**Syntax**

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

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

The system returns one (1) if it finds and deletes the field or zero (0) if it does not.

**NOTE:** The DelField function can not be used in a script called by these AFGJOB rules: PreTransDAL and PostTransDAL.

**Example**

Let's assume you have the following forms in your form set; *Information* and *Multi-section* in the group named *DAL Test Company*.

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

The form named *Multi-section* is comprised of three sections: Section1, Section2, and Section3. Section1 has objects with these field names: a/n, date, yes/no, and multi-line.

Section2 has the same objects with the same field names as Section1.

Section3 has following objects: graphic, box, and input value.

The DAL script which is executed is on a field named *Test* on *Part1* of *Information*.

Here are some examples:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return(DelField (&quot;abc3&quot;));</td>
<td>1</td>
<td><em>Abc3 on Information/Part1</em> is deleted because the section, field, and group parameters were omitted specified. The system defaulted to the current section, form, and group.</td>
</tr>
<tr>
<td>Return(DelField (&quot;abc3&quot;, &quot;part2&quot;));</td>
<td>1</td>
<td><em>Abc3 on Information/Part2</em> is deleted because you specified Part2 and the form defaulted to the current form, Information. Note that Abc3 will still exist on Information/Part1.</td>
</tr>
<tr>
<td>Return(DelField (&quot;test&quot;));</td>
<td>0</td>
<td><em>Test</em> is not deleted because it is the current field.</td>
</tr>
<tr>
<td>Procedure</td>
<td>Result</td>
<td>Explanation</td>
</tr>
<tr>
<td>-----------------------------------------------------</td>
<td>--------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Return(DelField (&quot;a/n&quot;));</td>
<td>0</td>
<td>The field a/n is not deleted because it is not on Information/Part1.</td>
</tr>
<tr>
<td>Return(DelField (&quot;a/n&quot;, &quot;Section1&quot; ));</td>
<td>0</td>
<td>The field a/n is not deleted because Section1 is not a field on the current form (Information).</td>
</tr>
<tr>
<td>Return(DelField (&quot;a/n&quot;, &quot;Section1&quot;, &quot;Multi-section&quot; ));</td>
<td>1</td>
<td>The field a/n on Multi-section/Section1 is deleted because this field is on the specified form/section.</td>
</tr>
<tr>
<td>Return(DelField (&quot;a/n&quot;, &quot;Multi-section&quot; ));</td>
<td>1</td>
<td>The field a/n on Multi-section/Section1 is deleted because field is on the specified form and the section parameter defaults to the first section on the form. Field a/n on Multi-section/Section1 will still exist. If you immediately execute the script again, the field a/n on Image2 would be deleted.</td>
</tr>
<tr>
<td>Return(DelField (&quot;a/n&quot;, &quot;DAL Test Company&quot; ));</td>
<td>1</td>
<td>The field a/n on Multi-section/Section1 is deleted because it was the first field in the group, DAL Test Company. Field a/n on Multi-section/Section2 will still exist. If you immediately execute the script again, the field a/n on Image2 would be deleted.</td>
</tr>
<tr>
<td>Return(DelField (&quot;box&quot;, &quot;Section3&quot;, &quot;Multi-section&quot; ));</td>
<td>0</td>
<td>The field Box is not deleted because you can only delete variable fields. You can not delete objects such as boxes, charts, lines, text labels, text areas, notes, and so on. You can, however, use the DelLogo function to delete graphics.</td>
</tr>
</tbody>
</table>

See also  
Field Functions on page 61  
Locating Fields on page 64  
DelLogo on page 214
**DelForm**

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

**Syntax**

```
DelForm (Form, Group)
```

**Parameter** | **Description**
--- | ---
Form | Enter the name of the form you want to remove.
Group | Enter the name of the group which contains the form you want to remove. The default is the current group.

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

Remove the specified form from the document set. It is not permitted to remove the form executing the script—the *current* form.

**NOTE:** Removing a form means that all data associated with the form will be lost.

**Example**

Here are some examples:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DelForm( &quot;FORM&quot;)</td>
<td>1 or 0</td>
<td>Assuming FORM is located in the current group and is not the current form, it will be deleted.</td>
</tr>
<tr>
<td>DelForm ( &quot;FORM\3&quot;, &quot;GRP&quot;)</td>
<td>1 or 0</td>
<td>Locate the third occurrence of FORM within the GRP and delete that form.</td>
</tr>
</tbody>
</table>

**See also** [Section Functions on page 77](#)
**DelImage**

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

**Syntax**

```
DelImage (Section, Form, Group, Paginate)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section</td>
<td>Enter the name of the section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of a form in the form set. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of a group to contain the specified form. The default is the current group.</td>
</tr>
</tbody>
</table>
| Paginate  | (Optional) This parameter follows the Group parameter. If you enter anything other than a zero (0), it tells the system that you want form pagination to occur upon the successful removal of the section. If you omit this parameter or enter zero (0), the section is deleted, but no other sections are moved to occupy the space left vacant. Subsequent form re-pagination and the application of section origins may change the layout of the form. Here is an example:

```
DelImage("mySection", , , 1)
```

This example omits the Form and Group parameters, but does include the Paginate parameter.

Note: If you enter zero (0) or omit this parameter, the function works as it prior to version 11.2.

The default is zero (0). |

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

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

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

**NOTE:** Removing a section means that all data associated with that section will be lost.

This procedure does not update the displayed form. Use the Refresh procedure to update the display.
Example

Here are some examples:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DelImage( &quot;SEC&quot;)</td>
<td>1 or 0</td>
<td>Delete the specified section from the current form. This assumes that the named section is not the current section.</td>
</tr>
<tr>
<td>DelImage( &quot;SEC\3&quot;, ,&quot;GRP&quot; )</td>
<td>1 or 0</td>
<td>Locate the third occurrence of SEC in the specified GRP. If this is not the current section, delete the section.</td>
</tr>
</tbody>
</table>

See also

AddImage on page 122
PaginateForm on page 333
Section Functions on page 77
**DelLogo**

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

**Syntax**

```
DelLogo (Graphic, Section, Form, Group)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphic</td>
<td>Enter the name of the graphic to be deleted from a section or form. Graphic names are assigned in Studio or Image Editor.</td>
</tr>
<tr>
<td>Section</td>
<td>Enter the name of a section that contains the specified graphic. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of a form that contains the section. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of a group to use to locate the specified object. The default is the current group.</td>
</tr>
</tbody>
</table>

This procedure deletes the specified graphic from the section or form. The system optionally returns one (1) on success or zero (0) on failure.

**NOTE:** Use the Refresh procedure after you use the DelLogo procedure.

**Example**

Here are some examples:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DelLogo(&quot;LOG1&quot;)</td>
<td>1 or 0</td>
<td>Deletes LOG1 on the current section, form, group.</td>
</tr>
<tr>
<td>DelLogo(&quot;LOG1&quot;, &quot;IMH1\3&quot;,&quot;UpRate&quot;)</td>
<td>1 or 0</td>
<td>Deletes LOG1 on the 3rd occurrence of the named section IMH1 on the form UpRate in the default group.</td>
</tr>
</tbody>
</table>

**See also**

- ChangeLogo on page 163
- HaveLogo on page 272
- InlineLogo on page 282
- RenameLogo on page 359
- Logo on page 303
- Refresh on page 357
- Graphics Functions on page 71
DeWIP

Use this procedure/function to delete the work-in-process and its associated data.

Syntax

DeWIP ( )

There are no parameters for this procedure.

This procedure removes the current work-in-process (form set) information from the WIP.DFD file, deletes the associated data files (POL and DAT, if they exist) from the WIP subdirectory, and writes comments to the AFELOG file to note the work-in-process (form set) was deleted.

This procedure returns success (1) if no error occurred during the complete process, otherwise a failure (0). This procedure only works with the Entry module, it will not work in the data entry mode of Studio or Image Editor.

Example

Here is an example:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeWIP ( )</td>
<td>Deletes the work-in-process.</td>
<td>Deletes information associated with the work-in-process and updates the AFELOG file.</td>
</tr>
</tbody>
</table>

See also

WIP Functions on page 88
Documaker Workstation Administration Guide
Documaker Workstation User Guide
**DESTROYLIST**

Use this function to destroy the XML tree created by the LoadXMLList function.

**Syntax**

`DestroyList (%xXMLTree)`

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%xXMLTree</td>
<td>Enter a list type DAL variable that passes the XML tree handle.</td>
</tr>
</tbody>
</table>

The system returns one (1) for success or zero (0) for failure. The returned DAL variable is of the integer type.

**Example**

For an example, see the DAL script in *Scenario 2 on page 90.*

**See also**

XML Functions on page 89

LoadXMLList on page 302
**DeviceName**

Use this function to return the current output device file name, such as the name of the current print stream output file.

**Syntax**

```
DeviceName ( )
```

There are no parameters for this function.

**Example**

This example shows an example post-transaction banner DAL script:

```
IF TotalSheets() > 16000
    #COUNTER += 1
    CurFile = DeviceName()
    Drive = FileDrive(CurFile)
    Path = FilePath(CurFile)
    Ext = FileExt(CurFile)
    RecipBatch = RecipBatch()
    NewFile = FullFileName(Drive, Path, RecipBatch & #COUNTER, Ext)
    SetDeviceName(NewFile)
    BreakBatch()
END
```

**NOTE:** See FileDrive, FileExt, FileName, FilePath, and FullFileName for information on using DAL functions to manipulate file names.

Keep in mind...

- These print drivers are supported: PCL5, PCL6, PST, MET, AFP, PDF, HTML, and RTF. These print drivers are not supported: EPT, MDR, and GDI.

- All platforms are supported, but note that while UniqueString is supported on z/OS, z/OS does not support PDF or long file names, so the PDF example does not apply to z/OS.

- Both multi- and single-step processing are supported.

The only DAL function actually involved in splitting the print stream is BreakBatch. The others make it easier to implement this functionality. For example, since you need to name the new print stream, you use the SetDeviceName procedure. To find the name of the current device, you use the DeviceName function. If you need to create unique file names, you can use the UniqueString function.

**NOTE:** While you can call all of these DAL functions in Documaker Server or Documaker Workstation, the BreakBatch and SetDeviceName functions are not applicable in Documaker Workstation because it does not use the batch printing engine. DeviceName and UniqueString are applicable to both Documaker Workstation and Documaker Server.

See also

Printer and Recipient Functions on page 76

BreakBatch on page 158
SetDeviceName on page 373
UniqueString on page 421
**DiffDate**

Use this function to determine the number of days difference between two dates and enter that value.

**Syntax**

\[
\text{DiffDate (Date1, Format1, Date2, Format2)}
\]

**Parameter** | **Description**
--- | ---
Date1 | Enter a date string. The system assumes this date string is in the format specified by the Format1 parameter. The default is the current date.
Format1 | Enter a date format string that describes the Date1 parameter. The default is date format 1 (MM/DD/YY).
Date2 | Enter a date string. The system assumes this date string is in the format specified by the Format2 parameter. The default is the current date.
Format2 | Enter a date format string that describes the Date2 parameter. The default is date format 1.

The system returns a positive value if the first date is earlier than the second date. The result is negative if the first date is later than the second date. Use the DiffDate function when the chronological order of the dates is important.

**Example**

Here are some examples:

(Assume the current date is 07/01/95.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DiffDate (&quot;7/15/95&quot;)</td>
<td>-14</td>
<td>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.</td>
</tr>
<tr>
<td>DiffDate (&quot;06/01/95&quot;, &quot;1&quot;)</td>
<td>30</td>
<td>Note that the result is positive because the first date is earlier than the current date.</td>
</tr>
<tr>
<td>DiffDate (&quot;October 31, 1961&quot;, &quot;4&quot;, &quot;10/31/95&quot;, &quot;1&quot;)</td>
<td>12418</td>
<td>Note that two different date formats are used.</td>
</tr>
</tbody>
</table>

**See also**

Date Functions on page 51
Date Formats on page 52
**DiffDays**

Use this function to determine the absolute number of days difference between two dates and return that value.

**Syntax**

\[ \text{DiffDays (Date1, Format1, Date2, Format2)} \]

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date1</td>
<td>Enter a date string. The system assumes this date string is in the format specified by the Format1 parameter. The default is the current date.</td>
</tr>
<tr>
<td>Format1</td>
<td>Enter a date format string that describes the Date1 parameter. The default is date format 1 (MM/DD/YY).</td>
</tr>
<tr>
<td>Date2</td>
<td>Enter a date string. The system assumes this date string is in the format specified by the Format2 parameter. The default is the current date.</td>
</tr>
<tr>
<td>Format2</td>
<td>Enter a date format string that describes the Date2 parameter. The default is date format 1.</td>
</tr>
</tbody>
</table>

The system always returns a positive number regardless of which date string parameter is later in time. The result is always given in number of days regardless of the number of months and/or years that are included.

**Example**

Here are some examples:

(Assume the current date is 07/01/95.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DiffDays (&quot;7/15/95&quot;)</td>
<td>14</td>
<td>The second parameter defaults to the current date. The resulting difference in days is 14.</td>
</tr>
<tr>
<td>DiffDays (&quot;06/01 95&quot;, &quot;1&quot;)</td>
<td>30</td>
<td>The second parameter defaults to the current date.</td>
</tr>
<tr>
<td>DiffDays (&quot;October 31, 1961&quot;, &quot;4&quot;, &quot;10/31/95&quot;, &quot;1&quot;)</td>
<td>12418</td>
<td>Note that two different date formats are used and that the result includes several years worth of days.</td>
</tr>
</tbody>
</table>

See also

- Date Functions on page 51
- Date Formats on page 52
- Using INI Options on page 8
DIFFHOURS

Use this function to calculate the absolute time difference in hours between two times. The system returns an integer value, rounded down to the number of whole hours.

Syntax

DiffHours (Time1, Format1, Time2, Format2)

Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time1</td>
<td>Enter a time string. The system assumes this time string is in the format specified by the Format1 parameter. The default is the current time.</td>
</tr>
<tr>
<td>Format1</td>
<td>Enter a time format string that describes the Time1 parameter. The default is time format 1 (HH:MM:SS).</td>
</tr>
<tr>
<td>Time2</td>
<td>Enter a time string. The system assumes this time string is in the format specified by the Format2 parameter. The default is the current time.</td>
</tr>
<tr>
<td>Format2</td>
<td>Enter a time format string that describes the Time2 parameter. The default is time format 1.</td>
</tr>
</tbody>
</table>

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)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return(DiffHours (&quot;09:30:00 AM&quot;,2))</td>
<td>1</td>
<td>The given time is in format 2. The difference in hours between 9:30:00 AM and the current time is one hour.</td>
</tr>
<tr>
<td>Return(DiffHours (&quot;10:30:00 AM&quot;,2))</td>
<td>0</td>
<td>The given time is in format 2. The difference in hours between 10:30:00 AM and the current time is zero.</td>
</tr>
</tbody>
</table>

See also Time Formats on page 80
**DiffMinutes**

Use this function to calculate the absolute time difference in minutes between two times. The system returns an integer value.

**Syntax**

DiffMinutes (Time1, Format1, Time2, Format2)

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time1</td>
<td>Enter a time string. The system assumes this time string is in the format specified by the Format1 parameter. The default is the current time.</td>
</tr>
<tr>
<td>Format1</td>
<td>Enter a time format string that describes the Time1 parameter. The default is time format 1 (HH:MM:SS).</td>
</tr>
<tr>
<td>Time2</td>
<td>Enter a time string. The system assumes this time string is in the format specified by the Format2 parameter. The default is the current time.</td>
</tr>
<tr>
<td>Format2</td>
<td>Enter a time format string that describes the Time2 parameter. The default is time format 1.</td>
</tr>
</tbody>
</table>

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.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DiffMinutes</td>
<td>120</td>
<td>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.</td>
</tr>
</tbody>
</table>

See also Time Formats on page 80
**DIFFMONTHS**

Use this function to determine the number of months difference between two dates and return that value.

**Syntax**

```
DiffMonths (Date1, Format1, Date2, Format2)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date1</td>
<td>Enter a date string. The system assumes this date string is in the format specified by the Format1 parameter. The default is the current date.</td>
</tr>
<tr>
<td>Format1</td>
<td>Enter a date format string that describes the Date1 parameter. The default is date format 1 (MM/DD/YY).</td>
</tr>
<tr>
<td>Date2</td>
<td>Enter a date string. The system assumes this date string is in the format specified by the Format2 parameter. The default is the current date.</td>
</tr>
<tr>
<td>Format2</td>
<td>Enter a date format string that describes the Date2 parameter. The default is date format 1.</td>
</tr>
</tbody>
</table>

The system calculates the number of complete months between given dates. For example, from 2/10 to 3/10 is considered one month, and from 2/10 to 3/15 is also considered one month.

The system always returns a positive number regardless of which date string parameter is later in time. The result is always given in number of months regardless of the number of years included.

**Example**

Here are some examples:

(Assume the current date is 07/01/95.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DiffMonths (&quot;7/15/95&quot;)</td>
<td>0</td>
<td>The second parameter defaults to the current date. Since the value does not equal an entire month the result is 0.</td>
</tr>
<tr>
<td>DiffMonths (&quot;05/01/95&quot;, &quot;1&quot;)</td>
<td>2</td>
<td>The second parameter defaults to the current date.</td>
</tr>
<tr>
<td>DiffMonths (&quot;October 31, 1961&quot;, &quot;4&quot;, &quot;10/31/95&quot;, &quot;1&quot;)</td>
<td>408</td>
<td>Note that the result includes several years worth of months. In addition, two different date formats are used.</td>
</tr>
</tbody>
</table>

See also  
Date Functions on page 51  
Date Formats on page 52  
Using INI Options on page 8
**DiffSeconds**

Use this function to calculate the absolute time difference in seconds between two times. The system returns an integer value.

**Syntax**

```
DiffSeconds (Time1, Format1, Time2, Format2)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time1</td>
<td>Enter a time string. The system assumes this time string is in the format specified by the Format1 parameter. The default is the current time.</td>
</tr>
<tr>
<td>Format1</td>
<td>Enter a time format string that describes the Time1 parameter. The default is time format 1 (HH:MM:SS).</td>
</tr>
<tr>
<td>Time2</td>
<td>Enter a time string. The system assumes this time string is in the format specified by the Format2 parameter. The default is the current time.</td>
</tr>
<tr>
<td>Format2</td>
<td>Enter a time format string that describes the Time2 parameter. The default is time format 1.</td>
</tr>
</tbody>
</table>

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.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DiffSeconds (&quot;2:04:35PM&quot;, 2, )</td>
<td>7199</td>
<td>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.</td>
</tr>
</tbody>
</table>

**See also**

Time Formats on page 80
**DiffTime**

Use this function to calculate the difference in time between two times. The system returns a signed (positive or negative) value, given in seconds.

**Syntax**

```
DiffTime (Time1, Format1, Time2, Format2)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time1</td>
<td>Enter a time string. The system assumes this time string is in the format specified by the Format1 parameter. The default is the current time.</td>
</tr>
<tr>
<td>Format1</td>
<td>Enter a time format string that describes the Time1 parameter. The default is time format 1 (HH:MM:SS).</td>
</tr>
<tr>
<td>Time2</td>
<td>Enter a time string. The system assumes this time string is in the format specified by the Format2 parameter. The default is the current time.</td>
</tr>
<tr>
<td>Format2</td>
<td>Enter a time format string that describes the Time2 parameter. The default is time format 1.</td>
</tr>
</tbody>
</table>

The system returns a positive value if Time1 is earlier than Time2. The result is negative if Time2 is earlier than Time1.

**Example**

Here is an example:

(Assume the current time is 4:06:50 pm.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DiffTime (&quot;4:06:40PM&quot;, 2)</td>
<td>+10</td>
<td>The second parameter defaults to the current time. The resulting difference in time is +10 seconds.</td>
</tr>
</tbody>
</table>

**See also** Time Formats on page 80
**DiffYears**

Use this function to determine the number of years difference between two dates and return that value.

**Syntax**

```
DiffYears (Date1, Format1, Date2, Format2)
```

**Parameter** | **Description**
--- | ---
Date1 | Enter a date string. The system assumes this date string is in the format specified by the Format1 parameter. The default is the current date.
Format1 | Enter a date format string that describes the Date1 parameter. The default is date format 1 (MM/DD?YY).
Date2 | Enter a date string. The system assumes this date string is in the format specified by the Format2 parameter. The default is the current date.
Format2 | Enter a date format string that describes the Date2 parameter. The default is date format 1.

The system calculates the number of complete years between the given dates. For example, from 2/10/08 to 2/10/09 is considered one year, while 3/1/08 to 2/29/09 is considered zero years.

The system always returns a positive number, regardless of which date string parameter occurs later.

**NOTE:** When calculating leap years, February 28th and 29th are considered equal, since both represent the last day of February. For example, February 29, 2008 to February 28, 2009, is considered one year.

**Example**

Here are some examples (assume the current date is 07/01/09):

- 01/31/2008 to 01/30/2009 = zero years difference (it will not be a year until 01/31/2009 as the year 2008 is a leap year)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DiffYears (&quot;7/15/09&quot;)</td>
<td>0</td>
<td>The second parameter defaults to the current date. Since the value is not an entire year, the result is zero (0).</td>
</tr>
<tr>
<td>DiffYears (&quot;01/31/2009&quot;, &quot;4&quot;, &quot;01/30/2009&quot;, &quot;1&quot;)</td>
<td>0</td>
<td>The result will not become one (1) until January 31, 2009.</td>
</tr>
<tr>
<td>DiffYears (&quot;01/010/05&quot;, &quot;1&quot;)</td>
<td>4</td>
<td>The second parameter defaults to the current date.</td>
</tr>
<tr>
<td>DiffYears (&quot;October 31, 1975&quot;, &quot;4&quot;, &quot;10/31/09&quot;, &quot;1&quot;)</td>
<td>34</td>
<td>Note that the result includes numerous years. In addition, two different date formats are used.</td>
</tr>
</tbody>
</table>
See also

- Date Functions on page 51
- Date Formats on page 52
- Using INI Options on page 8
**DUPFORM**

Use this procedure/function to duplicate a form. No data is duplicated, except global data that propagates in naturally.

**NOTE:** For the system to be able to duplicate a form, you must first check the Multicopy option in that form’s Properties window.

<table>
<thead>
<tr>
<th>Syntax</th>
<th><code>DupForm (Form, Group)</code></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form</td>
<td>Enter the name of the form you want to duplicate</td>
</tr>
<tr>
<td>Group</td>
<td>(Optional) Enter the name of the group if the form is not in the current group.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>AddForm on page 119</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AddForm_Propagate on page 120</td>
</tr>
<tr>
<td></td>
<td>CopyForm on page 174</td>
</tr>
<tr>
<td></td>
<td>TriggerForm on page 414</td>
</tr>
<tr>
<td></td>
<td>WIP Functions on page 88</td>
</tr>
</tbody>
</table>
**EmbedLogo**

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

**Syntax**

```plaintext
EmbedLogo (Graphic, Section, Form, Group)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphic</td>
<td>Enter the name of the graphic you want to embed.</td>
</tr>
<tr>
<td>Section</td>
<td>Enter the name of a section that contains the graphic. If the current section does not contain the graphic being referenced this parameter is required to locate the section; otherwise this parameter is optional.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of the form that contains the graphic you specified. If the current form does not contain the section for the graphic being referenced this parameter is required to locate the graphic; otherwise, this parameter is optional.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of the form group that contains the graphic you specified. If the current form is not in the form group that contains the graphic being referenced this parameter is required to locate the graphic; otherwise, this parameter is optional.</td>
</tr>
</tbody>
</table>

Execute this DAL procedure for each graphic on the form or section. This procedure sets the embedded graphic flag in the graphic bitmap structure. Documaker Workstation and Documaker Server check for this flag when they write to the NAFILE.DAT file.

If the flag is not set, the graphic data is not written to the NAFILE.DAT file. Place this procedure in the data field of the IF or DAL rule when used with Documaker Server.

This procedure returns success (1) if no error occurred during the complete process, otherwise a failure (0).

**NOTE:** If the LoadCordFAP in the RunMode control group is set to No; then Documaker Server execution requires you to include the section level rule, CheckImageLoaded.

**Example**

Here is an example:

```plaintext
rc = EmbedLogo("JaneDoe");
```

**Result** | **Explanation**
---|---
1 | The embedded graphic flag in the JaneDoe bitmap structure will be set to On.

**See also**

Section Functions on page 77
**Exists**

Use this function to determine if a DAL symbolic variable exists. This can be useful because referencing a variable that does not exist will cause a runtime syntax error. You can use this function to verify that DAL variables which are created external to your script have been created before you try to reference them.

**Syntax**

```
Exists (Symbol)
```

**Parameter** | **Description**
--- | ---
Symbol | Specify the name of a DAL symbolic variable. This can be from an expression or from another string variable.

The system returns (1) if the variable exists, otherwise it returns zero (0).

**Example**

Here is an example. Assume the string variables 'tbl_1', 'tbl_2', 'tbl_3', and 'tbl_4' respectively contain: 'Ford', 'Chev', 'Olds', and 'VW'.

```
If Exists("tbl_" & #line) Then
    Return ( GetValue("tbl_" & #line) )
Else
    Return (" ")
End
```

In this example, if #line is set to 3, the string 'Olds' is returned. If #line is set to 5, a 'blank' is returned.

**See also**

- GetValue on page 263
- Miscellaneous Functions on page 73
**FIELDFORMAT**

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

**Syntax**

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

**Parameter** | **Description**
--- | ---
Field | Enter the name of a field. The default is the name of the current field.
Section | Enter the name of the section that contains the field you specified in the Field parameter. The default is the current section.
Form | Enter the name of the form that contains the section and/or field you specified. The default is the current form.
Group | Enter the name of the form group that contains the form, section, and/or field you specified. The default is the current group.

Certain field types (like date and numeric data types) will sometimes have additional format information specified. Typically, a user will not be concerned with this type since the fields are designed appropriately for data entry. However, a script may be written that does not assume the field’s format and must query the information to be accurate.

The value returned from this function is a string. If a field cannot be located matching the specified information, an empty string will be returned.

**Example**

Here are some examples:

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return(FieldFormat (&quot;First&quot;))</td>
<td>ZZ9.99</td>
<td>Locate the field and return its format. This example assumes that the field was a numeric type with a format of ZZ9.99.</td>
</tr>
<tr>
<td>Return(FieldFormat (&quot;Second&quot;))</td>
<td></td>
<td>This example returns an empty string. This either means the field has no format string or could not be located.</td>
</tr>
<tr>
<td>Return(FieldFormat (&quot;Third&quot;, &quot;FRM&quot;))</td>
<td>1/4</td>
<td>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.</td>
</tr>
</tbody>
</table>

**See also**

- Field Functions on page 61
- Field Formats on page 62
- Locating Fields on page 64
**FieldName**

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

**Syntax**

```
FieldName (Count, Field, Section, Form, Group)
```

**Parameter**  
**Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Enter positive or negative number. The system uses your entry to move beyond the field you specify. The default is zero (0).</td>
</tr>
<tr>
<td>Field</td>
<td>Enter the name of a field. The default is the current section.</td>
</tr>
<tr>
<td>Section</td>
<td>Enter the name of a section that contains the field named. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of a form that contains the section and/or field named. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of the form group that contains the form, section, and/or field named. The default is the current group.</td>
</tr>
</tbody>
</table>

Keep in mind you can use an asterisk (*) as the object name to match parent objects. This lets you find objects without explicitly knowing the parent names.

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

At first glance, FieldName may seem like an odd function. After all, one of its parameters is a field name. This function first locates the specified field. If you omit the FieldName parameter, the system uses the current field. Then the count is used to move to another field on the section.

A positive or negative number can be used for the count parameter. A positive count moves forward from the located field. A negative count moves backward from the located field. Forward and backward refer to the order that the field appears in the section’s edit list, not necessarily to physical position on the section. All fields are included in the search regardless of whether they are editable or not.

If the system cannot find a field that matches the information you specified, it returns an empty string.

**Example**

Here are some examples: (Assume the section has three fields named First, Second, and Third, which occur in that order.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return(FieldName (1, &quot;Second&quot;).)</td>
<td>Third</td>
<td>Locate the field named Second and then move to the next field.</td>
</tr>
</tbody>
</table>
### Function Result Explanation

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Return(Field Name (-1, “Second”))</code></td>
<td>First</td>
<td>Locate the field named Second and then move to the previous field.</td>
</tr>
<tr>
<td><code>Return(Field Name ( 8, “MyField”, , “FRM”))</code></td>
<td>a name or “”</td>
<td>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.</td>
</tr>
</tbody>
</table>

See also:  
- Name Functions on page 74  
- Field Formats on page 62  
- Locating Fields on page 64
**FieldPrompt**

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

**Syntax**

FieldPrompt (Field, Section, Form, Group)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>Enter the name of a field. The default is the current field.</td>
</tr>
<tr>
<td>Section</td>
<td>Enter the name of a section that contains the field named. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of a form that contains the section and/or field named. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of the form group that contains the form, section, and/or field named. The default is the current group.</td>
</tr>
</tbody>
</table>

The system returns a string unless it cannot find a field that matches your criteria. If the system cannot find a field that matches the criteria you specified, it returns an empty string.

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

**Example**

Here are some examples:

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return(FieldPrompt(“Name”))</td>
<td>Name</td>
<td>Locates the field on the current section and returns its prompt.</td>
</tr>
<tr>
<td>Return(FieldPrompt(“Address1”))</td>
<td>Street Address</td>
<td>Locates the field on the current section and returns its prompt.</td>
</tr>
</tbody>
</table>

**See also**

- Field Functions on page 61
- Field Formats on page 62
- Locating Fields on page 64
**FieldRule**

Use this procedure/function when you need to execute a field-level rule in a DAL script. See the Rules Reference for more information on field-level rules.

**NOTE:** The FieldRule procedure requires a section to be able to process. It cannot be used in an external DAL script called by the SETRCPTB.DAT file, a custom rule, the RecipIf rule, or placed in the SETRCPTB.DAT custom rule parameters field.

**Syntax**

```java
FieldRule ( )
```

There are no parameters for this procedure.

This procedure lets you execute field-level rules from within a DAL script. The DAL script is called by one of these Documaker processing rules: DAL or IF. This procedure requires the same number of parameters as are required for a field level rule. While not all fields must contain data, you must include the correct number of delimiters.

You can use overflow variables if the called field level rule supports overflow. Generally, the IF rule does not support overflow but it can be supported using the FieldRule procedure. See the examples for this procedure for more information.

**NOTE:** All semicolons in a field-level rule must be replaced with two colons (::). If any of your DDT parameters contain quotation marks ("), use instead apostrophes (’) to send in the DDT information. Here is an example:

```
FIELDRULE('::0::1::FIELD_NAME::45::4::FIELD_NAME::0::4::move_it::!
/field1/field2[field3="Yes"]::N::N::N::N::::::')
```

Here is a list of parameters for this procedure with sample entries. The entries illustrate the following example. An asterisk indicates the parameter is generally required, depending on the rule you are using.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>File number</td>
<td>* (required by TblLkUp)</td>
<td>0</td>
</tr>
<tr>
<td>Record number</td>
<td>* (required for overflow)</td>
<td>1</td>
</tr>
<tr>
<td>Source field name</td>
<td>* (required by TblText)</td>
<td>Town_State</td>
</tr>
<tr>
<td>Source field offset</td>
<td>*</td>
<td>55</td>
</tr>
<tr>
<td>Source field length</td>
<td>*</td>
<td>9</td>
</tr>
<tr>
<td>Destination field name</td>
<td>*</td>
<td>Rec-Town_State</td>
</tr>
<tr>
<td>Destination field offset</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>Destination field length</td>
<td>*</td>
<td>25</td>
</tr>
</tbody>
</table>
For example, suppose you want the transaction sent to WIP when the record PRODAREC, at offset 11, contains a string of four characters ("0000") starting at position 20. And, you always want the system to get 25 characters of data from PRODAREC, starting at position 65. Furthermore, you want the system to remove any trailing spaces.

For this scenario, you would use the FieldRule procedure to call the KickToWip field level rule and use the standard IF rule to do the rest. The script for this example would look like this:

```
::A={11,PRODAREC 20,4}::B={11,PRODAREC 65,25}:: IF(A='0000')::
FieldRule("::0::1::Town_State::55:9::;Rec-Town_State::0::25::::
KickToWip::N::N::Y::N::3001::5602::11010::")::Else::B=Trim(B)::
Return("^" & B & "^")::End::Return("^" & 1 & "^");
```

Here's another example. Suppose you want to move multiple lines of text from $N$ number of specific external extract records to the output buffer when the HEADERREC record (at offset 11) contains an $F$ in position 1.

For this scenario, you could use the FieldRule procedure to call the MoveExt rule and use the standard IF rule to do the rest. The script for this example would look like this:

```
CON={11,HEADERREC 1,1}:: A=FIELDRULE("::0::1::E::45::4::PREM/OPS
RATE1::0::4::moveext::@GETRECSUSED,QCPVR5,OVSYM1/
11,CLSSCDREC::N::N::N::N::::")::if(CON='F')::return("^" & A & "^")::end ;N;N;Y;N;12461;2119;16010
```

---

**Example**

For example, suppose you want the transaction sent to WIP when the record PRODAREC, at offset 11, contains a string of four characters ("0000") starting at position 20. And, you always want the system to get 25 characters of data from PRODAREC, starting at position 65. Furthermore, you want the system to remove any trailing spaces.

For this scenario, you would use the FieldRule procedure to call the KickToWIP field level rule and use the standard IF rule to do the rest. The script for this example would look like this:

```
::A={11,PRODAREC 20,4}::B={11,PRODAREC 65,25}:: IF(A='0000')::
FieldRule("::0::1::Town_State::55:9::;Rec-Town_State::0::25::::
KickToWip::N::N::Y::N::3001::5602::11010::")::Else::B=Trim(B)::
Return("^" & B & "^")::End::Return("^" & 1 & "^");
```

Here's another example. Suppose you want to move multiple lines of text from $N$ number of specific external extract records to the output buffer when the HEADERREC record (at offset 11) contains an $F$ in position 1.

For this scenario, you could use the FieldRule procedure to call the MoveExt rule and use the standard IF rule to do the rest. The script for this example would look like this:

```
CON={11,HEADERREC 1,1}:: A=FIELDRULE("::0::1::E::45::4::PREM/OPS
RATE1::0::4::moveext::@GETRECSUSED,QCPVR5,OVSYM1/
11,CLSSCDREC::N::N::N::N::::")::if(CON='F')::return("^" & A & "^")::end ;N;N;Y;N;12461;2119;16010
```

**See also**

Documaker Server Functions on page 58
Field Formats on page 62
Locating Fields on page 64
**FieldType**

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

**Syntax**

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

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>Enter the name of a field. The default is the current field.</td>
</tr>
<tr>
<td>Section</td>
<td>Enter the name of a section that contains the field you specified. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of a form that contains the section and/or field you specified. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of the form group that contains the form, section, and/or field you specified. The default is the current group.</td>
</tr>
</tbody>
</table>

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 field type is a string. If a field cannot be located matching the specified information, an empty string will be returned.

**Example**

Here are some examples:

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return(FieldType(&quot;First&quot;))</td>
<td>n</td>
<td>Locate the field and return its type. This example assumes that the field was a numeric type.</td>
</tr>
<tr>
<td>Return(FieldType(&quot;Second&quot;))</td>
<td>k</td>
<td>This example returns K which corresponds to the International Alphanumeric data type.</td>
</tr>
<tr>
<td>Return(FieldType(&quot;MyField&quot;, &quot;FRM&quot;))</td>
<td>m</td>
<td>Locate the form specified within the current form group. Then locate MyField anywhere on that form. If found, the field’s type is returned. In this example, M corresponds with the X or Space field type.</td>
</tr>
</tbody>
</table>

**See also**

- Field Functions on page 61
- Field Formats on page 62
- Locating Fields on page 64
**FieldX**

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

**Syntax**

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>Enter the name of a field. The default is the current field.</td>
</tr>
<tr>
<td>Section</td>
<td>Enter the name of the section that contains the field you specified. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of a form that contains the section and/or field you specified. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of the form group that contains the form, section, and/or field you specified. The default is the current group.</td>
</tr>
</tbody>
</table>

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.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return(FieldX( ))</td>
<td>1250</td>
<td>Returns the current field’s X coordinate.</td>
</tr>
<tr>
<td>Return(FieldX (&quot;MyField&quot;))</td>
<td>1250</td>
<td>Returns the field’s X coordinate if the field is located on the current section.</td>
</tr>
<tr>
<td>Return(FieldX(&quot;MyField&quot;, &quot;IMG\2&quot;, &quot;GRP&quot;))</td>
<td>1250</td>
<td>Returns the X coordinate of MyField located on the second occurrence of IMG within the specified form set group.</td>
</tr>
</tbody>
</table>

**See also**

Field Functions on page 61
Field Formats on page 62
Locating Fields on page 64
FieldY on page 239
FieldY

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

Syntax

FieldY (Field, Section, Form, Group)

Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>Enter the name of a field. The default is the current field.</td>
</tr>
<tr>
<td>Section</td>
<td>Enter the name of the section that contains the field you specified. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of the form that contains the section and/or field you specified. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of the form group that contains the form, section, and/or field you specified. The default is the current group.</td>
</tr>
</tbody>
</table>

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.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return(FieldY( ))</td>
<td>6020</td>
<td>Return the current field’s Y coordinate.</td>
</tr>
<tr>
<td>Return(FieldY(“MyField”))</td>
<td>6020</td>
<td>Returns the field’s Y coordinate if located on the current section.</td>
</tr>
<tr>
<td>Return(FieldY(“MyField”, “FRM”))</td>
<td>6020</td>
<td>Returns the first occurrence of MyField on the specified form.</td>
</tr>
</tbody>
</table>

See also

Field Functions on page 61
Field Formats on page 62
Locating Fields on page 64
FieldX on page 238
**FileDrive**

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

**Syntax**

```
FileDrive (FullFileName)
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FullFileName</td>
<td>Enter a string that contains a fully qualified file name, such as: &quot;d:\mypath\myfile.ext&quot;</td>
</tr>
</tbody>
</table>

The system returns a string that contains the drive component of that file name.

**Example**

Here is an example:

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

In this example, MYDRIVE would contain:

```
"d:"
```

**See also**

FilePath on page 243  
FileName on page 242  
FileExt on page 241  
FullName on page 249  
File and Path Functions on page 68
**FileExt**

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

**Syntax**

```plaintext
FileExt (FullFileName)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FullFileName</td>
<td>Enter a string that contains a fully qualified file name, such as:</td>
</tr>
<tr>
<td></td>
<td>&quot;d:\mypath\myfile.ext&quot;</td>
</tr>
</tbody>
</table>

The system returns a string that contains the extension component of that file name.

**Example**

Here is an example:

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

In this example MYEXT would contain:

```
".ext"
```

**See also**

- File and Path Functions on page 68
- FullFileName on page 249
- FileDrive on page 240
- FilePath on page 243
- FileName on page 242
**FileName**

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

**Syntax**

```
FileName (FullFileName)
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FullFileName</td>
<td>Enter a string that contains a fully qualified file name, such as:</td>
</tr>
<tr>
<td></td>
<td>“d:\mypath\myfile.ext”</td>
</tr>
</tbody>
</table>

The system returns a string that contains the name component of that file name.

**Example**

Here is an example:

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

In this example, MYNAME would contain:

```
"myfile"
```

**See also**

- File and Path Functions on page 68
- FullFileName on page 249
- FileDrive on page 240
- FilePath on page 243
- FileExt on page 241
FilePath

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

Syntax

FilePath (FullFileName)

Parameter | Description
---|---
FullFileName | Enter a string that contains a fully qualified file name, such as: “d:\mypath\myfile.ext”

The system returns a string that contains the path component of that file name.

Example

Here is an example:

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

In this example, MYPATH would contain:

“\mypath\”

See also

File and Path Functions on page 68
FullFileName on page 249
FileDrive on page 240
FileExt on page 241
FileName on page 242
**FIND**

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

**Syntax**

```markdown
Find (String, Substring, Integer)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Enter a valid string. The default is the value of the current field.</td>
</tr>
<tr>
<td>Substring</td>
<td>A string of one or more characters that will be located in parameter one.</td>
</tr>
<tr>
<td>Integer</td>
<td>Choose from these options: 0 - a left to right search 1 - a right to left search Both search options return a position relative to the first (left-hand) character of the string parameter. The default is zero (0).</td>
</tr>
</tbody>
</table>

The system returns a zero (0) if the substring is not found in the search string, otherwise it returns the position of the substring. The search is not case sensitive.

**Example**

Here are some examples:

(Assume the current field contains the text **Insured's responsibility**.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return(Find (, “RESP” ))</td>
<td>11</td>
<td>Defaults to the current field and finds the first occurrence of “RESP” at position 11. Note that the search is not case sensitive.</td>
</tr>
<tr>
<td>Return(Find (, “usual and customary”))</td>
<td>0</td>
<td>The term “usual and customary” is not found in the current field.</td>
</tr>
<tr>
<td>Return(Find (“Complete all the blanks.”, “all”))</td>
<td>10</td>
<td>Searching left to right, “all” was first found at position 10.</td>
</tr>
<tr>
<td>Return(Find (“Complete all the blanks.”, “all”, 1))</td>
<td>10</td>
<td>Searching right to left, “all” was first found at position 10.</td>
</tr>
</tbody>
</table>

See also **String Functions on page 78**
Use this function to format a string field and return the result.

**Syntax**

```
Format (String, FieldType, Format)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Enter a valid string of non-formatted text. The default is the current field.</td>
</tr>
<tr>
<td>FieldType</td>
<td>Enter the field type indicator you want the system to use to format the first parameter. The default is the current field type.</td>
</tr>
<tr>
<td>Format</td>
<td>Enter the field format you want the system to use to format the first parameter. The default is the current field format.</td>
</tr>
</tbody>
</table>

The system applies formatting to a given string. Some field types do not require format strings to accomplish formatting. For example, the X field type indicator automatically uppercases all letters in a string without requiring a format.

**NOTE:** The variable field *must be* the same length as the format mask.

**Example**

Here are some examples:

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return(Format (&quot;1234.89&quot;, &quot;n&quot;, &quot;zzz,zzz.99&quot;))</td>
<td>1,234.89</td>
<td>Formats the field as numeric, by adding a comma and using two decimal positions, as specified in the Format parameter.</td>
</tr>
<tr>
<td>Return(Format (&quot;ABCDEF&quot;, &quot;C&quot;, &quot;3,.123.&quot;))</td>
<td>ABC.123. DEF</td>
<td>Custom formats the field by adding .123. after the third input character.</td>
</tr>
<tr>
<td>Return(Format (&quot;222334444&quot;, &quot;n&quot;, &quot;999.99-9999&quot;))</td>
<td>222-33-4444</td>
<td>Formats the field as a numeric, by adding hyphens as specified in the Format parameter.</td>
</tr>
</tbody>
</table>

See also

- Field Formats on page 62
- String Functions on page 78
**FORMDESC**

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

**Syntax**

FormDesc (Count, StartForm, Group)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>An index reference to locate a form before or after the specified form. To move backwards, enter a negative number. The default is zero (0).</td>
</tr>
<tr>
<td>StartForm</td>
<td>Enter the name of a form from which to start the search. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of a group which contains the form you specified. The default is the current group.</td>
</tr>
</tbody>
</table>

The system lets you get the description specified in the FORM.DAT file for the specified form, relative to a known form. If you omit all parameters, the system returns the description of the current form.

The Count parameter tells the system to move a number of forms forwards or backwards from the specified form before it returns the form description.

If the system cannot locate the starting form or the Count parameter tells the system to move beyond the number of forms contained in the group, the system returns an empty string.

**Example**

Here are some examples:

Assume there are three forms: FORMA, FORMB, and FORMC. Also assume the current form is FORMB and its description is Fire Form # 2345.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>FormDesc( )</td>
<td>Fire Form # 2345</td>
<td>No parameters will result in returning the current form description.</td>
</tr>
<tr>
<td>FormDesc ( 2, “FormC”)</td>
<td>Empty string</td>
<td>Returns an empty string if the form cannot be located.</td>
</tr>
<tr>
<td>FormDesc ( -1, “FormC”)</td>
<td>Fire Form # 2345</td>
<td>Locates FORMC in the current group. Then returns the description of the form that occurs before this form.</td>
</tr>
</tbody>
</table>

**See also**

FormName on page 247
ImageName on page 277
Name Functions on page 74
DAL Script Examples on page 35
**FORMNAME**

Use this function to get the name from a form.

### Syntax

```
FormName (Count, StartForm, Group)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Enter an index reference to use to locate a form before or after the specified form. The default is zero (0).</td>
</tr>
<tr>
<td>StartForm</td>
<td>Enter the name of the form from which to start the search. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of a group that contains the form you specified. The default is the current group.</td>
</tr>
</tbody>
</table>

The system returns the name of the form it located.

Keep in mind you can use an asterisk (*) as the object name to match parent objects. This lets you find objects without explicitly knowing the parent names.

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

The system lets you get the name of a form relative to a known form. If you omit all parameters, the system returns the name of the current form. The Count parameter moves a number of forms forwards or backwards (negative) from a located form before returning the form name.

If the starting form cannot be located or the Count parameter causes the system to move beyond the number of forms contained in the group, the system returns an empty string.

If there is more than one copy of the form, the name returned contains the occurrence notation used by DAL functions to locate forms. For instance, a name like FORM\3 identifies the third copy of FORM within the same group.

**Example**

Here are some examples:

(Assume there are three forms: FORMA, FORMB, and FORMC. Also assume the current form is FORMB.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>FormName( )</td>
<td>FORMB</td>
<td>No parameters will result in returning the current form name.</td>
</tr>
<tr>
<td>FormName ( -1, “FormC”)</td>
<td>FORMB</td>
<td>Locates FORMC in the current group. Then returns the name of the form that occurs before this form.</td>
</tr>
</tbody>
</table>

See also  
FormDesc on page 246

Name Functions on page 74
**FrenchNumText**

This function is a French version of the NumText function. The NumText function provides written numeric equivalents, such as *One Hundred and Twenty* for 120. The FrenchNumText function serves the same purpose, but its output is in French.

**Syntax**

\[
\text{FrenchNumText} \left( \text{Number}, \text{DollarWord}, \text{CentWord}, \text{Decimode} \right)
\]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>Enter a valid amount. The default is the current field value.</td>
</tr>
<tr>
<td>DollarWord</td>
<td>Enter the word you want to use instead of <em>dollars</em>. The default is:</td>
</tr>
<tr>
<td></td>
<td>“dollars et”</td>
</tr>
<tr>
<td>CentWord</td>
<td>Enter the word you want to use instead of <em>cents</em>. The default is:</td>
</tr>
<tr>
<td></td>
<td>“cents”</td>
</tr>
<tr>
<td>Decimode</td>
<td>Choose from these options:</td>
</tr>
<tr>
<td></td>
<td>1 - numeric decimal amount</td>
</tr>
<tr>
<td></td>
<td>2 - spell decimal amount</td>
</tr>
<tr>
<td></td>
<td>3 - suppress zero, numeric decimal amount</td>
</tr>
<tr>
<td></td>
<td>4 - suppress zero, spell decimal amount</td>
</tr>
<tr>
<td></td>
<td>The default is one (1).</td>
</tr>
</tbody>
</table>

**Example**

Please note the system returns only lowercase letters. For instance, if you entered 2000000, the system would return:

\[
\text{deux millions de dollars et 0 cents}
\]

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return(FrenchNumText())</td>
<td>\text{deux millions de dollars et 0 cents}</td>
<td>The current field value is returned in a written form using <em>dollars et</em> and <em>cents</em>. The zero (0) is displayed in a numeric decimal amount format.</td>
</tr>
<tr>
<td>Return(FrenchNumText(123.45,2) )</td>
<td>\text{cent vingt-trois dollars et quatre-vingt-cinq cents}</td>
<td>The written equivalent for 123.45 is displayed using Decimode 2 with the decimal spelled out.</td>
</tr>
</tbody>
</table>

**See also**

- String Functions on page 78
- NumText on page 327
**FullFileName**

Use this function to make the full file name.

**Syntax**

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

**Parameter** | **Description**
--- | ---
Drive | Enter the drive letter, followed by a colon.
Path | Enter the full path.
Name | Enter the file name, omitting the extension.
Ext | Enter the file extension.

The system accepts a string containing the drive, path, name, and extension components of a fully qualified file name, assembles them, and returns a string that contains the full file name.

Here is an example:

```
MYFILENAME = FullFileName("d:","\mypath\","myfile",".ext")
```

In this example, `MYFILENAME` would contain:

```
"d:\mypath\myfile.ext"
```

**NOTE:** If, in this example, `\mypath` had no trailing slash, the FullFileName function would have added it for you.

Here is a z/OS example:

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

In this example, the result would be:

```
DD:DEFLIB(MEMBER)
```

See also

- File and Path Functions on page 68
- FileDrive on page 240
- FileExt on page 241
- FileName on page 242
- FilePath on page 243
**GetAddresseeValues**

Use this function to get the mapped values of addressee record members, as defined in the Extract Dictionary (XDD) Addressee record. You can use these values to update documents or mailer pages at print time. Use this function with the AddresseeMap capability and the AddresseeCount function.

**Syntax**

```
GetAddresseeValues(RecipName, Index, TagName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RecipName</td>
<td>Enter the name of the recipient for whom you want to know the values of the corresponding addressee record. The name you enter should be a valid recipient name in the document set.</td>
</tr>
<tr>
<td>Index</td>
<td>(Optional) Enter the addressee index to identify the addressee you want to reference. The default is one (1).</td>
</tr>
<tr>
<td>TagName</td>
<td>(Optional) Enter the name of a prefix for the addressee record member names this function returns. For instance, if you enter <code>CST</code>, the addressee Name value would be referenced as <code>CST.Name</code>. The default is the RecipName. The TagName must adhere to DAL variable naming convention. If the RecipName is used and this name contains spaces, the system replaces the spaces with underscores.</td>
</tr>
</tbody>
</table>

When used as a function, GetAddresseeValues returns a one (1) if successful or zero (0) if the recipient or named index was not found. If it returns a zero (0), the system creates the variables with empty values.

**Example**

Here are some examples.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>If GetAddresseeValues (&quot;Memo&quot;,2)</td>
<td></td>
<td>If the Memo recipient contains at least two addressee members, the second member is retrieved and assigned to the appropriate DAL variables. If not, the GetAddresseeValues function returns a zero (0) which indicates a false result to the If statement.</td>
</tr>
<tr>
<td>#Cnt = GetAddresseeValues (&quot;Agent&quot;)</td>
<td>0</td>
<td>The system tries to locate the first addressee defined for the Agent recipient. If Agent is not a valid recipient or does not have any addressees mapped, the system sets #Cnt to zero (0).</td>
</tr>
<tr>
<td>#CNT2 = GetAddresseeValues(&quot;Recipient&quot;,1,&quot;CST&quot;);</td>
<td>0</td>
<td>This says that you want your recipient values tagged with &quot;CST&quot;, so you end up with variables you can reference like <code>CST.NAME, CST.CITY, CST.STATE</code>. Without the quotes, you are saying that whatever value in CST will be the token name. So, you either get an error because this is an undefined variable, or if the variable happens to contain an invalid value, you would get the 0 return value.</td>
</tr>
</tbody>
</table>
Here is another example:

```pascal
If GetAddresseeValues( "Customer", 2)
  Return ( Customer.Name )
END
Return ("Not Provided");
```

This example script locates the Customer recipient’s second addressee and assigns the appropriate values to the DAL variables. If the recipient index is found, a non-zero (true) result causes the If statement to proceed with the subsequent statements and return the name of the addressee member.

If the recipient or addressee index is not located, the DAL variable Customer.Name will be empty. In this example, the script does not use the empty value, but instead returns the following text:

```pascal
Not Provided
```

### See also
- **AddresseeCount on page 128**
- **Have Functions on page 69**

### Function Result Explanation

| XYZ="ABC" #CNT2 = GetAddresseeValues ("Recipient",1, XYZ); |
|-----------------|--------------------------------------------------------|
| This shows how a variable can define the text to tag to the recipient variables. |

| CST="" #CNT2 = GetAddresseeValues ("Recipient",1, CST); |
|-----------------|--------------------------------------------------------|
| 0               | If you pass in an empty string as the tag, it will revert to using the recipient name as the tag — just as if you did not specify the parameter. |
**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**

\[
\text{GetAttachVar} \ (\text{Name}, \ DSIqueue)
\]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter the name of the attachment variable.</td>
</tr>
<tr>
<td>DSIqueue</td>
<td>(Optional) Enter one (1) for input or two (2) for output. The default is one (1).</td>
</tr>
</tbody>
</table>

**See also**

- AddAttachVAR on page 114
- RemoveAttachVAR on page 358
- Docupresentment Functions on page 60
**GetData**

Use this function to retrieve data from a flat file extract file.

**NOTE:** The SrchData function, released in version 11.1 and included in version 11.0, patch 32, lets you include spaces in the search criteria, whereas the GetData function does not. Here is an example:

```plaintext
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**

```plaintext
GetData (SearchMask, Occurence)
```

**Parameter** | **Description**
---|---
SearchMask | Enter the criteria that defines what data you want the system to look for. Format the search mask as shown here:

```
"extract search mask offset, length"
```

Occurrence | This parameter lets you specify which occurrence of the data to get. The default is the first occurrence.

The system returns the data from the extract file based on the search mask.

**Example**

Here is an example:

```plaintext
GetData( "11,HEADEREC 40,17")
```

In this example, the GetData function finds the extract record designated by “11,HEADEREC” and returns the data at offset 40 for a length of 17. The GetData function does not format the data.

You can use an occurrence variable to get the Nth iteration of the data. Enter zero (0) to return the first record, one (1) to return the second, and so on. Here is an example:

```plaintext
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:

```plaintext
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/PolicyImages/PolicyImage/premium_fullterm 1,7") ) = "2549" Then;
Return ("equal - " & GetData ("!/descendant::Personalauto/child::Vehicles/child::Vehicle[**vehovfsym**]/vehicle_num 1,2")
Else Return ("not equal - " & value)
End;
```
In this example, the GetData function checks to see if the specified XML extract record equals 2549, if it does, the function returns the string: equal - concatenated with the value from another XML extract record. If not, it returns the string: not equal - concatenated to a value from a different XML extract record.

See also

- SrchData on page 395
- Documaker Server Functions on page 58
**GetFormAttrib**

Use this function to return the content of the named user attribute (metadata) for the form you specify.

**Syntax**

```plaintext
GetFormAttrib (Name, Form, Group)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter the name of the user attributes (metadata) to retrieve.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of a form from which to retrieve data. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of the group that contains the specified form. The default is the current group.</td>
</tr>
</tbody>
</table>

If you omit both the Form and Group parameters, the system chooses the current form, based on where the script executes. During Entry (via the Workstation or the plug-in) this will be the form that contains the DAL script. During Documaker Server processing, the first logical form found within the document set is the current form, unless the script is executed from a section or field rule.

If you include the Form parameter, but omit the Group parameter, the system looks for the form within the current group of forms, as defined by where the script executes. During Entry (via the Workstation or the WIP Edit plug-in) this is the group that contains the form where the script executes. During Documaker Server processing, the first logical group found within the document set is the current group, unless the script is executed from a section or field rule.

If you omit the Form parameter but include the Group parameter, the system locates the first form within the group you specified.

If you define an attribute, form, or group that is not included in the current document, the system returns an empty string.

**Example**

For the following examples assume that form 1111 has the following metadata. Also assume form 9999 was not selected or triggered.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offer</td>
<td>Good until cancelled</td>
</tr>
<tr>
<td>Codes</td>
<td>R4,79, ZW</td>
</tr>
</tbody>
</table>

Here is the first example:

```plaintext
xx = GetFormAttrib("Offer", "1111")
```

In this example the variable `xx` is set to:

Good until cancelled

Here is another example:

```plaintext
xx = GetFormAttrib("Codes", "9999")
```
In this example the variable \( xx \) is set to an empty string.

See also
- PutFormAttrib on page 347
- Have Functions on page 69
**GetINIBool**

Use this function to retrieve from cache memory the Boolean value of an INI control group and option.

**Syntax**

```plaintext
GetINIBool (Context, Group, Option, Default)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context</td>
<td>(Optional) A name (valid name) associated to a set of INI control groups and options that have been loaded into cache memory.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the group name (valid string) which contains the INI option Boolean value to retrieve.</td>
</tr>
<tr>
<td>Option</td>
<td>Enter the option name (valid string) which contains the INI Boolean value to retrieve. If the control group and option does not contain a Boolean value, the system returns a zero (0).</td>
</tr>
<tr>
<td>Default</td>
<td>(Optional) Enter the default string value to return from the function instead of the actual control group and option value.</td>
</tr>
</tbody>
</table>

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

If you omit the context, the function searches all INI files loaded in memory. If there are multiple control groups and options with the same name, this function returns the first INI control group and option string it finds.

If a context name is present, this function only searches for the control group and option in the set of control groups and options associated with the context name.

**Example**

Let’s assume that an INI file, `TEST1.INI`, was loaded with the context name, `MVF`. The TEST1.INI file contains this control group and option:

```plaintext
< Control >
    LogEnabled = Yes
```

In addition, the FSIUSER.INI file contains this control group and option:

```plaintext
< Control >
    LogEnabled = No
```

Plus, the FSISYS.INI file contains this control group and option:

```plaintext
< Control >
    LogEnabled = Yes
```

Based on this scenario, this table shows and explains several possible results.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bool_value = GetINIBool (&quot;Control&quot;, &quot;LogEnabled&quot;);</code></td>
<td>The variable <code>bool_value</code> now contains a zero (0).</td>
<td>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.</td>
</tr>
</tbody>
</table>
The variable `bool_value` now contains a one (1).
The function scans only the control group and option set associated with the context name `MVF`.

The function scans only the control group and option set associated with the context name `MVF`.

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

See also
- INI Functions on page 70
- Using INI Options on page 8
- GetINIString on page 259
- LoadINIFile on page 300
**GetINIString**

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

**Syntax**

\[
\text{GetINIString (Context, Group, Option, Default)}
\]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context</td>
<td>(Optional) A name (valid string) associated to a set of INI control groups and options which have been loaded into cache memory.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the control group name (valid string) which contains the INI option string to retrieve.</td>
</tr>
<tr>
<td>Option</td>
<td>Enter the option name (valid string) which contains the INI string value to retrieve. If the control group and option does not contain a string, the system returns a null value.</td>
</tr>
<tr>
<td>Default</td>
<td>(Optional) Enter the default string value to return from the function instead of the actual control group and option value.</td>
</tr>
</tbody>
</table>

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

If you omit the context, the function searches all INI files loaded in memory. If there are multiple control groups and options with the same name, this function returns the first INI control group and option string it finds.

If a context name is present, this function only searches for the control group and option in the set of control groups and options associated with the context name.

**Example**

Assume an INI file (TEST1.INI) was loaded with the context name, MVV. The TEST1.INI file contains this control group and option:

\[
< \text{Control} > \\
\quad \text{Title} = \text{MVV’s group/option}
\]

In addition, the FSIUSER.INI file contains this control group and option:

\[
< \text{Control} > \\
\quad \text{Title} = \text{Test group 1}
\]

Plus, the FSISYS.INI file contains this control group and option:

\[
< \text{Control} > \\
\quad \text{Title} = \text{FAP entry 1}
\]

Based on this scenario, the following table shows and explains several possible results.
<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>string_value = GetINIString (&quot;Control&quot;, &quot;Title&quot;);</code></td>
<td>The variable <code>string_value</code> now contains this string: <em>Test group 1</em></td>
<td>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.</td>
</tr>
<tr>
<td><code>string_value = GetINIString (&quot;MVF&quot;, &quot;Control&quot;, &quot;Title&quot;);</code></td>
<td>The variable <code>string_value</code> now contains this string: <em>MVF's group/option</em></td>
<td>The function scans only the control group and option set associated with the context name <em>MVF</em>.</td>
</tr>
<tr>
<td><code>string_value = GetINIString (&quot;MVF&quot;, &quot;Control&quot;, &quot;Title&quot;, &quot;Bob's group/option&quot;);</code></td>
<td>The variable <code>string_value</code> now contains this string: <em>MVF's group/option</em></td>
<td>The function scans only the control group and option set associated with the context name <em>MVF</em>.</td>
</tr>
</tbody>
</table>

See also

INI Functions on page 70
Using INI Options on page 8
GetINIBool on page 257
GetListElem

Use this XML function to retrieve list elements.

Syntax

GetListElem (%xXMLTree, SrchCriteria)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%xXMLTree</td>
<td>Enter a list type DAL variable that passes the XML tree handle.</td>
</tr>
<tr>
<td>SrchCriteria</td>
<td>Enter a string type DAL variable that passes the search criteria. The search criteria can be a node name, followed by up to five pairs of attribute names and values.</td>
</tr>
</tbody>
</table>

If successful, the system returns a text string which contains the first element that matches the search criteria.

Example

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

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

See also XML Functions on page 89
**GETOVFLWSYM**

Use this function to retrieve the value stored in an overflow symbol. This is value that would be used during the next Documaker Server record overflow operation.

**Syntax**

```
GetOvFlwSym (Form, Symbol)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form</td>
<td>Enter the name of the form that contains the fields on which overflow processing will occur.</td>
</tr>
<tr>
<td>Symbol</td>
<td>Enter the name you want to use as the overflow symbol.</td>
</tr>
</tbody>
</table>

The system returns the value contained in the specified overflow symbol.

**Example**

Here is an example:

```
#content = GetOvFlwSym ("CP0101NL", "Loc_Cnt")
```

In this example, the DAL integer variable, `#content`, would be set to the value of the overflow symbol, `Loc_Cnt`.

**See also**

- AddOvFlwSym on page 127
- IncOvFlwSym on page 280
- ResetOvFlwSym on page 361
- Documaker Server Functions on page 58
**GetValue**

Use this function to return a string that contains the contents of the DAL symbolic variable specified by the parameter. You can use this function when the name of the DAL variable is also stored in a variable, such as when a variable has to be addressed in another external script.

**Syntax**

`GetValue (Symbol)`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbol</td>
<td>Enter a string that specifies the name of a DAL symbolic variable. This can be from an expression or from another string variable.</td>
</tr>
</tbody>
</table>

**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":

```plaintext```
variable_name = "my_variable"
contents = GetValue(variable_name)
```

This example stores the value, 20, in the field entitled 'total # of vehicles' in the current section:

```plaintext```
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.

```plaintext```
If Exists("tbl_" & #line) Then
  Return ( GetValue("tbl_" & #line) )
Else
  Return (" ")
End
```

**See also**

- `Exists` on page 230
- Miscellaneous Functions on page 73
**GROUPNAME**

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

**Syntax**

```
GroupName (Count, StartGroup)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>An index reference to locate a group before or after the specified group. Enter a negative number to move backwards. The default is zero (0).</td>
</tr>
<tr>
<td>StartGroup</td>
<td>Enter the name of a group from which to start the search. The default is the current group.</td>
</tr>
</tbody>
</table>

The system returns the name of the group it located.

Keep in mind you can use an asterisk (*) as the object name to match parent objects. This lets you find objects without explicitly knowing the parent names.

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

The system returns the name of a group of forms relative to another group. If you omit the parameters, the system returns the name of the current group.

The count parameter tells the system to move forward or backwards from a located group before returning the group name.

If it cannot find the starting group cannot or the count parameter causes it to move beyond the number of groups contained in the document set, the system returns an empty string.

Groups are unique within a document set.

**Example**

Here are some examples:

(Assume the current group is GROUPONE.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupName()</td>
<td>GROUPONE</td>
<td>No parameters will result in returning the current group name.</td>
</tr>
<tr>
<td>GroupName(-1)</td>
<td></td>
<td>Returns the name of the group before the current group.</td>
</tr>
</tbody>
</table>

**See also** Name Functions on page 74
Use this function to retrieve the contents of a GVM variable.

**Syntax**

\[ \text{GVM (Name, Instance)} \]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter the name of the GVM variable.</td>
</tr>
<tr>
<td>Instance</td>
<td>Enter the instance number of the GVM variable. The default is one (1).</td>
</tr>
</tbody>
</table>

The system returns the content of the variable if it exists or a blank string if it does not.

**Example**

Here is an example:

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>If (HaveGVM('Company')) AddComment(GVM('Company')) End</td>
<td>String or a blank string</td>
<td>Return the content of the GVM variable &quot;company&quot; if it exist.</td>
</tr>
</tbody>
</table>

**NOTE:** If the GVM variable does not exist, you will receive the error message: DM12041.

**See also**

- Documaker Server Functions on page 58
- HaveGVM on page 270
- AddComment on page 117
- DAL Script Examples on page 35
- SetGVM on page 381
**HAVEFIELD**

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

**Syntax**

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

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

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

Keep in mind you can use an asterisk (*) as the object name to match parent objects. This lets you find objects without explicitly knowing the parent names. For instance, if you enter

```
HaveField("FIELD", , "**")
```

The system will find the field named `FIELD` on any form within the current group. This works because the asterisk in the form name position indicates that any form will do.

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

The system searches for the specified field on a particular section, form, and/or group. If the field is located, one (1) is returned. Otherwise, zero (0) is returned.

Although the return value from some of the other field’s functions might be used to determine the availability of a certain field, this function merely locates the field and does not change or query any particular information about the field.

**Example**

Here are some examples:

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return(HaveField( ))</td>
<td>1</td>
<td>If this script is associated with an entry field, it will always return one (1) if no parameters are provided.</td>
</tr>
<tr>
<td>Return(HaveField (&quot;Second&quot;))</td>
<td>1 or 0</td>
<td>The current section will be searched for the field. A one (1) is returned if located.</td>
</tr>
<tr>
<td>Return(HaveField (&quot;Third&quot;, &quot;FRM&quot;))</td>
<td>1 or 0</td>
<td>Locate the form specified within the current form group. Then locate Third anywhere on that form. If found, a one (1) is returned.</td>
</tr>
</tbody>
</table>
See also

Have Functions on page 69
Field Formats on page 62
Locating Fields on page 64
**HAVEFORM**

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

**Syntax**

```
HaveForm (Form, Group)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form</td>
<td>Enter the name of a form. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of a group to contain the specified form. The default is the current group.</td>
</tr>
</tbody>
</table>

The system optionally returns one (1) if the form is located or zero (0) if it cannot be found.

Keep in mind you can use an asterisk (*) as the object name to match parent objects. This lets you find objects without explicitly knowing the parent names.

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

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

The function does not require any parameters. However, calling it in this manner will typically return 1, since it will locate the current form.

**Example**

Here are some examples:

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>HaveForm(&quot;Form&quot;)</td>
<td>1 or 0</td>
<td>Attempts to locate the named form. If found, returns 1.</td>
</tr>
<tr>
<td>HaveForm(&quot;Form\3&quot;, &quot;GRP&quot;)</td>
<td>1 or 0</td>
<td>Locates the third occurrence of the file named Form within the specified group. If found, returns 1.</td>
</tr>
</tbody>
</table>

**See also**

Have Functions on page 69
**HAVEGROUP**

Use this function to determine if a given group is part of a document.

**Syntax**

```
HaveGroup (Group)
```

**Parameter** | **Description**
--- | ---
Group | Enter the name of a group to locate. The default is the current group.

The system returns one (1) if the group is located and zero (0) if it cannot be found.

Keep in mind you can use an asterisk (*) as the object name to match parent objects. This lets you find objects without explicitly knowing the parent names.

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

Several DAL functions can return values that indicate a group is or is not a part of the document. However, those functions also intend to perform some other procedure other than searching for the group. The HaveGroup function simply identifies whether a given group is present in the document.

The function does not require any parameters. However, calling it in this manner will typically return 1, since it will locate the current group.

**Example**

Here is an example:

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>HaveGroup ( &quot;GRP&quot;)</code></td>
<td>1 or 0</td>
<td>Returns one (1) if the identified group is a part of the document.</td>
</tr>
</tbody>
</table>

**See also**

[Have Functions on page 69](#)
**HaveGVM**

Use this function to determine if a GVM variable exists.

**Syntax**

```
HaveGVM (Name, Instance)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter the name of the GVM variable.</td>
</tr>
<tr>
<td>Instance</td>
<td>Enter the instance number of the GVM variable. The default is one (1).</td>
</tr>
</tbody>
</table>

The system returns one (1) if it locates the GVM variable or a zero (0) if it cannot find the variable.

**Example**

Here is an example:

```
If (HaveGVM('Company'))
  AddComment( GVM('Company'))
End
```

1 or 0

If a GVM variable "company" exist; then add the content of the GVM variable to the print stream.

**See also**

- Documaker Server Functions on page 58
- GVM on page 265
- AddComment on page 117
- DAL Script Examples on page 35
- SetGVM on page 381
- GVM on page 265
**HAVEIMAGE**

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

**Syntax**

\[ \text{HaveImage(Section, Form, Group)} \]

**Parameter** | **Description**
---|---
Section | Enter the name of a section to locate. The default is the current section.
Form | Enter the name of an form that is assumed to contain the specified section. The default is the current form.
Group | Enter the name of a group to contain the specified section or form. The default is the current group.

The system returns one (1) if the form is located and zero (0) if it cannot be found.

Keep in mind you can use an asterisk (*) as the object name to match parent objects. This lets you find objects without explicitly knowing the parent names.

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

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

The function does not require any parameters. However, calling it in this manner will typically return 1, since it will locate the current section.

**Example**

Here are some examples:

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>HaveImage(&quot;IMG&quot;)</td>
<td>1 or 0</td>
<td>Attempts to locate the named section on the current form. If found, return 1.</td>
</tr>
<tr>
<td>HaveImage(&quot;IMG\2&quot;, &quot;Form\3&quot;, &quot;GRP&quot;)</td>
<td>1 or 0</td>
<td>Locate the third occurrence of Form within the specified group. If found, then locate the second occurrence of IMG. If successful, return 1.</td>
</tr>
</tbody>
</table>

**See also**

*Have Functions on page 69*

*Where DAL Functions are Used on page 97*
**HAVELOGO**

Use this function to determine if a graphic (LOG) exists on a section or form which is in the current form set.

**Syntax**

```plaintext
HAVELOGO (Graphic, Section, Form, Group)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphic</td>
<td>Enter the name of the graphic you want to find. Graphic names are assigned in Studio or Image Editor.</td>
</tr>
<tr>
<td>Section</td>
<td>Enter the name of a section that contains the graphic you specified. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of a form that contains the section you specified. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of a group to use to locate the graphic. The default is the current group.</td>
</tr>
</tbody>
</table>

The system returns one (1) if it finds the graphic and zero (0) if it does not.

Keep in mind you can use an asterisk (*) as the object name to match parent objects. This lets you find objects without explicitly knowing the parent names.

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

**Example**

Here are some examples:

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAVELOGO( &quot;Log1&quot;)</td>
<td>1 or 0</td>
<td>Determines if Log1 exists on the current section, form, group.</td>
</tr>
<tr>
<td>HAVELOGO(&quot;Log1&quot;, &quot;IMH1\3&quot;,&quot;UpRate&quot;)</td>
<td>1 or 0</td>
<td>Determines if Log1 exists on the 3rd occurrence of the section, IMH1, on the form, UpRate, within the default group.</td>
</tr>
</tbody>
</table>

**See also**

- ChangeLogo on page 163
- DelLogo on page 214
- HaveField on page 266
- HaveForm on page 268
- HaveGroup on page 269
- HaveImage on page 271
- InlineLogo on page 282
- Logo on page 303
RenameLogo on page 359
Have Functions on page 69
**HAVERECIP**

Use this function to see if the specified recipient name is defined in the form set for the specified section, form, or group.

You can use this function along with the RecipientName function in DAL scripts to place a sequence number on each page of each recipient batch.

**Syntax**

```
HaveRecip (Recipient, Section, Form, Group)
```

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

The system returns one (1) if true or zero (0) if false.

**NOTE:** You must enter a recipient name.

**See also**
- RecipientName on page 355
- Have Functions on page 69
**Hex2Dec**

Use this function to return the integer equivalent of a hexadecimal string.

**Syntax**

\[
\text{Hex2Dec} \ (\text{Value1})
\]

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value1</td>
<td>This parameter specifies a string of characters you want converted into an integer value. If the string value does not represent a valid hexadecimal number, the results are questionable and can result in only part of the value being converted.</td>
</tr>
</tbody>
</table>

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:

\[
y = \text{"1A2B"}
\]

\[
z = \text{Hex2Dec}(y)
\]

\[
\text{Result is } z = 6699
\]

\[
y = \text{"FF00"}
\]

\[
z = \text{Hex2Dec}(y)
\]

\[
\text{Result is } z = 65280
\]

**See also**

Dec2Hex on page 206

Bit/Binary Functions on page 42
**Hour**

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

**Syntax**

\[
\text{Hour (Time1, Format1)}
\]

**Parameter** | **Description**
---|---
Time1 | Enter a valid time string. Assumed to be in the format specified by the next parameter. The default is the current time.
Format1 | Enter a valid time format string. Describes the first parameter (time1). The default is time format 1 (HH:MM:SS).

**Example**

Here are some examples:

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return(Hour())</td>
<td>3</td>
<td>Defaults to the current time and extracts 3.</td>
</tr>
<tr>
<td>Return(Hour(&quot;9:50:20AM&quot;, 2))</td>
<td>9</td>
<td>Reads the given time which is in format 2 and extracts 9.</td>
</tr>
</tbody>
</table>

**See also**  
Time Formats on page 80
**ImageName**

Use this function to get the name of a section. This name is returned.

**Syntax**

```
ImageName (Count, StartImage, Form, Group)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Enter an index reference to locate a form before or after the specified form. The default is zero (0).</td>
</tr>
<tr>
<td>StartImage</td>
<td>Enter the name of a section from which to begin the search. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of a form containing the requested section. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of a group to contain the specified form. The default is the current group.</td>
</tr>
</tbody>
</table>

Keep in mind you can use an asterisk (*) as the object name to match parent objects. This lets you find objects without explicitly knowing the parent names.

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

The system returns the name of a section relative to another section on the same form. If no parameters are provided to this function, the current section’s name is returned. The Count parameter tells the system to move a number of sections forwards or backwards (negative) from a located section before returning the section name.

If the starting section cannot be located or the Count parameter causes the system to move beyond the number of sections contained on the form, the system returns an empty string.

If there is more than one copy of a section on the located form, the name returned will contain the occurrence notation used by DAL functions to locate sections. For instance, a name like IMG\2 identifies the second copy of IMG on a particular form.

**Example**

Here are some examples:

(Assume the current section is named `IMG`.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImageName( )</td>
<td>IMG</td>
<td>No parameters will result in returning the current section name.</td>
</tr>
<tr>
<td>ImageName( 2, &quot;IMG&quot;, &quot;FormC&quot;)</td>
<td></td>
<td>Locate FORMC in the current group. Next, locate IMG on that form. Then, return the name of the section two positions beyond the located section.</td>
</tr>
</tbody>
</table>

**See also** Name Functions on page 74
**IMAGE_RECT**

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

**Syntax**

```
IMAGE_RECT (PrefixVariable, Section, Form, Group)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrefixVariable</td>
<td>Enter the coordinates for the section.</td>
</tr>
<tr>
<td>Section</td>
<td>Enter the name of a section in the form set. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of the form that contains the section. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of the form group that contains the form and section. The default is the current form group.</td>
</tr>
</tbody>
</table>

This procedure gets the coordinates for the section and stores them in the defined variable names. If the prefix name variables do not exist in DAL, the system creates them. The system creates four internal variables: `prefix name.top`, `prefix name.left`, `prefix name.bottom`, and `prefix name.right`. If these variables exist, the system modifies them with the new coordinates.

**Example**

For these examples, assume the prefix name is `MyImage`, the current section is `Image25`, the form is `Input_form`, and the form group is `package1`. The coordinates are:

<table>
<thead>
<tr>
<th>Image25</th>
<th>Image50</th>
</tr>
</thead>
<tbody>
<tr>
<td>top</td>
<td>25</td>
</tr>
<tr>
<td>left</td>
<td>50</td>
</tr>
<tr>
<td>bottom</td>
<td>100</td>
</tr>
<tr>
<td>right</td>
<td>200</td>
</tr>
</tbody>
</table>

Here are some examples:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| IMAGE_RECT ("MyImage") | Internal variables equal:  
  MyImage.top=25  
  MyImage.left=50  
  MyImage.bottom=100  
  MyImage.right=200  |
<p>|                   |        | The procedure returns the coordinates for the current section (<em>Image25</em>) on the current form in the current form group. If it does not exist, the procedure returns zero (0). |</p>
<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMAGERECT (&quot;MyImage&quot;, &quot;Image50&quot;)</td>
<td>Internal variables equal:</td>
<td>The procedure returns the coordinates for Image50 on the current form in the current form group. If it does not exist, the procedure returns zero (0).</td>
</tr>
<tr>
<td></td>
<td>MyImage.top=125</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MyImage.left=150</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MyImage.bottom=200</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MyImage.right=200</td>
<td></td>
</tr>
<tr>
<td>IMAGERECT (&quot;m&quot;, &quot;MVF2&quot;, &quot;XYZ&quot;)</td>
<td>Internal variables equal:</td>
<td>Gets and stores the coordinates for the second occurrence of the section MVF on the form XYZ into the DAL target variables. If it does not exist, the procedure returns zero (0).</td>
</tr>
<tr>
<td></td>
<td>m.top = 75</td>
<td></td>
</tr>
<tr>
<td></td>
<td>m.left = 125</td>
<td></td>
</tr>
<tr>
<td></td>
<td>m.bottom = 300</td>
<td></td>
</tr>
<tr>
<td></td>
<td>m.right = 225</td>
<td></td>
</tr>
</tbody>
</table>

See also  
Section Functions on page 77  
SetImagePos on page 382
**INCOVFLOWSYM**

Use this procedure/function to increment an overflow symbol. This procedure provides DAL with the Documaker Server equivalent to the IncOvFlwSym rule, with the exception that it will only increment by one.

**Syntax**

```
IncOvFlwSym (Form, Symbol)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form</td>
<td>Enter the name of the form that contains the fields on which overflow processing will occur.</td>
</tr>
<tr>
<td>Symbol</td>
<td>Enter the name you want to use as the overflow symbol.</td>
</tr>
</tbody>
</table>

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

This procedure increments the value contained in the specified overflow symbol.

**Example**

Here is an example:

```
rc = IncOvFlwSym ("CP0101NL", "Loc_Cnt")
```

In this example, the overflow symbol, *Loc_Cnt* is incremented and the DAL integer variable, # rc, is set to one (1) on success or zero (0) on failure.

**Syntax**

- AddOvFlwSym on page 127
- GetOvFlwSym on page 262
- ResetOvFlwSym on page 361
- Documaker Server Functions on page 58
INI

Use this function to get the value of an INI option from the currently loaded INI files. If there is more than one occurrence of a control group and option in the various INI files the system uses, like the FSIUSER.INI and the FSISYS.INI files, this function uses the values in the first control group and option it finds that matches the criteria you enter. The system usually first loads the FSIUSER.INI file, which tells it to then load the FSISYS.INI file.

Syntax
INI (Group, Option, Default)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group</td>
<td>Enter the name of the INI control group name (valid string) which contains the INI option string you want to retrieve.</td>
</tr>
<tr>
<td>Option</td>
<td>Enter the name of the INI option (valid string) which contains the INI string value you want to retrieve. If the control group and option do not contain a string, the system returns a null value.</td>
</tr>
<tr>
<td>Default</td>
<td>(Optional) The default string value to return from the function instead of the actual control group and option value.</td>
</tr>
</tbody>
</table>

The system retrieves the specified control group and option string. The system returns one (1) if no errors occur and zero (0) if errors occur.

Example
This example:

INI("UserInfo","File")

retrieves the name of the user information file, as stored in this control group:

<UserInfo>
File =

See also
INI Functions on page 70
Using INI Options on page 8
GetINIBool on page 257
GetINIString on page 259
**INLINELOGO**

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

**Syntax**

```
INLINELOGO (Graphic, Option, Section, Form, Group)
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphic</td>
<td>Enter the name of the graphic to be in-lined in the print stream. Graphic names are assigned in Studio or Image Editor.</td>
</tr>
<tr>
<td>Option</td>
<td>This parameter sets the inline flag. You can choose from these options: One (1) equals On Zero (0) equals Off The default is one (1).</td>
</tr>
<tr>
<td>Section</td>
<td>Enter the name of a section that contains the specified graphic. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of a form that contains the section. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of a group to use to locate the specified object. The default is the current group.</td>
</tr>
</tbody>
</table>

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

**Example**

Here are some examples:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>INLINELOGO(&quot;Log1&quot;)</td>
<td>1 or 0</td>
<td>In-lines Log1 (on the current section, form, and group) into the print stream.</td>
</tr>
<tr>
<td>INLINELOGO(&quot;Log1&quot;, 1,&quot;IMH1\3&quot;,&quot;UpRate&quot;)</td>
<td>1 or 0</td>
<td>In-lines Log1 (on the 3rd occurrence of the named section, IMH1, on the form, UpRate) into the print stream.</td>
</tr>
</tbody>
</table>

**See also**

- ChangeLogo on page 163
- DelLogo on page 214
- HaveLogo on page 272
- Logo on page 303
- RenameLogo on page 359
- Graphics Functions on page 71
**Input**

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

**Syntax**

```
Input (Prompt, Title, Length, DefText)
```

**Parameter** | **Description**
--- | ---
Prompt | Enter a text string to assign as the prompt for the field. The default is Text.
Title | Enter a text string to assign as the title of the window. The default is Title.
Length | Enter the maximum input text length. The default is set by Windows.
DefText | Enter a text string to assign as the default input data.

The system returns the input results.

This function creates a window you can use to gather information from a user. The text entered through the window is returned as a string. If no text is assigned, or if the user closes the window without choosing Ok, the returned string will be empty.

**Example**

Here are some examples:

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>NAME = Input (&quot;Please enter your name: &quot;, &quot;Name Entry&quot;); Return(Name)</code></td>
<td>Produces a window requesting input.</td>
<td>The name of the window is Name Entry. The user sees the prompt Please enter your name: If the user selects Cancel, NAME is an empty string. If the user selects Ok, NAME contains the text entered by the user.</td>
</tr>
<tr>
<td><code>Return(Input( ))</code></td>
<td>Produces a window requesting input.</td>
<td>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.</td>
</tr>
<tr>
<td><code>Return(Input (&quot;Confirm this result&quot;, 30, &quot;123.45&quot;))</code></td>
<td>Produces a window requesting input.</td>
<td>This window will have the prompt Confirm this result. The input field accepts up to 30 characters and defaults to “123.45”. There will be no title.</td>
</tr>
</tbody>
</table>

**See also**

Documaker Workstation Functions on page 59
**INSERT**

Use this function to insert a substring into a string at the position you specify. The result string is returned.

**Syntax**

\[ \text{Insert \ (String, Position, SubString)} \]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Enter a valid string. The default is the value of the current field.</td>
</tr>
<tr>
<td>Position</td>
<td>Enter the position in the field to perform the insert. The default is the one (1), the first position.</td>
</tr>
<tr>
<td>SubString</td>
<td>Enter the string that you want to insert.</td>
</tr>
</tbody>
</table>

The system adds the substring to the string you specified in the first parameter at the indicated position. If the position indicated in the second parameter is greater than the length of the original string, the string is increased to the given length before the third parameter is inserted.

If no position is given in the second parameter the insertion begins at position one. If no value is provided for the third parameter (Substring), nothing is inserted.

**Example**

Here are some examples:

(Assume the current field contains the text *Your Name.*)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return(Insert (, &quot;Type &quot;))</td>
<td>Type Your Name</td>
<td>Defaults to the first position of the current field and inserts Type.</td>
</tr>
<tr>
<td>Return(Insert (, CFIND (, &quot; &quot;, &quot; First&quot;))</td>
<td>Your First Name</td>
<td>First the CFind function locates a blank space at position 5 within Your Name. Next, First is inserted at position 5.</td>
</tr>
<tr>
<td>Return(Insert (&quot;Complete blank.&quot;, 10, &quot;every &quot;))</td>
<td>Complete every blank.</td>
<td>Goes to position 10 and inserts every.</td>
</tr>
<tr>
<td>Return(Insert (&quot;Complete blanks&quot;, 17, &quot;with black ink.&quot;))</td>
<td>Complete blanks with black ink.</td>
<td>Increases the length of the field to 17 and appends with black ink.</td>
</tr>
</tbody>
</table>

**See also**

String Functions on page 78  
CFind on page 162
**INT**

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

**Syntax**

```plaintext
INT (Number)
```

**Parameter** | **Description**
--- | ---
Number | Enter a valid numeric data type. The default is the integer value of the current field.

The system returns the integer value of a number.

The decimal portion of the number is truncated. The number is not rounded up or down. The sign of the number is not changed.

**Example**

Here are some examples:

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT(-101.99)</td>
<td>-101</td>
<td>Defaults to the current field.</td>
</tr>
<tr>
<td>$TEMP = 99.99</td>
<td>99</td>
<td>After executing these statements, $TEMP will be 99.99 and #RESULT will be 99, without a decimal.</td>
</tr>
<tr>
<td>#RESULT = INT($TEMP)</td>
<td>2</td>
<td>The parameter value will equate to 2.5 The INT function will truncate this result to 2. The function does not round.</td>
</tr>
</tbody>
</table>

See also [Mathematical Functions on page 72](#)
**IsPrintObject**

Use this function during banner processing or in another print operation to determine if the section (image), form, or group is printable. This determination is based on the current print recipient and the recipient copy count.

**Syntax**

```
IsPrintObject (Section, Form, Group)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section</td>
<td>Enter the name of the section you want to check. If you omit this parameter, the system uses the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of the form you want to check. If you omit this parameter, the system uses the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of the group you want to check. If you omit this parameter, the system uses the current group.</td>
</tr>
</tbody>
</table>

**NOTE:** You can use this function outside of a print operation to determine if a section is printable, but a true (1) result is not a guarantee the section will print during the next print operation.

**Example**

Here is an example:

```
IsPrintObject();
```

This example checks the current section on the current form in the current group and returns a one (1) if that section is printable or a zero (0) if it is not.

**See also**  
Printer and Recipient Functions on page 76
**IsXMLError**

Use this function to check the list for error status.

**Syntax**

```plaintext
IsXMLError (%xXMLTree, SrchCriteria)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%xXMLTree</td>
<td>Enter a list type DAL variable that passes the XML tree handle.</td>
</tr>
<tr>
<td>SrchCriteria</td>
<td>Enter a string type DAL variable that passes the search criteria. The search</td>
</tr>
<tr>
<td></td>
<td>criteria can be a node name, followed by up to five pairs of attribute names</td>
</tr>
<tr>
<td></td>
<td>and values.</td>
</tr>
</tbody>
</table>

The system returns one (1) if no errors occur or zero (0) if errors occur.

See also  

XML Functions on page 89
**JCenter**

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

**NOTE:** To justify a display item, such as a field, on a fixed point use the JustField function. The JCenter function is for padding a text string so it will appear centered within a given string length.

**Syntax**

\[
\text{JCenter} \ (\text{String}, \text{Length})
\]

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Enter a valid string. The default is the value of the current field.</td>
</tr>
<tr>
<td>Length</td>
<td>Specify the desired length of output. The default is the length of the input string.</td>
</tr>
</tbody>
</table>

The system justifies the text characters of the string parameter within the specified length and returns the new string.

If the length specified in the Length parameter is longer than the string, the result will be increased to the given length before the system centers the string. If the length specified is less than the string, the length of the string is used.

For example, if the variable field has a length of 30, the DAL script says Return(JCenter (,10)), and you enter ABC in the variable field, the system will center ABC using a length of 10 instead of 30.

**Example**

Here are some examples:

(Assume the current field contains the text *Name* and can be up to 20 characters.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>JCenter ( , Size ( ) )</td>
<td>“ Name “</td>
<td>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.</td>
</tr>
<tr>
<td>JCenter (&quot;Complete blanks.&quot;, 5)</td>
<td>Complete blanks.</td>
<td>Ignores the specified length (5) because it is less than the given string.</td>
</tr>
<tr>
<td>JCenter (&quot;Complete blanks.&quot;, 25)</td>
<td>“ Complete blank. “</td>
<td>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.</td>
</tr>
</tbody>
</table>

See also

JustField on page 291

String Functions on page 78

Size on page 390
**JLeft**

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

**Syntax**

```
JLeft (String, Length)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Enter a valid string. The default is the value of the current field.</td>
</tr>
<tr>
<td>Length</td>
<td>Specify the desired length of output. The default is the length of the input string.</td>
</tr>
</tbody>
</table>

The system left justifies the text characters of the string parameter within the specified length and returns the new string.

If the length specified in the length parameter is longer than the string, the result will be increased to the given length before the justification. If the length specified is less than the string, the length of the string is used.

**Example**

Here are some examples:

(Assume the current field contains the text *Name* and can be up to 20 characters.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>JLeft (&quot;Heading&quot;, 20)</td>
<td>“Heading”</td>
<td>Left justifies the text within a length of 20 spaces.</td>
</tr>
<tr>
<td>JLeft (&quot;Complete blanks. &quot;, 5)</td>
<td>“Complete blanks. ”</td>
<td>Ignores the specified length (5) because it is less than the given string.</td>
</tr>
<tr>
<td>JLeft ( , Size ( ) &amp; &quot;X&quot;)</td>
<td>“Name X”</td>
<td>First the Size function determines that the maximum length of the field is 20. Then X is added to the end of the field. There are 15 spaces between the end of the word Name and the X.</td>
</tr>
</tbody>
</table>

**See also**

- *String Functions on page 78*
- *@ on page 109*
- *Size on page 390*
**JRight**

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

### Syntax

\[
\text{JRight} \left( \text{String}, \text{Length} \right)
\]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Enter a valid string. The default is the value of the current field.</td>
</tr>
<tr>
<td>Length</td>
<td>Specify the desired length of output. The default is the length of the input string.</td>
</tr>
</tbody>
</table>

The system justifies the text characters of the string parameter within the specified length and returns the new string.

If the length you specify in the Length parameter is longer than the string, the result is increased to the given length before the text is justified.

If the length specified is less than the string, the system uses the length of the string.

### Example

Here are some examples:

(Assume the current field contains the text *Name* and can be up to 20 characters.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>JRight (&quot;Heading&quot;, 20)</td>
<td>“Heading”</td>
<td>Increases the size of the field to 20 and right justifies the text.</td>
</tr>
<tr>
<td>JRight (&quot;Complete blanks. &quot;, 5)</td>
<td>“Complete blanks. ”</td>
<td>Ignores the specified length (5) because it is less than the given string.</td>
</tr>
<tr>
<td>JRight (, SIZE () &amp; &quot;!&quot;)</td>
<td>“Name!”</td>
<td>First the Size function determines that the maximum length of the field is 20. Then the original text in the field is right justified and an exclamation point (!) is concatenated after Name.</td>
</tr>
</tbody>
</table>

**NOTE:** If you are aligning decimal numbers, be sure to use a fixed or non-proportional font, such as Courier.

### See also

- String Functions on page 78
- @ on page 109
- Size on page 390
**JUSTFIELD**

Use this procedure/function to justify (left, right, or center) a variable field content by modifying its field coordinates.

**NOTE:** To pad a text string so it will appear centered within a given string length, use the JCenter function. The JustField function is for justifying display items, such as fields, on a fixed point.

**Syntax**

```
JustField (Mode, Xcoordinate, Justification, Field, Section, Form, Group)
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>Enter L (left), R (right), or C (center). The default is L.</td>
</tr>
</tbody>
</table>
| Xcoordinate  | Enter the X coordinate used to align the field. If Mode is R, this will be zero (0), the right-most position of the field. If Mode is C, this will be the center of the field. Here is an example:  
```
"R", 5000
```
If the data is 12345, the character 5 will be positioned at 5000 FAP units. |
| Justification| Enter a character found in the data to use to align the field. The procedure aligns the field so the character you specify overlays the X coordinate. You must define the X-coordinate parameter when using the justification character. If you omit the X-coordinate the system runs as if the justification character was not specified. 
Here is an example:  
```
R,5000,"."
```
If the data is 123.45, then the decimal point will be positioned at 5000 FAP units. |
| Field        | Enter the name of the field. The default is the current field.              |
| Section      | Enter the name of the section that contains the field. The default is the current section. |
| Form         | Enter the name of the form that contains the section and/or field. The default is the current form. |
| Group        | Enter the name of the group that contains the form, section, and/or field. The default is the current group. |
Example

This example centers the original address lines data in the section, QJUSTFIELD2, at 10,000 FAP units.

```java
JustField("C",10000,"line 1", "qjustfield2")
JustField("C",10000,"line 2", "qjustfield2")
JustField("C",10000,"line 3", "qjustfield2")
```

Here is an example:

| line 1 | Oracle Insurance |
| line 2 | Atlanta, GA 30339-4000 |
| line 3 | 404.439.5500 |

<table>
<thead>
<tr>
<th>5,000 FAP units</th>
</tr>
</thead>
<tbody>
<tr>
<td>line 1</td>
</tr>
<tr>
<td>line 2</td>
</tr>
<tr>
<td>line 3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>10,000 FAP units</th>
</tr>
</thead>
<tbody>
<tr>
<td>line 1</td>
</tr>
<tr>
<td>line 2</td>
</tr>
<tr>
<td>line 3</td>
</tr>
</tbody>
</table>

This example justifies the original line data (left aligned at 5,000 FAP units) on the decimal point at 10,000 FAP units.

```java
JustField("C",10000,".","line 1")
JustField("C",10000,".","line 2")
```

Here is an example:

| line 1 | 5,000.00 |
| line 2 | 12345.888888 |

<table>
<thead>
<tr>
<th>5,000 FAP units</th>
</tr>
</thead>
<tbody>
<tr>
<td>line 1</td>
</tr>
<tr>
<td>line 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>10,000 FAP units</th>
</tr>
</thead>
<tbody>
<tr>
<td>line 1</td>
</tr>
<tr>
<td>line 2</td>
</tr>
</tbody>
</table>

See also

JCenter on page 288
Field Functions on page 61
**KickToWIP**

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

**Syntax**

```plaintext
KickToWIP ( )
```

There are no parameters for this function.

Use the `ShowWIPWarning` option to suppress the Sent to Manual Batch warning messages:

```plaintext
< RunMode >
    ShowWIPWarning = No
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ShowWIPWarning</td>
<td>Enter No to suppress warning messages included the error logs when using the KickToWIP DAL function. The default is Yes, which tells the system to include the messages in the error logs.</td>
</tr>
</tbody>
</table>

**Example**

Here is an example of how you would set your AFGJOB.JDT file:

```plaintext
<Base Form Set Rules>
    ;NoGenTrnTransactionProc;;;
    ...
    ;WriteOutput;;;
    ;WriteNaFile;;;
    ;PostTransDAL;;KickToWIP( );

In this example, the PostTransDAL function sets the Manual batch flag before the NA, POL, and Receipt batch files are written. Here is an example of the section-level rules:

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

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

Here is an example of the field-level rules:

```plaintext
;0;0;area1;0;area1;0;0;;DAL;Call(“Chk_If_Kick”);N;N;N;N;919;6736;12112;
```

In this example, when the Areal field is executed the system calls the DAL script named `Chk_If_Kick`. The DAL script checks for the presence of two conditions and if true, sets the Manual batch flag for the transaction.

Here is an example of the Chk_If_Kick DAL script:

```plaintext
BeginSub Chk_If_Kick
    If (CountRec(“1,Second_Address”) = 0) AND \ 
        (GetData(“1,Second_party, 45,1) = “X”) Then
        KickToWIP( )
```

Option Description
---
ShowWIPWarning | Enter No to suppress warning messages included the error logs when using the KickToWIP DAL function. The default is Yes, which tells the system to include the messages in the error logs.
End

EndSub

**NOTE:** You must execute this DAL function before the ConvertWIP form set level rule is executed, if it is included in the AFGJOBJDT file

See also  [Documaker Server Functions on page 58](#)
**LeapYear**

Use this function to find out whether or not the specified year is a leap year.

**Syntax**

```
LeapYear (Year)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
<td>Enter the year. You can enter either a two- or four-digit number. If you enter a two-digit number, the current century is added to create the year value. The default is the current year.</td>
</tr>
</tbody>
</table>

The system returns one (1) if the year is a leap year and zero (0) if it is not.

This function is most often used with the Year function. The Year function extracts the year number from a given date.

**Example**

Here are some examples:

(Assume the current date is 07/01/08.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeapYear ( )</td>
<td>1</td>
<td>The parameter defaults to the current year (2008). Since 2008 is a leap year, one (1), which represents true, is the result.</td>
</tr>
<tr>
<td>LeapYear (07)</td>
<td>0</td>
<td>The year 2007 was not a leap year. Therefore, the result is zero (0), representing false.</td>
</tr>
<tr>
<td>LeapYear (Year(“2009/09/09”, “34”) )</td>
<td>0</td>
<td>First the Year function extracts the year number (2009) from the date, which is given in the date format &quot;34&quot;. Then LeapYear determines that 2009 is not a leap year and returns zero (0.)</td>
</tr>
</tbody>
</table>

**See also**

- Using INI Options on page 8
- Date Formats on page 52
- Year on page 448
- Date Functions on page 51
**Left**

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

**Syntax**

`Left (String, Length)`

**Parameter** | **Description**
---|---
String | Enter a valid string. The default is the value of the current field.
Length | Specify the desired length of output. The default is the length of the input string.

The system returns a string equivalent to the given length from the left portion of the string.

The input string is trimmed of leading and trailing spaces. If the length specified in the second parameter exceeds the length of the string, the result is increased to the given length.

**Example**

Here are some examples:

(Assume the current field contains the text `Your Name` and can be up to 20 characters.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left ( )</td>
<td><code>Your Name</code></td>
<td>Defaults to the current field and returns the full length of the field.</td>
</tr>
<tr>
<td>Left (&quot;Complete blanks.&quot;, 5)</td>
<td><code>Compl</code></td>
<td>Default to position one (1) and returns the first five characters.</td>
</tr>
<tr>
<td>Left (&quot; final payment&quot;, 13)</td>
<td>&quot;final payment&quot;</td>
<td>Trims the field of leading spaces and returns 13 characters.</td>
</tr>
</tbody>
</table>

**See also**

Right on page 363

String Functions on page 78
LEN

Use this function to return the length of the specified string. The length includes all characters, including leading and trailing spaces.

Syntax

LEN (String)

Parameter | Description
--- | ---
String | Enter a valid string. The default is the value of the current field.

This function is often confused with the Size function. The LEN function returns the length of the actual data contained in a text string, including leading and trailing spaces. The Size function returns the length of the defined data area for a section field.

Example

Here are some examples:

(Assume the current field contains the text Your Name.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEN ()</td>
<td>9</td>
<td>Defaults to the current field.</td>
</tr>
<tr>
<td>LEN (&quot; Your Name &quot;)</td>
<td>19</td>
<td>The result includes the leading and trailing spaces of the given field.</td>
</tr>
<tr>
<td>LEN (&quot;Street Address&quot;)</td>
<td>14</td>
<td>Returns the length of the given string.</td>
</tr>
<tr>
<td>LEN (@(&quot;ThisField&quot;) )</td>
<td>8</td>
<td>Finds the variable field named ThisField on the current section and counts the length of the data. The data in this field is Jane Doe, so the number 8 is returned.</td>
</tr>
</tbody>
</table>

See also

String Functions on page 78
@ on page 109
Size on page 390
**ListInList**

Use this function to search for the comma-delimited list specified by the second parameter for each character string in the comma-delimited list specified by the first parameter. If a match is found, the function returns the ordinal position (integer) of the first string in the second parameter that matches any of the strings in the first parameter. If no match is found, the function returns a zero (0).

**Syntax**

\[
\text{ListInList}( \text{StringList}, \text{ListString} )
\]

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StringList</td>
<td>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.</td>
</tr>
<tr>
<td>ListString</td>
<td>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.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>This function statement</th>
<th>Returns</th>
<th>Assuming</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListInList( @(&quot;e_codes&quot;), &quot;ABC,AB,DE,A,GFHI,ABCD&quot; )</td>
<td>1</td>
<td>Field e_codes contains: ABC, A.</td>
</tr>
<tr>
<td>ListInList( GetValue(&quot;e_codes&quot;), &quot;ABC,AB,DE,A,GFHI&quot; )</td>
<td>2</td>
<td>DAL variable, e_codes, contains: AB, ab.</td>
</tr>
<tr>
<td>ListInList( ?(&quot;e_codes&quot;), &quot;ABC,AB,DE,A,GFHI,ABCD&quot; )</td>
<td>3</td>
<td>XDB entry e_codes returns: DE, a.</td>
</tr>
<tr>
<td>ListInList( ?(&quot;e_codes&quot;), ?(&quot;t_codes&quot;), )</td>
<td>4</td>
<td>XDB entry e_codes returns: A. The entry t_codes contains: ABC, AB, DE, A, GFHI, ABCD.</td>
</tr>
<tr>
<td>ListInList( ?(&quot;e_codes&quot;), &quot;ABC,AB,DE,A,GFHI,ABCD&quot; )</td>
<td>0</td>
<td>XDB entry e_codes returns: XYZ.</td>
</tr>
</tbody>
</table>

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.
The return value for the above example returns a four (4) because two spaces exist between the comma and EE.

Keep in mind:

- The search is not case-sensitive. This means A will match a.
- Spaces are considered. This means the system will find no matches in the following examples:

  ListInList("Steel,Wood", "Steel,Aluminum")
  ListInList("Steel,Wood", "Steel ,Aluminum")
  ListInList("Steel,Wood", "Aluminum,Steel ")

See also [String Functions on page 78](#)
**LoadINIFile**

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

**Syntax**

```
LoadINIFile (Context, File)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context</td>
<td>(Optional) A name (valid string) that will be associated to the set of INI control groups and options contained in the physical file.</td>
</tr>
<tr>
<td>File</td>
<td>Enter the name of the INI file to load. If you omit the extension, the system assumes it is <code>INI</code>. The system searches in the current directory, or uses a full path name if you specify one</td>
</tr>
</tbody>
</table>

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

If you specify a context name, that name can be used by other INI functions to reference the loaded set of INI control groups and options.

**Example**

Here are some examples:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoadINIFile (&quot;DALRun&quot;);</td>
<td>The INI control groups and options can now be referenced by executing modules.</td>
<td>The INI file is loaded into cache memory. Execution of this procedure assumes the file extension is <code>INI</code>.</td>
</tr>
<tr>
<td>LoadINIFile (&quot;Run_process&quot;, &quot;DALRun.ini&quot;);</td>
<td>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 <code>Run_process</code>.</td>
<td>The INI file is loaded into cache memory.</td>
</tr>
</tbody>
</table>

**See also**

INI Functions on page 70
Using INI Options on page 8
SaveINIFile on page 370
GetINIBool on page 257
GetINIString on page 259
PutINIBool on page 349
PutINIString on page 351
**LOADLIB**

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

**Syntax**

\[ \text{LoadLib (File)} \]

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>Enter the name of the file which contains the DAL scripts. If you omit the path, the system looks for the file in DefLib. If you omit the extension, the system uses the one defined in the Ext option of the DAL control group in your INI file. You must include the File parameter.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoadLib (&quot;DB_Func&quot;)</td>
<td>The system loads the DB_Func file into cache memory.</td>
<td>Once loaded, you can reference the DAL scripts stored in memory as named subroutines.</td>
</tr>
</tbody>
</table>

**See also**

- Miscellaneous Functions on page 73
- Creating a DAL Script Library on page 5
**LOADXMLList**

Use this function to load an XML document and extract an XML tree.

**Syntax**

```
LoadXMLList (FileName)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileName</td>
<td>Enter the name of the XML file you want to load.</td>
</tr>
</tbody>
</table>

The system returns the XML tree in a list type DAL variable.

**Example**

For an example, see the DAL script in Scenario 2 on page 90.

**See also**

- XML Functions on page 89
- DestroyList on page 216
**Logo**

Use this procedure/function to place a graphic file (LOG) at a specified position in the section.

**Syntax**

Logo (Graphic, Xcoordinate, Ycoordinate, Section, Form, Group)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphic</td>
<td>Enter a valid name for a graphic. Must be a variable field object.</td>
</tr>
<tr>
<td>Xcoordinate</td>
<td>Enter a valid X coordinate location.</td>
</tr>
<tr>
<td>Ycoordinate</td>
<td>Enter a valid Y coordinate location.</td>
</tr>
<tr>
<td>Section</td>
<td>Enter the name of the section name that contains the new graphic. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of the form name that contains the section you specified. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of the group that contains the specified section or form. The default is the current group.</td>
</tr>
</tbody>
</table>

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

This procedure uses FAP units (1 inch = 2400 FAP units). The top-left position of a page represents coordinate (0, 0). To place a graphic an inch from the top and an inch from the left of the page, the X and Y coordinates would be (2400, 2400).

If the location for a particular graphic can be described in relation to a field on the form, you can use the FieldX and FieldY functions to get the coordinates of that field.

This function does not redraw the section display. Use the Refresh procedure with the Logo procedure to view the changes.

**Example**

Here are some examples:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logo (&quot;janedoe&quot;, &quot;7500&quot;, &quot;5500&quot;));Refresh( )</td>
<td>1 or 0</td>
<td>Defaults to add the graphic on the current section at the location specified.</td>
</tr>
<tr>
<td>Logo(&quot;Hancock&quot;, FieldX(&quot;MyField&quot;), FieldY(&quot;MyField&quot;), &quot;IMG&quot;, &quot;FORM&quot;)Refresh( )</td>
<td>1 or 0</td>
<td>First locate the specified form in the current group. Next locate IMG on that form. Finally, add the graphic at the same location as the field, &quot;MyField&quot;.</td>
</tr>
</tbody>
</table>

**See also**

ChangeLogo on page 163
DelLogo on page 214
HaveLogo on page 272
InlineLogo on page 282
FieldX on page 238
FieldY on page 239
Refresh on page 357
RenameLogo on page 359
Graphics Functions on page 71
**LOWER**

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

**Syntax**

```
Lower (String, Length)
```

**Parameter** | **Description**
--- | ---
String | Enter a valid string. The default is the value of the current field.
Length | Specify the desired length of output. The default is the length of the input string.

If the length you specify is longer than the string, the string is increased to the given length. If the specified length is less than the string, the length of the string is used. The string is not truncated.

**Example**

Here are some examples:

(Assume the current field contains the text *Your Name.*)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower ( )</td>
<td>&quot;your name&quot;</td>
<td>Defaults to the current field</td>
</tr>
<tr>
<td>Lower (&quot;Street Address&quot;)</td>
<td>&quot;street address&quot;</td>
<td>Lowercases the given string</td>
</tr>
<tr>
<td>Lower ( , 15)</td>
<td>&quot;your name &quot;</td>
<td>Lowercases the current field and increases the length to 15</td>
</tr>
</tbody>
</table>

**See also**

Upper on page 420

String Functions on page 78
**MailWIP**

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

### Syntax

```
MailWIP (Address)
```

### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>Enter a valid email address. The system defaults to the email address window which lets the user select a valid recipient. This window appears if the email address is omitted or incorrect.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MailWIP( )</td>
<td>1 or 0</td>
<td>The default presents the user with the email system’s Address window, which lets the user choose the destination.</td>
</tr>
<tr>
<td>MailWIP( “TOM” )</td>
<td>1 or 0</td>
<td>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.</td>
</tr>
</tbody>
</table>

### 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:

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>#MAJOR = MajorVersion ()</td>
<td>string</td>
<td>Returns the system’s major version number.</td>
</tr>
</tbody>
</table>

**See also**

- Miscellaneous Functions on page 73
- MinorVersion on page 312
- DAL Script Examples on page 35
**MAX**

Use this function to return the greatest decimal value from a group of fields which have names that begin with common characters.

**Syntax**

\[ \text{MAX} (\text{PartialName}, \text{Section}, \text{Form}, \text{Group}) \]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PartialName</td>
<td>Enter a valid string. The string must be the common (prefix) portion of a set of field names that occur on the current section. The default is the current field.</td>
</tr>
<tr>
<td>Section</td>
<td>Enter the name of a section that contains the field named. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of a form that contains the section and/or field named. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of the form group that contains the form, section, and/or field named. The default is the current group.</td>
</tr>
</tbody>
</table>

The system calculates and returns the average of the values of all fields that begin with the specified partial name. An example of field names that have a common start are:

- Myfield1
- Myfield2
- Myfield20

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

The maximum is calculated by comparing all those fields that have values and have names matching the criteria. If all the field values are negative, then the result will be the negative number nearest the value zero. Note that zero (0) is a valid field value. Fields which have never been given a value are excluded from the calculation.

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

**Example**

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Section</th>
<th>Form</th>
<th>Group</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyField1</td>
<td>IMG A</td>
<td>FRM A</td>
<td>GRP</td>
<td>100.24</td>
</tr>
<tr>
<td>MyField2</td>
<td>IMG A</td>
<td>FRM A</td>
<td>GRP</td>
<td>200.16</td>
</tr>
</tbody>
</table>
Here are some examples:

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX ( )</td>
<td>100.24</td>
<td>Without any other information, the function will assume the current field and section. There will only be one value included in the search.</td>
</tr>
<tr>
<td>MAX (&quot;Myfield2&quot;)</td>
<td>200.16</td>
<td>Again, there is only one field included in this result.</td>
</tr>
<tr>
<td>MAX(&quot;MyField&quot;)</td>
<td>200.16</td>
<td>In this example, the current section contains two fields that begin with the name &quot;MyField&quot;. The second field has the greatest value.</td>
</tr>
<tr>
<td>MAX(&quot;MyField&quot;, &quot;IMG B&quot;)</td>
<td>98.60</td>
<td>Although two fields on IMG B have a matching name, only one field actually has a value.</td>
</tr>
<tr>
<td>MAX(&quot;MyField&quot;, &quot;IMG B&quot;, &quot;FRM A&quot;)</td>
<td>200.16</td>
<td>No section is specified in this example, so the entire form is searched. Four fields match the name criteria, but only three have values.</td>
</tr>
<tr>
<td>MAX(&quot;MyField&quot;, &quot;IMG B&quot;, &quot;GRP&quot;)</td>
<td>98.60</td>
<td>This example specifies a section and group, but no form. There are four fields that match the name criteria, but only two have values.</td>
</tr>
<tr>
<td>MAX(&quot;MyField&quot;, &quot;GRP&quot;)</td>
<td>200.16</td>
<td>This example names the group without a form or section. Eight fields meet the naming criteria, but only five fields actually have values.</td>
</tr>
</tbody>
</table>

See also Field Functions on page 61
Field Formats on page 62
Locating Fields on page 64
MIN

Use this function to return the least decimal value from a group of fields which have names that begin with common characters.

Syntax

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PartialName</td>
<td>Enter a valid string. The string must be the common (prefix) portion of a set of field names. The default is the current field.</td>
</tr>
<tr>
<td>Section</td>
<td>Enter the name of a section that contains the field named. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of a form that contains the section and/or field named. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of the form group that contains the form, section, and/or field named. The default is the current group.</td>
</tr>
</tbody>
</table>

The system calculates and returns the average of the values of all fields that begin with the specified partial name. An example of field names that have a common start are:

- Myfield1
- Myfield2
- Myfield20

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

The minimum is calculated by comparing all those fields that have values and match the naming criteria. If all the values are negative, then the result will be the negative number most distant the value of zero. Note that zero (0) is a valid field value. Fields which have never been given a value are excluded from the calculation.

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

Example

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Section</th>
<th>Form</th>
<th>Group</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyField1</td>
<td>IMG A</td>
<td>FRM A</td>
<td>GRP</td>
<td>100.24</td>
</tr>
<tr>
<td>MyField2</td>
<td>IMG A</td>
<td>FRM A</td>
<td>GRP</td>
<td>200.16</td>
</tr>
</tbody>
</table>
Here are some examples:

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Section</th>
<th>Form</th>
<th>Group</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyField1</td>
<td>IMG B</td>
<td>FRM A</td>
<td>GRP</td>
<td>98.60</td>
</tr>
<tr>
<td>MyField2</td>
<td>IMG B</td>
<td>FRM A</td>
<td>GRP</td>
<td>* no value yet *</td>
</tr>
<tr>
<td>MyField1</td>
<td>IMG A</td>
<td>FRM B</td>
<td>GRP</td>
<td>0.00</td>
</tr>
<tr>
<td>MyField2</td>
<td>IMG A</td>
<td>FRM B</td>
<td>GRP</td>
<td>* no value yet *</td>
</tr>
<tr>
<td>MyField1</td>
<td>IMG B</td>
<td>FRM B</td>
<td>GRP</td>
<td>70.77</td>
</tr>
<tr>
<td>MyField2</td>
<td>IMG B</td>
<td>FRM B</td>
<td>GRP</td>
<td>* no value yet *</td>
</tr>
</tbody>
</table>

Function Result Explanation

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN ( )</td>
<td>100.24</td>
<td>Without any other information, the function will assume the current field and section. There will only be one value included in the search.</td>
</tr>
<tr>
<td>MIN (&quot;MyField2&quot;)</td>
<td>200.16</td>
<td>Again, there is only one field included in this result.</td>
</tr>
<tr>
<td>MIN(&quot;MyField&quot;)</td>
<td>100.24</td>
<td>In this example, the current section contains two fields that begin with the name MyField. The first field has the least value.</td>
</tr>
<tr>
<td>MIN(&quot;MyField&quot;, &quot;IMG B&quot;)</td>
<td>98.60</td>
<td>Although two fields on IMG B have a matching name, only one field actually has a value.</td>
</tr>
<tr>
<td>MIN(&quot;MyField&quot;, &quot;FRM A&quot;)</td>
<td>98.60</td>
<td>No section is specified in this example, so the entire form is searched. Four fields match the name criteria, but only three have values.</td>
</tr>
<tr>
<td>MIN(&quot;MyField&quot;, &quot;IMG B&quot;, &quot;GRP&quot;)</td>
<td>70.77</td>
<td>This example specifies a section and group, but no form. There are four fields that match the name criteria, but only two have values.</td>
</tr>
<tr>
<td>MIN(&quot;MyField&quot;, &quot;GRP&quot;)</td>
<td>0.00</td>
<td>This example names the group without a form or section. Eight fields meet the naming criteria, but only five fields actually have values. The least of these five contains the value 0.00.</td>
</tr>
</tbody>
</table>

See also  Field Functions on page 61
Field Formats on page 62
Locating Fields on page 64
MINORVERSION

Use this function to get the minor version number of the system being executed.

Syntax

\[
\text{MinorVersion ( )}
\]

There are no parameters for this function.

Example

Here is an example:

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{vers} = \text{MajorVersion()} &amp; \text{.} &amp; \text{MinorVersion()} )</td>
<td>a string</td>
<td>Returns the system’s major and minor version number concatenated together with a period used as a separator.</td>
</tr>
</tbody>
</table>

See also

Miscellaneous Functions on page 73
MajorVersion on page 307
DAL Script Examples on page 35
**Minute**

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

**Syntax**

\[
\text{Minute} \ (\text{Time, Format})
\]

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>Enter a valid time string. The system assumes your entry is in the time format specified in the Format parameter. The default is the current time.</td>
</tr>
<tr>
<td>Format</td>
<td>Enter a valid time format string that describes the Time parameter. The default is time format 1 (HH:MM:SS).</td>
</tr>
</tbody>
</table>

**Example**

Here are some examples:

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minute(  )</td>
<td>05</td>
<td>Defaults to the current time and extracts 05</td>
</tr>
<tr>
<td>Minute (&quot;03:07:09&quot;)</td>
<td>07</td>
<td>Reads the given time and extracts 07</td>
</tr>
</tbody>
</table>

**See also**

Time Formats on page 80
MLEInput

Use this function to create a window with a title, prompt message, and a place for a user to enter multiple lines of text, such as the one shown here:

![MLEInput Window](image)

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)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prompt</td>
<td>Enter a text string to assign as the prompt for the field</td>
</tr>
<tr>
<td>Title</td>
<td>Enter a text string to assign as the title of the window.</td>
</tr>
<tr>
<td>Length</td>
<td>Enter the maximum input text length. The default is 1024.</td>
</tr>
<tr>
<td>DefText</td>
<td>Enter a text string to assign as the default input data.</td>
</tr>
</tbody>
</table>

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 `\`, 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”
```
<table>
<thead>
<tr>
<th>Function</th>
<th>Results</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| `input_data = MLEInput ("Enter comments; up to 1024 characters.", "Comments Input"); SetFld (input_data, variable);` | line 1
\nline 3
\nline 4
\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
  Now is the time
  line 3
  line 4
  gray area
  line 6 | (Assume this DAL script is executed after the example above.) The window would contain the data under item 1. If you enter:
  - 'Now is the time' in blank line 1
  - 'gray area: after line 4 ' and click Ok. The DAL variable, 'input_data', will contain the string under item 2. This example uses an A/N variable field. |
| `input_data = MLEInput ("Enter comments; up to 1024 characters.", "Comments Input");` | Null string | Assume you clicked Ok or Cancel without entering any data. The system stores a null string in the variable. |
| `input_data = MLETranslate (MLEInput ("Enter comments; up to 1024 characters.", "Comments Input"), ");` SetFld (input_data, "output"); | 1. `DAL internal variable` line 1
\nline 3
\nline 4
\nline 6 | After you enter the assumed information and click Ok, the DAL variable, 'input_data', contains the string shown in item 1. The data in the multi-line variable field, 'output', contains six lines, as shown in item 2. This example uses a multi-line variable field. |
| | 2. Multi-line variable field, `output`<br>  
  line 1
  blank line
  line 3
  line 4
  blank line
  line 6 |  |
See also  
MLETranslate on page 317  
Documaker Workstation Functions on page 59
MLETranslate

Use this function to translate the \n characters in a data string created by the MLEInput function. This function translates those characters into whatever characters you want.

Syntax

MLETranslate (String, ReplaceChar)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Enter the text string returned from the MLEInput function.</td>
</tr>
<tr>
<td>ReplaceChar</td>
<td>Enter a text string to replace each set of \n characters in a returned MLEInput data string.</td>
</tr>
</tbody>
</table>

The system returns the translated data string for display, storage, or both.

If the user presses ENTER to type on a new line, the system replaces the new line character with a \n when it returns the text. You can leave the result like this, so you know where the line breaks are supposed to be, or, you can send it to the MLETranslate function, which will translate the \n into whatever characters you want.

**NOTE:** Multi-line variable fields cannot accept the data captured by the MLEInput function without the data first being translated. Before you assign the output from a MLEInput function to a multi-line variable field, you should do the following.

```plaintext
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

<table>
<thead>
<tr>
<th>Function</th>
<th>Results</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>input_data = MLETranslate (MLEInput (&quot;Enter comments&quot;, &quot;Comments Input&quot;), &quot;=&quot;); SetFld (input_data, variable);</td>
<td>line 1&quot;line3&quot;line4 &quot;line 6</td>
<td>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.</td>
</tr>
<tr>
<td>Function</td>
<td>Results</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| `input_data = MLETranslate(MLEInput(“Enter comments; up to 1024 characters.”, “Comments Input”), “@(‘variable’), “#”);` | 1. Multi-line edit window line 1 **line 3 **line 4 **line 6  
2. Input_data  
line 1 Now is the time.  
#line 3 *line 4 gray area **line 6  
(Assume this DAL script is executed after the example above.)  
The window contains the data under item 1.  
Assuming you deleted the first two asterisks and entered *Now is the time*, followed by the entry key. Plus added *gray area* after line 4 and then clicked Ok. The DAL variable, 'input_data', would contain the data under item 2.  
This example uses an A/N variable field. |
| `input_data = MLETranslate(MLEInput(“Enter comments; up to 1024 characters.”, “Comments Input”), “*”);` | Null string  
Assume you clicked Ok or Cancel without entering any data. The system stores a null string in the variable. |
| `input_data = MLETranslate(MLEInput(“Enter comments; up to 1024 characters.”, “Comments Input”), “\n”);` | DAL internal variable line 1 \n \n line 3 \n \n line 4 \n \n line 6 \n  
Multi-line variable field line 1  
blank line  
line 3  
line 4  
blank line  
line 6  
After entering the assumed information and clicking Ok, the DAL variable, 'input_data', contains the data shown in item 1.  
The data in the multi-line variable, *output*, would contain six lines as shown in item 2.  
This example uses a multi-line variable field |
```plaintext
input_data = MLETranslate (MLEInput("Enter comments; up to 1024 characters.", "Comments Input", 
"output") , "\n");
SetFld (input_data, "output1");
```

<table>
<thead>
<tr>
<th>Function</th>
<th>Results</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Window&lt;br&gt;line 1&lt;br&gt;blank line&lt;br&gt;line 3&lt;br&gt;line 4&lt;br&gt;blank line&lt;br&gt;line 6</td>
<td>(Assume this DAL script is executed after the example above.)&lt;br&gt;After executing the script, the window contains the data shown in item 1.&lt;br&gt;Assuming you entered: <em>Now is the time.</em> for blank line 1 area, and added <em>gray area</em>, after the data 'line 4' and then clicked Ok.&lt;br&gt;The DAL internal variable, 'input_data', would contain the data string shown in item 2.&lt;br&gt;The multi-line variable field, 'output1', would contain the data shown in item 3.&lt;br&gt;This example uses a multi-line variable field.</td>
<td></td>
</tr>
<tr>
<td>2. Input_data&lt;br&gt;line 1 \nNow is the time. \nline 3 \nline 4 gray area \nline 6 \n</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Multi-line variable&lt;br&gt;output1&lt;br&gt;line 1&lt;br&gt;Now is the time.&lt;br&gt;line 3&lt;br&gt;line 4 gray area&lt;br&gt;blank line&lt;br&gt;line 6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See also  
MLEInput on page 314  
Documaker Workstation Functions on page 59
MOD

Use this function to return the remainder from modular arithmetic.

Syntax

\[ \text{MOD}(\text{Numerator}, \text{Denominator}) \]

Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numerator</td>
<td>Enter the value you want used as the numerator.</td>
</tr>
<tr>
<td>Denominator</td>
<td>Enter the value you want used as the denominator.</td>
</tr>
</tbody>
</table>

The system returns the integer remainder from an integer division.

**NOTE:** If you enter zero (0) as either the numerator or denominator, the system returns zero. Decimal or string input parameters are converted to integer values prior to the calculation.

Example

Assume you have the following entry in the SETRCPTBL.DAT file for the form trigger being processed. Also assume there are 30 records in the extract file that match the search mask.

```
;RP10;CIS;qa_f1550;;;Customer(1);;1,M;25;0;1;;DALTrigger;F1550;
```

Here is an example:

```
BeginSub F1550
    #rec = CountRec("1,F1550,31,Data")
    #remaining = MOD(#rec, TriggerRecsPerOvFlw( ))
    While(#remaining > 0)
        * write additional records
        Write_fm( )
        #mod -= 1
    Wend
    Return(#rec)
EndSub
```

In this example, the MOD function returns the integer remainder of 5. If no extract records matched the search mask, the system would have returned zero (0).

See also Mathematical Functions on page 72
MONTH

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

Syntax

\[
\text{Month (Date, Format, Locale)}
\]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>Enter a valid date string. The system assumes your entry is in the date format specified in the Format parameter. The default is the current date.</td>
</tr>
<tr>
<td>Format</td>
<td>Enter a valid date format string that describes the Date parameter. The default is date format 1.</td>
</tr>
<tr>
<td>Locale</td>
<td>(Optional) Enter the locale code. If you omit this parameter, the system checks the Locale INI option. If the Locale INI option offers no value, the system defaults to USD (United States/English).</td>
</tr>
</tbody>
</table>

The system determines the month portion of the given date based on the format you specify. This function is often used with the MonthName function.

Example

Here are some examples:

(Assume the current date is 07/01/09.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Month ()</td>
<td>7</td>
<td>The parameter defaults to the current date.</td>
</tr>
<tr>
<td>Month (&quot;09/138&quot;, &quot;I&quot;)</td>
<td>5</td>
<td>The given date (09/138) in the date format I is the equivalent of May 18, 2009. Therefore the number of the month (5) is returned.</td>
</tr>
<tr>
<td>datestring=DateAdd(,,3); Month(datestring)</td>
<td>10</td>
<td>First the DateAdd function defaults to the current date and adds three months. The resulting date of October 1, 2009 is returned to the target variable datestring. The Month function then returns the number of the month of October (10).</td>
</tr>
</tbody>
</table>

See also

Date Functions on page 51
Locales on page 55
Date Formats on page 52
Locales on page 55
DateAdd on page 184
MonthName on page 322
**MONTHNAME**

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

### Syntax

```
MONTHNAME (Month, Locale)
```

### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Month</td>
<td>Enter a valid month value. For example, enter one (1) for January or 12 for December. The default is the current month.</td>
</tr>
<tr>
<td>Locale</td>
<td>(Optional) Enter the locale code. If you omit this parameter, the system checks the Locale INI option. If the Locale INI option offers no value, the system defaults to USD (United States/English).</td>
</tr>
</tbody>
</table>

This function is most often used with the Month function. The Month function extracts the month number from a given date.

### Example

Here are some examples:

(Assume the current date is 07/01/09.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>RETURN(MONTHNAME())</code></td>
<td>July</td>
<td>Defaults to the current month.</td>
</tr>
<tr>
<td><code>RETURN(MONTHNAME(11))</code></td>
<td>November</td>
<td>Returns November, which corresponds to the given parameter (11).</td>
</tr>
<tr>
<td><code>RETURN(MONTHNAME(Month(&quot;09/138&quot;, &quot;I&quot;)))</code></td>
<td>May</td>
<td>First the Month function determines that the month number for the given date is 5, (09/138 is equivalent to May 18, 2009) Then MONTHNAME returns the corresponding month name of May.</td>
</tr>
</tbody>
</table>

### See also

- Date Functions on page 51
- Locales on page 55
- Month on page 321
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)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(MsgLine1</td>
<td>Enter the first line of the message.</td>
</tr>
<tr>
<td>MsgLine2</td>
<td>Enter the second line of the message.</td>
</tr>
<tr>
<td>MsgLine3</td>
<td>Enter the third line of the message.</td>
</tr>
<tr>
<td>Title</td>
<td>Enter a title for the message window.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG (&quot;Sample Line 1&quot;, &quot;Sample Line 2&quot;, &quot;Sample Line 3&quot;, &quot;Sample Message&quot;)</td>
<td>&quot;Sample Message&quot; is the title of the message. &quot;Sample Line 1&quot; &quot;Sample Line 2&quot; &quot;Sample Line 3&quot; is the message to the user.</td>
<td></td>
</tr>
<tr>
<td>MSG (&quot;Don't forget to inform the customer about the luxury tax.&quot;)</td>
<td>The message appears without a title.</td>
<td></td>
</tr>
</tbody>
</table>

See also [Documaker Workstation Functions on page 59](#)
NL

Use this function to retrieve a string that contains a new line character sequence. This is useful when you are creating output text messages that contain line breaks.

**NOTE:** On Windows, this function returns a carriage return/line feed pair. On UNIX, it returns a line feed. The function works in both Documaker Server and Workstation.

**Syntax**

```
NL()
```

There are no parameters for this function.

**Example**

This example shows how you can use this function with the `Print_It` function:

```
Print_It("This is line one." & NL() & "This is line two.")
```

In this example, two lines are output to the command line during Documaker Server processing. Without this function, you would have to include two `Print_It` statements.

```
This is line one.
This is line two.
```

This example shows how you can create multi-line text area messages:

```
data = ?("cus_name") & NL() & ?("state") & ", " & ?("zip")
SetFld(data, "cus_ss")
```

In this example, two lines are stored in a multi-line text area on separate lines. Without this function, you would have to define the multi-line text area, a fixed-size font, and the script would have calculated the number of spaces to pad to the first line to make sure the line wrapped properly.

```
John A. Smith
CA,  81234-4444
```

You can also use the `NL` function when you are creating comment strings you want inserted into a print stream using the `AddComment` procedure.

**See also**  
String Functions on page 78
**NUM**

Use this function to return the numeric value of a field. On numeric formatted fields, this function operates the same as the @ function, however, NUM automatically converts a non-numeric field into its numeric content.

**Syntax**

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

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>Enter the name of a section field. The default is the current field.</td>
</tr>
<tr>
<td>Section</td>
<td>Enter the name of a section that contains the field you specified. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of a form that contains the section and/or field you specified. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of the form group that contains the form, section, and/or field you specified. The default is the current group.</td>
</tr>
</tbody>
</table>

The system uses the parameters provided to search for one field on a section and return that field’s data as a number. The field does not have to be defined as a numeric data type.

**Example**

Here are some examples:

(Assume the current field value is ABC1234.23XYZ and is named MyField. Also, assume that a second occurrence of MyField appears on the form, MyForm, and contains the value *automobile.*)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM( )</td>
<td>1234.23</td>
<td>Returns the value in the current field as a number. Notice that any non-numeric value is removed before returning the value.</td>
</tr>
<tr>
<td>NUM (&quot;MyField&quot;)</td>
<td>1234.23</td>
<td>Returns the value in the named field, located on the current section.</td>
</tr>
<tr>
<td>NUM(&quot;MyField\2&quot;, &quot;MyForm&quot;)</td>
<td>0</td>
<td>Since the second occurrence of MyField on this form does not contain any numeric values, the result is zero (0).</td>
</tr>
</tbody>
</table>

**See also**

Field Functions on page 61  
Field Formats on page 62  
Locating Fields on page 64  
@ on page 109
**NUMERIC**

Use this function to test if a string contains a valid numeric value. The system returns one (1) if the string is a valid number and zero (0) if not.

**Syntax**

```plaintext
Numeric (String)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Enter a valid string. The default is the value of the current field.</td>
</tr>
</tbody>
</table>

The system returns true or false depending on whether the string parameter contains a valid numeric value.

Leading or trailing spaces are removed before the string is evaluated. A numeric value contains only numbers, a sign (leading or trailing), and a single decimal point.

**Example**

Here are some examples:

(Assume the current field value is -101.564)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric ()</td>
<td>1</td>
<td>Defaults to the current field and determines a true statement, such as if the field contains a valid numeric value.</td>
</tr>
<tr>
<td>Numeric (“123T456”)</td>
<td>0</td>
<td>Determines a false statement, such as if the field does not contain a valid numeric value.</td>
</tr>
<tr>
<td>IF Numeric (“4633392”) result = “Yes”; ELSE result = “No”; END Return(Result)</td>
<td>“YES”</td>
<td>The specified value is numeric therefore the variable result will be assigned Yes.</td>
</tr>
</tbody>
</table>

**See also** Mathematical Functions on page 72
**NumText**

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

**Syntax**

```plaintext
NumText (Number, DollarWord, CentWord, DeciMode)
```

**Parameter** | **Description**
--- | ---
Number | Enter an amount. The default is the value of the current field.
DollarWord | Enter the word you want the system to use to describe the main unit of currency. The default is: “dollars and”
CentWord | Enter the word you want the system to use to describe the secondary unit of currency. The default is: “cents”
DeciMode | Choose from these options:
1 - numeric decimal amount
2 - spell decimal amount
3 - suppress zero, numeric decimal amount
4 - suppress zero, spell decimal amount
The default is one (1).

The system returns the written word equivalent of a numeric value.

The system attempts to remove formatting information from the parameter number. If the value after deformatting is not a valid number, the function returns an empty result.

This function is basically designed to produce the text that might appear on a bank check. The default type strings are dollars and and cents. When the default descriptions are used, this function uses the singular word dollar or cent when the associated value is 1, otherwise it uses the plural text. Alternate descriptions provided as parameters are not changed for any value amount.

The optional decimode parameter is an integer value from 1 to 4. This parameter includes or suppresses the zero (0) decimal value. You can also use this parameter to specify if the decimal amount should be presented as a number or spelled out.

**NOTE:** This function only supports two decimal places. Additional places are truncated without rounding.

**Example**

Here are some examples (assume the current field value is 1641.56):

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumText ()</td>
<td>One thousand six hundred forty-one dollars and 56 cents</td>
<td>Defaults to dollars and cents and numeric decimal result.</td>
</tr>
<tr>
<td>NumText( , , 2)</td>
<td>One thousand six hundred forty-one dollars and fifty-six cents</td>
<td>Decimal mode 2 spells the decimal amount.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumText(12.00, ,3)</td>
<td>Twelve dollars</td>
<td>A decimal mode of 3 suppresses the zero decimal.</td>
</tr>
<tr>
<td>NumText(34.55,&quot;meters and&quot;,&quot;centimeters&quot;,3)</td>
<td>Thirty four meters and 55 centimeters</td>
<td>Demonstrates substituting alternate references.</td>
</tr>
<tr>
<td>NumText(1.00, )</td>
<td>One dollar</td>
<td></td>
</tr>
<tr>
<td>NumText(1.01, )</td>
<td>One dollar and one cent</td>
<td></td>
</tr>
<tr>
<td>NumText(1.00,&quot;meters and&quot;,&quot;centimeters&quot;,3)</td>
<td>One meters and</td>
<td></td>
</tr>
<tr>
<td>NumText(1.01,&quot;meters and&quot;,&quot;centimeters&quot;,3)</td>
<td>One meters and one centimeters</td>
<td></td>
</tr>
</tbody>
</table>

If you include Dollarword and Centword and the number does not contain a decimal, the exact content you specify in Dollarword is printed and the system does not distinguish the number from being singular or plural. The Dollarword and Centword are printed exactly as specified. Notice the difference in the default format (dollars and cents) in the last two examples.

See also  
String Functions on page 78  
FrenchNumText on page 248
PAD

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

Syntax

```
PAD (String, Length, Char)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Enter a valid string. The default is the value of the current field.</td>
</tr>
<tr>
<td>Length</td>
<td>Specify the desired length of output. The default is the length of value in the String parameter.</td>
</tr>
<tr>
<td>Char</td>
<td>Enter a valid string that contains the pad characters you want to use. The default is the space character.</td>
</tr>
</tbody>
</table>

The system returns the string created by padding parameter 1 with the characters from parameter 3.

If the length specified in parameter 2 is longer than the string, the result is increased to the integer length you specified. If the specified integer length is less than the string, the length of the string is maintained.

The string is not truncated. All leading and trailing spaces are removed from the input string before the PAD function.

Example

Here are some examples:

(Assume the current field contains the text *Last Name Only.*)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAD(&quot;may &quot;, 9)</td>
<td>&quot;may &quot;</td>
<td>Pad the result string to a length of 9. The pad character defaults to the space character.</td>
</tr>
<tr>
<td>PAD ( , 20, !)</td>
<td>&quot;Last Name Only!!!!!!&quot;</td>
<td>Defaults to the current field, and adds the pad character (!) until the length reaches 20.</td>
</tr>
<tr>
<td>PAD ( )</td>
<td>&quot;Last Name Only&quot;</td>
<td>Defaults to the current field; No length was specified; therefore the field remains the same.</td>
</tr>
<tr>
<td>PAD (&quot;Ten dollars &quot;, 15, &quot;)</td>
<td>&quot;Ten dollars****&quot;</td>
<td>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.</td>
</tr>
</tbody>
</table>

See also String Functions on page 78
**PageImage**

Use this function to return the name of a section on a given page number within the form set or form. If you include the name of a recipient as a parameter, the system will filter the sections by that name. Once you have a section name, you can use other DAL functions to query the section, to insert a new section, or to delete the section.

**Syntax**

`PageImage (Page, Recipient, Form, Group)`

**Parameter** | **Description**
--- | ---
Page | Include this parameter to indicate the specific page where you want to locate a section. If you omit this parameter, a section from page one is located. Depending upon the remaining parameters, this page will be the page within the entire form set, or within a given form.
Recipient | Include this parameter to filter the sections located by that recipient. If you omit this parameter, the name of the first section on the requested page is returned.
Form | Include this parameter if you want the system to first locate the specified form and then use the Page parameter to find the specified page within that form. If you omit this parameter, the Page parameter is based on a page located by starting at the first page of the form set or group (if the Group parameter is specified).
Group | Include this parameter to tell the system to first locate a specific group. If you also include the Form parameter, the system will find that form in that group. If you omit the group but include the form, the system looks for that form in the current group — which is identified by the current field or section executing the script. If you include the group but omit the form, the system uses the Page parameter to return that page in the specified group.

The name returned by this function also includes the occurrence value if the section occurs more than once. For instance, if a section named, `MySection`, is located on the given page, but this is the second occurrence of the section within the named form, the name returned will be `MySection\2`.

**See also**

- PageInfo on page 331
- Name Functions on page 74
**PAGEINFO**

Use this function to get information about the page of a form you specify. This information includes height, width, and orientation.

**Syntax**

```
PageInfo (Prefix, Page, Recipient, Section, Form, Group)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefix</td>
<td>This parameter identifies a prefix for creating the variable names to contain the page information.</td>
</tr>
<tr>
<td>Page</td>
<td>(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.</td>
</tr>
<tr>
<td>Recipient</td>
<td>(Optional) This parameter names a specific recipient that must be used on a section of the page located. If you omit this parameter, the function matches the first page identified by the remaining search criteria.</td>
</tr>
<tr>
<td>Section</td>
<td>(Optional) This parameter names a section that should be found to identify the page.</td>
</tr>
<tr>
<td>Form</td>
<td>(Optional) This parameter names a form that contains the page to be found.</td>
</tr>
<tr>
<td>Group</td>
<td>(Optional) This parameter names a group that must contain the page to be found.</td>
</tr>
</tbody>
</table>

The Section, Form, and Group parameters are optional and when used will work together to locate the starting page for the search. Here are some examples:

- If you name a section, but no form or group, the assumption is the section is on the current form.
- If you name a form, without a group, the assumption is the form must be within the current group of forms.
- If you name a section and a group, but no form, the assumption is the section can occur on any form within that group.
- If you omit the section, form, and group parameters the search starts from the beginning of the document set.

Once the requested page is located, the system assigns the page information to DAL variables using the Prefix parameter. If these variables do not exist in DAL, the system creates them for you. The system creates four internal variables: `prefix.height`, `prefix.width`, `prefix:landscape`, and `prefix:paper`. If these variables exist, the system modifies them with the new information.

For example, a call like this will create four variables.

```
PageInfo (*MYPAGE*);
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MYPAGE.Height</td>
<td>Contains the height of the page in FAP units (2400 DPI).</td>
</tr>
<tr>
<td>MYPAGE.Width</td>
<td>Contains the width of the page in FAP units (2400 DPI).</td>
</tr>
</tbody>
</table>
Note that for landscape pages, the height and width values reflect the rotation of the width and height. For instance, non-landscape letter documents return a height of 26400 and a width of 20400. Landscape letter documents return a height of 20400 and a width of 26400.

The page size (height and width) is determined by finding the first section on a page with the required recipient. If no recipient is specified, the first section on the page is used. The form pages within a document do not have to be the same size. Also note that if the first section on a page is a custom size, the width and height will reflect the best values.

Generally when a section is a custom size, the actual page size is found in the form definition. If, however, the form size (height or width) is smaller than the corresponding section size, then the larger of the values is returned.

Also remember since page size is determined by the first section designated for a given recipient, it is possible for the same page to have a different size for different recipients.

The PageInfo function returns a value if used in an expression that requires it. The possible return values are zero (0) if the requested page could not be found, or non-zero if the page is found.

Possible reasons for a page not to be located include:

- The page number is outside the range of pages for the given search criteria. For instance, you ask for page three of a form that only has two pages.
- The recipient cannot be located within the document search criteria.
- The section, form, or group (or combination thereof) cannot be located within the specified document.

See also PageInfo on page 331
Page Functions on page 75
**PaginateForm**

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

**NOTE:** The AddImage and DelImage DAL functions include a parameter (Paginate) which you can use to force re-pagination after the affected section has been manipulated.

**Syntax**

```
PaginateForm (Form, Group)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form</td>
<td>(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.</td>
</tr>
<tr>
<td>Group</td>
<td>(Optional) This parameter identifies the Key2 or GroupName2-level parent that contains the form. This is sometimes referred to as the line of business that contains the form. If you omit the Group parameter, the system tries to locate the named form within the current group that is controlling the execution of the script.</td>
</tr>
</tbody>
</table>

You can call PaginateForm as a function or procedure. As a function, it returns a one (1) if the requested form is located or a zero (0) if it could not be located.

Note that if the form is found and paginated, there may not be any visible change to the document. The form layout is determined when you design the form and by the application of section origin rules.

See also

- AddImage on page 122
- DelImage on page 212
- Page Functions on page 75
**PARSELISTCOUNT**

Use this function to count the indexed components within the formatted text.

**NOTE:** Use the ParseListCount and ParseListItem functions when accepting tokenized (comma or semicolon-delimited) data, such as data from a spreadsheet program or other application. These are sometimes referred to as CSV (comma separated value) files.

### Syntax

`ParseListCount (String, Separator)`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Enter the formatted string you want the system to search and parse.</td>
</tr>
<tr>
<td>Separator</td>
<td>Enter the list of character separators used within the formatted text parameter. If you omit this parameter, the system uses semicolons and commas.</td>
</tr>
</tbody>
</table>

The system returns the number of formatted items found within the String parameter. If the String parameter text starts with delimiter characters, those characters are skipped. If you do not have at least a space character between delimiters, this will not be identified as a separate index item.

**NOTE:** You can use the ParseListItem function to return the text components parsed from the formatted text.

### Example

For these examples, assume `xString = “A,B;C”`

```
value = ParseListCount(xString)
```

The value is 3.

```
value = ParseListCount(xString,”;”)   
```

The value is 2. In this example the parameter overrides and assigns only a semicolon as a valid separator. Therefore, there are two items within this string.

For these examples, assume `xString = “;A;B;C”`

```
value = ParseListCount(xString)
```

The value is 3. If the formatted string starts with separator characters, these characters are skipped. Note that adjacent separators are treated as a single separation.

For these examples, assume `xString = “;A; ,B;”`

```
value = ParseListCount(xString)
```

The value is 4. Note the intervening character – a space - between some of the separator characters.
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

(ParseListCount on page 335)

(ParseListItem on page 336)

(String Functions on page 78)
**ParseListItem**

Use this function to return indexed components from the formatted text.

**NOTE:** Use the ParseListCount and ParseListItem functions when accepting tokenized (comma or semicolon-delimited) data, such as data from a spreadsheet program or other application. These are sometimes referred to as CSV (comma separated value) files.

**Syntax**

```
ParseListItem (String, Item, Separator)
```

**Parameter** | **Description**
--- | ---
String | Enter a formatted string to search and parse.
Item | Enter the number of the item you want from within that formatted string. If you omit this parameter, the first item parsed from the formatted text is returned.
Separator | Enter a list of character separators used within the formatted text parameter. If you omit this parameter, the semicolons and commas are used.

The return value is a string of text. If the formatted text contains leading or trailing spaces on items formatted within it, they are not removed. You can use the Trim function on the returned text if you do not want the spaces.

If the first parameter text starts with delimiter characters, they will be skipped. Because the function will return spaces, you know when you have exceeded the number of items formatted within the string when you get an empty string returned.

**NOTE:** If you do not have at least a space character between delimiters, this will not be identified as a separate index item.

**Example**

Here are some examples. Assume `xString = “A,B;C”`

```
value = ParseListItem(xString)
The value is A.
```

```
value = ParseListItem(xString,3)
The value is C because the default separators include both commas and semicolons.
```

```
value = ParseListItem(xString,1,";")
The value is A. 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 \textit{A}. Note that if the formatted string starts with separator characters they are skipped.

\begin{verbatim}
value = ParseListItem(xString,2)
\end{verbatim}

The value is \textit{B}. Note again how adjacent separators without intervening characters (or space) are skipped. Therefore the semicolon and comma (,;) between the \textit{A} and \textit{B} are treated as a single separation.

\begin{verbatim}
value = ParseListItem(xString,3)
\end{verbatim}

The value is \textit{C}. Note again how adjacent separators without intervening characters (or space) are skipped. Therefore the semicolon and comma (,;) between the \textit{A} and \textit{B} are treated as a single separation and the semicolon and comma (,;) between the \textit{B} and \textit{C} are also treated as a single separation.

\begin{verbatim}
value = ParseListItem(xString,3,"",")
\end{verbatim}

The value is \textit{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 \texttt{xString = "; ,A; ,B;"}

\begin{verbatim}
value = ParseListItem(xString)
\end{verbatim}

The value is a space. Note that there is at least one intervening character — a space — between the first set of separator characters.

\begin{verbatim}
value = ParseListItem(xString,2)
\end{verbatim}

The value is \textit{A}.

\begin{verbatim}
value = ParseListItem(xString,3)
\end{verbatim}

The value is a space.

\begin{verbatim}
value = ParseListItem(xString,4)
\end{verbatim}

The value is \textit{B}.

\begin{verbatim}
value = ParseListItem(xString,5)
\end{verbatim}

The value is an empty string because this index item exceeds the list of items provided.

See also \texttt{ParseListCount on page 334}

\texttt{String Functions on page 78}
**PathCreate**

Use this function to create the parameter subdirectory path if it does not already exist. The function assumes all of the text you pass in is a path and does not remove any of it before it tries to verify or create the path.

The function creates multiple subdirectories as necessary in an attempt to satisfy the request.

**NOTE:** The PathCreate and PathExist functions let you create paths and verify that paths exist. These are useful, for instance, if you are trying to create printed output and organize that output into subdirectories on disk. You can do this using one of the print callback methods that support a DAL script.

### Syntax

PathCreate (Path)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>Enter the full path you want the system to verify or create.</td>
</tr>
</tbody>
</table>

The system returns zero (0) if it cannot create the path requested. Anything else means the path now exists, but is not an indication that it had to be created.

**NOTE:** This function is not valid on the z/OS operating system.

See also  
PathExist on page 339  
File and Path Functions on page 68
**PathExist**

Use this function to take the parameter path you provide and check for its existence. This function does not create subdirectories.

**NOTE:** The PathCreate and PathExist functions let you create paths and verify that paths exist. These are useful, for instance, if you are trying to create printed output and organize that output into subdirectories on disk. You can do this using one of the print callback methods that support a DAL script.

**Syntax**

PathExist (Path)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>Enter the full path you want the system to verify.</td>
</tr>
</tbody>
</table>

The system returns zero (0) if the path is invalid. Anything else indicates the path you provided exists.

**NOTE:** This function merely checks for the existence of the path you specified. Provided the path does exist, this is not an indication that the process will be able to access or create files within that path.

**See also**

- PathCreate on page 338
- File and Path Functions on page 68
POW

Use this function to raise a number to an exponential power.

Syntax

```
POW (Base, Exponent)
```

Parameter | Description
---|---
Base | Enter the base number, positive or negative, to be raised to an exponential power. The default is 1.00.
Exponent | Enter the exponent (power) to which the base number will be raised. The default is zero (0).

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

This function handles calculations such as those needed to figure annuities and interest rates. Using the function, a decimal number is returned from a base number that has been raised to an exponential power. Values can contain up to 14 digits.

The function handles both positive and negative integer or decimal values for the base number and exponent.

Example

Here is an example:

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>POW (2, 3)</td>
<td>8</td>
</tr>
<tr>
<td>POW (2, -3)</td>
<td>0.125</td>
</tr>
<tr>
<td>POW (34.5, 3.14)</td>
<td>67414.289005316</td>
</tr>
</tbody>
</table>

See also Mathematical Functions on page 72
**PRINT**

Use this procedure/function to print the entire document. Optionally this procedure returns one (1) on success or zero (0) on failure.

**Syntax**

```plaintext
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:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Print ( )</td>
<td>1 or 0 (zero)</td>
<td>Print the current form set.</td>
</tr>
</tbody>
</table>

**See also** [WIP Functions on page 88](#)
**Print_It**

Use this procedure to print a string to the console.

**Syntax**

\[
\text{Print}_\text{It} \ (\text{Text})
\]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>Enter the string you want the system to print to the console.</td>
</tr>
</tbody>
</table>

**NOTE:** This is useful when debugging scripts in Documaker Server.

**Example**

Here is an example:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| If (HaveGVM('Company'))
  Print_It(GVM('Company'))
End | a string | The content of the GVM variable \textit{Company} is printed to the console. |

**See also**

- DAL Script Examples on page 35
- Miscellaneous Functions on page 73
**PrinterClass**

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

**Syntax**

\[ \text{PrinterClass ( )} \]

There are no parameters for this function.

**Example**

Here are some examples. Assume these INI options exist:

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

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

See also

- AddComment on page 117
- DAL Script Examples on page 35
- Printer and Recipient Functions on page 76
**PRINTERGROUP**

Use this function to retrieve the group name that is being used to generate the print stream. This name is stored in the INI file.

**Syntax**

```
PrinterGroup ( )
```

There are no parameters for this function.

**Example**

Here are some examples. Assume these INI options exist:

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>G_name = PrinterGroup()</td>
<td>PrtType:AFP</td>
<td>Retrieves the printer group name.</td>
</tr>
<tr>
<td>ScriptName = GetINIString (, PrinterGroup(), 'OnDemandScript')</td>
<td>OnDemand</td>
<td>Contains the name of the DAL script you want to execute.</td>
</tr>
</tbody>
</table>

**See also**

- GetINIString on page 259
- DAL Script Examples on page 35
- Printer and Recipient Functions on page 76
**PRINTERID**

Use this function to return the active printer ID assigned during a Documaker Server processing run. The printer ID is a string of text associated with the current batch output and normally determined via INI option during a batch run. The IDs are associated from the PrinterInfo control group with each batch printer definition.

You can use this ID, for instance, when naming print file. For example, you might want all the files from one printer ID in a separate location or have the names prefixed in a certain manner.

**Syntax**

```
PrinterID ( )
```

There are no parameters for this function.

**NOTE:** The printer ID is only valid during a batch print operation and calling the function at other times returns the last value assigned or an empty string.

**See also**

Printer and Recipient Functions on page 76
**PrinterOutputSize**

Use this function to get the approximate size of the current print output file during a batch print operation.

**Syntax**

```plaintext
PrinterOutputSize()
```

There are no parameters for this function.

This function is only available during Documaker batch process operations, such as GenPrint, and only returns a non-zero value if a print stream is actively being built and written to a physical file on disk.

**NOTE:** When printing through the Windows GDI device, there is no physical file and therefore the value returned is unreliable and may be zero.

**See also**

Printer and Recipient Functions on page 76
**PutFormAttrib**

Use this function to save the named attribute and information to a form within your document set. You can add new attributes via this function or update an attribute on a form you specify.

**NOTE:** Adding or changing a form attribute only affects the current document set. You cannot update the contents of a FORM.DAT or FOR file from a DAL script. Once changed, the attribute will stay with your form even if saved to WIP or archived.

### Syntax

```
PutFormAttrib (Name, Data, Form, Group)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter the name of the form attribute (metadata).</td>
</tr>
<tr>
<td>Data</td>
<td>Enter the value associated with the form attribute (metadata). The default is an empty string.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter name of a form to retrieve data from. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter name of the group that contains the specified form. The default is the current group.</td>
</tr>
</tbody>
</table>

If you omit both the Form and Group parameters, the system chooses the current form, based on where the script executes. During Entry (via the Workstation or the plug-in) this will be the form that contains the DAL script. During Documaker Server processing, the first logical form found within the document set is the current form, unless the script is executed from a section or field rule.

If you include the Form parameter, but omit the Group parameter, the system looks for the form within the current group of forms, as defined by where the script executes. During Entry (via the Workstation or the plug-in) this is the group that contains the form where the script executes. During Documaker Server processing, the first logical group found within the document set is the current group, unless the script is executed from a section or field rule.

If you omit the Form parameter but include the Group parameter, the system locates the first form within the group you specified.

If the function is successful in adding the attribute to a form, it returns a one (1). If the function is not successful, it returns a zero (0). A failure typically means that based on the form and group name parameters, the function could not locate the form.
Example

In this example assume the form 1111 has this metadata:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offer</td>
<td>Good until cancelled</td>
</tr>
<tr>
<td>Codes</td>
<td>R4,79, ZW</td>
</tr>
<tr>
<td>Restriction</td>
<td>Must be 18 or older</td>
</tr>
</tbody>
</table>

Here is an example:

```
xx=PutFormAttrib("Restriction", "Must be 18 or older", "1111")
```

After execution, the form contains the following:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offer</td>
<td>Good until cancelled</td>
</tr>
<tr>
<td>Codes</td>
<td>R4,79, ZW</td>
</tr>
<tr>
<td>Restriction</td>
<td>Must be 18 or older</td>
</tr>
</tbody>
</table>

Keep in mind...

- The name of a user-defined attribute must follow the naming convention used for Documaker objects. This means the name cannot include semicolons (;), backslashes (\), equals signs ( = ), or two pipe symbols in sequence ( || ). You can use underscores (_), hyphens and dashes (-), and periods or full stops ( . ).

- You cannot use a pipe symbol ( | ) as the first character in a name or value.

- The value size cannot exceed 1000 characters for each value.

- The names **Category** and **Key3** are reserved. Avoid using these names.

See also

- GetFormAttrib on page 255
- Have Functions on page 69
**PutINIBool**

Use this procedure/function to store a Boolean value in an INI control group and option Boolean variable.

**Syntax**

```
PutINIBool (Context, Group, Option, Default)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context</td>
<td>(Optional) Enter the name (valid string) associated with a set of INI control groups and options loaded into cache memory.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of the control group which contains the INI option Boolean variable.</td>
</tr>
<tr>
<td>Option</td>
<td>Enter the name of the option in which the INI Boolean variable will be stored. If the control group and option does not exist, the system creates them.</td>
</tr>
<tr>
<td>Default</td>
<td>(Optional) Enter the default Boolean value to store into the control group and option Boolean variable. The default is zero (0).</td>
</tr>
</tbody>
</table>

The system returns one (1) if no error occurred during execution and zero (0) if there was an error.

This procedure stores a Boolean value in the specified control group and option Boolean variable.

If you omit context name and the control group and option does not exist in any of the INI files, the procedure stores the Boolean value in the FSIUSER.INI file.

If there are multiple control groups and options with the same name, the procedure stores the Boolean value in the first INI control group and option variable equal to the specified control group and option name.

If a context name is present, the procedure only stores the Boolean value in the control group and option variable associated with the context name.

**Example**

Assume an INI file, `TEST1.INI`, was loaded with the context name, `MVF`. The TEST1.INI file contains this control group and option:

```
< Control >
  LogEnabled = 1
```

In addition, the FSIUSER.INI file contains this control group and option:

```
< Control >
  LogEnabled = 0
```

Plus, the FSISYS.INI file contains this control group and option:

```
< Control >
  LogEnabled = 1
```

Based on this scenario, the following table shows and explains several possible results.
<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rc = PutINIBool (&quot;control&quot;, &quot;LogEnabled&quot;);</code></td>
<td>The variable <code>bool_value</code> in the FSIUSER.INI file now contains a zero (0). The return code <code>rc</code> is set to one (1).</td>
<td>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.</td>
</tr>
<tr>
<td><code>rc = PutINIBool (&quot;MVF&quot;, &quot;control&quot;, &quot;LogEnabled&quot;);</code></td>
<td>The variable <code>bool_value</code> in the TEST1.INI file now contains a zero (0). The return code <code>rc</code> is set to one (1).</td>
<td>The procedure scans only the control group and option set associated with the context name <code>MVF</code>.</td>
</tr>
<tr>
<td><code>rc = PutINIBool (&quot;MVF&quot;, &quot;control&quot;, &quot;LogEnabled&quot;, 1);</code></td>
<td>The variable <code>bool_value</code> in the TEST1.INI file now contains a one (1). If <code>Control</code> and <code>LogEnabled</code> are not found, the system creates a control group and option and sets the Boolean variable <code>LogEnabled</code> to one (1).</td>
<td>The procedure scans only the control group and option set associated with the context name <code>MVF</code>.</td>
</tr>
</tbody>
</table>

See also  
INI Functions on page 70  
Using INI Options on page 8
**PutINIString**

Use this procedure/function to store a string value in a specified INI control group and option string variable.

**Syntax**

```
PutINIString (Context, Group, Option, Default)
```

**Parameter**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context</td>
</tr>
<tr>
<td>Group</td>
</tr>
<tr>
<td>Option</td>
</tr>
<tr>
<td>Default</td>
</tr>
</tbody>
</table>

The system returns one (1) if no error occurred during execution and zero (0) if there was an error.

This procedure stores a string value into the specified control group and option string variable. If the context name is not present and the control group and option does not exist in any of the INI sets, the procedure stores the string variable into the FSIUSER.INI file.

If there are multiple control groups and options of the same name, the procedure stores the string value in the first INI control group and option variable equal to the specified control group and option name.

If a context name is present, the procedure only stores the string value in the control group and option variable associated with the context name.

**Example**

Let’s assume that an INI file, `TEST1.INI`, was loaded with the context name, `MVF`. The TEST1.INI file contains this control group and option:

```
< Control >
  title = MVF’s string
```

In addition, the FSIUSER.INI file contains this control group and option:

```
< Control >
  Title = Bob’s string
```

Plus, the FSISYS.INI file contains this control group and option:

```
< Control >
  Title = fap entry
```

Based on this scenario, the following table shows and explains several possible results.
<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rc = PutINIString (&quot;Control&quot;, &quot;Title&quot;);</code></td>
<td>The string variable <em>Title</em> in the FSIUSER.INI file now contains <em>Bob's string</em>. The return code <code>rc</code> is set to one (1).</td>
<td>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.</td>
</tr>
<tr>
<td><code>rc = PutINIString (&quot;MVF&quot;, &quot;Control&quot;, &quot;Title&quot;);</code></td>
<td>The string variable <em>Title</em> in the TEST1.INI file now contains <em>MVF's string</em>. The return code <code>rc</code> will be set to one (1).</td>
<td>The procedure scans only the control group and option set associated with the context name <em>MVF</em>.</td>
</tr>
<tr>
<td><code>rc = PutINIString (&quot;MVF&quot;, &quot;Control&quot;, &quot;Title&quot;, &quot;New string&quot;);</code></td>
<td>The string variable <em>Title</em> in the TEST1.INI file now contains <em>New string</em>. If <em>Control</em> and <em>Title</em> are not found, the system creates them and sets the string variable <em>Title</em> to <em>New string</em>.</td>
<td>The procedure scans only the control group and option set associated with the context name <em>MVF</em>.</td>
</tr>
</tbody>
</table>

**Example**

INI Functions on page 70

Using INI Options on page 8
**RecipBatch**

Use this function to get the name of the recipient batch file being processed. This function is only applicable to batch banner processing or comment record processing with the GenPrint program.

**Syntax**

```java
RecipBatch()
```

There are no parameters for this function.

**Example**

Here is an example. Assume the recipient batch file entitled *Batch1* is being processed.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>rb = RecipBatch();</td>
<td>Batch1</td>
<td>Returns the name of the recipient batch being processed.</td>
</tr>
</tbody>
</table>

**See also**

- RecipCopyCount on page 354
- RecipName on page 356
- Printer and Recipient Functions on page 76
**RecipCopyCount**

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

**Syntax**

```
RecipCopyCount (Recip, Section, Form, Group)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recip</td>
<td>(Optional) Enter the names of the recipients you want included in the count.</td>
</tr>
<tr>
<td>Section</td>
<td>Enter the names of the sections you want the function to look through.</td>
</tr>
<tr>
<td>Form</td>
<td>(Optional) Enter the names of the forms you want the function to look through.</td>
</tr>
<tr>
<td>Group</td>
<td>(Optional) Enter the names of the groups you want the function to look through.</td>
</tr>
</tbody>
</table>

If a recipient has a zero copy count, it is omitted from the total. For instance, if there are three recipients, all with a zero copy count, zero (0) is returned.

**NOTE:** The recipient list this function uses is the same one that generates the POLFile. The list is not re-generated from the POLFile, therefore if any changes occurred in the POLFile, those changes would not be represented in the internal list.

**Example**

Here is an example:

```
RecipCopyCount(Recip,Section,Form,Group)
[ReqType:i_Check]
  function=atcw32->ATCLogTransaction
  function=atcw32->ATCLoadAttachment
  function=dprw32->DPRSetConfig
  function=atcw32->ATCUnloadAttachment
  function=dprw32->DPRCheck
```

**See also**

- RecipBatch on page 353
- RecipName on page 356
- Have Functions on page 69
**RecipientName**

Use this function to return from the FORM.DAT file the recipient name related to the specified section, form, or group.

You can use this function along with the HaveRecip function in DAL scripts to place a sequence number on each page of each recipient batch.

**Syntax**

```
RecipientName (Count, Section, Form, Group)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>An indexed reference to locate a recipient in the FORM.DAT file. The default is the first recipient in the FORM.DAT file.</td>
</tr>
<tr>
<td>Section</td>
<td>Enter the name of a section that contains the recipient. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of a form that contains the recipient. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of the form group that contains the recipient. The default is the current group.</td>
</tr>
</tbody>
</table>

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

If the section, form, or group can not be located or the Count parameter causes the system to move beyond the last recipient in the FORM.DAT file for the section, form, or group, an empty string is returned.

**See also**

HaveRecip on page 274

Name Functions on page 74
**RecipName**

Use this function to get the name of the recipient batch record for the transaction currently being printed. This function is only applicable to batch banner processing or comment record processing with the GenPrint program.

**Syntax**

```
RecipName ( )
```

There are no parameters for this function.

**Example**

Here is an example. Assume the transactions for the Insured batch are being processed.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>rb = RecipName();</td>
<td>Insured</td>
<td>Returns the name of the recipient batch being processed.</td>
</tr>
</tbody>
</table>

**See also**

- RecipCopyCount on page 354
- RecipBatch on page 353
- Printer and Recipient Functions on page 76
**Refresh**

Use this procedure to refresh or repaint the screen.

Syntax: `Refresh()`

There are no parameters for this procedure.

Use this procedure with the `AppendTxm`, `AppendText`, `DelLogo`, `Logo`, and `ChangeLogo` procedures. The result from these procedures may not immediately display. Use the `Refresh` procedure to repaint the screen and display the text or graphic (LOG).

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

**Example**

Here is an example:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Refresh()</code></td>
<td>Repaints the screen.</td>
<td>New graphics or text now appears.</td>
</tr>
</tbody>
</table>

See also:
- Documaker Workstation Functions on page 59
- `AppendText` on page 132
- `AppendTxm` on page 134
- `DelLogo` on page 214
- `Logo` on page 303
- `ChangeLogo` on page 163
**REMOVEATTACHVAR**

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

**Syntax**

```
REMOVEATTACHVAR (Name, DSIqueue)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter the name of the attachment variable.</td>
</tr>
<tr>
<td>DSIqueue</td>
<td>(Optional) Enter one (1) for input or two (2) for output. The default is one (1).</td>
</tr>
</tbody>
</table>

The system returns one (1) if the variable was found and zero (0) if it was not found.

See also

- Docupresentment Functions on page 60
- AddAttachVAR on page 114
- GetAttachVAR on page 252
**RENAMELOGO**

Use this procedure/function to rename a graphic (LOG).

**Syntax**

`syntax RenameLogo (Graphic, NewName, Section, Form, Group)`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphic</td>
<td>Enter the name of the graphic you want to rename. Graphic names are assigned in Studio or Image Editor.</td>
</tr>
<tr>
<td>NewName</td>
<td>Enter the new name for the graphic.</td>
</tr>
<tr>
<td>Section</td>
<td>Enter the name of the section that contains the specified graphic. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of the form that contains the section. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of the group to use to locate the specified object. The default is the current group.</td>
</tr>
</tbody>
</table>

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

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

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

You must specify both the Graphic and NewName parameters.

**Example**

Here are some examples:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>RenameLogo(&quot;Log1&quot;, &quot;Jane Doe&quot;)</td>
<td>1 or 0</td>
<td>Renames Log1 (on the current section, form, and group) to Jane Doe.</td>
</tr>
<tr>
<td>RenameLogo(&quot;Log1&quot;, &quot;Jane Doe&quot;, &quot;IMH1\3&quot;, &quot;UpRate&quot;)</td>
<td>1 or 0</td>
<td>Renames Log1 (on the 3rd occurrence of the named section, IMH1, on the form, UpRate) to Jane Doe.</td>
</tr>
</tbody>
</table>

**See also**

- ChangeLogo on page 163
- DelLogo on page 214
- HaveLogo on page 272
- InlineLogo on page 282
- Logo on page 303
- Name Functions on page 74
**ResetFld**

Use this procedure/function to delete the data from a variable field, including multi-line variable fields. This procedure works even if no data was entered into the field.

**Syntax**

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>Enter the name of the field you want to reset. Enclose the field name in quotation marks. Here is an example:</td>
</tr>
<tr>
<td></td>
<td>“FIELD01”</td>
</tr>
<tr>
<td>Section</td>
<td>Enter the name of the section that contains the field name. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of a form that contains the section or field name or both. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of the form group that contains the form, section, and/or field name. The default is the current group.</td>
</tr>
</tbody>
</table>

**Example**

Here are some examples:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResetFld (“ACCUM_TOT”)</td>
<td>1 or 0</td>
<td>Clears the data from the ACCUM_TOT field.</td>
</tr>
<tr>
<td>ResetFld (“YEARTODATE”)</td>
<td>1 or 0</td>
<td>Clears the data from the YEARTODATE field.</td>
</tr>
<tr>
<td>ResetFld (“TOTAL_PREM”, “BOAT PREM”)</td>
<td>1 or 0</td>
<td>Clears the data in the field, TOTAL_PREM, in the section, BOAT PREM.</td>
</tr>
</tbody>
</table>

**See also** Field Functions on page 61
**RESETOVFLWSYM**

Use this procedure/function to reset the value in an overflow symbol to zero.

**Syntax**

```
ResetOvFlwSym (Form, Symbol)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form</td>
<td>Enter the name of the form that contains the fields on which overflow processing will occur.</td>
</tr>
<tr>
<td>Symbol</td>
<td>Enter the name you want to use as the overflow symbol.</td>
</tr>
</tbody>
</table>

The system optionally returns one (1) on success or zero (0) on failure. This procedure stores a zero (0) in the specified overflow symbol.

**NOTE:** This procedure provides DAL with the functionality included in Documaker Server’s ResetOvFlw and ResetOvSym rules.

**Example**

Here is an example:

```
#reset_rc = ResetOvFlwSym ("CP0101NL", "Loc_Cnt")
```

In this example, the overflow symbol, `Loc_Cnt`, is set to zero and the DAL integer variable, `#reset_rc`, is set to a one (1) on success and zero (0) on failure.

**Syntax**

- AddOvFlwSym on page 127
- GetOvFlwSym on page 262
- IncOvFlwSym on page 280
- Documaker Server Functions on page 58
RETAI N

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

Syntax

\texttt{Retain \{Variable\}}

Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>Enter the names of the DAL variables (as a quoted string) you want to retain during the processing of transactions.</td>
</tr>
</tbody>
</table>

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:

\begin{verbatim}
< RunMode >
  FlushDALSymbols = Yes
\end{verbatim}

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.

\textbf{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:

\begin{verbatim}
$\texttt{total\_amt} = \texttt{Sum("$\texttt{prem\_"});}
\texttt{Retain \{"$\texttt{total\_amt}"\};}
\end{verbatim}

In this example, the DAL variable $\texttt{total\_amt}$ will survive transaction boundaries and can be referenced in any subsequent transaction DAL script.

See also

- Using INI Options on page 8
- Miscellaneous Functions on page 73
**RIGHT**

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

**Syntax**

\[
\text{Right} \ (\text{String}, \ \text{Integer})
\]

**Parameter** | **Description**
---|---
String | Enter a valid string. The default is the value of the current field text.
Integer | Enter the desired length for the output. The default is the length of the String parameter.

If the length you specify in the integer parameter is longer than the string, the system pads the result with spaces to reach the requested length. The input string is first trimmed of leading and trailing spaces before the output is determined.

**Example**

Here are some examples:

(Assume the current field contains the text *Your Name*.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Right ( )</td>
<td>“Your Name”</td>
<td>Defaults to the current field; No length was specified; therefore the field remains the same.</td>
</tr>
<tr>
<td>Right (“ est text”, 9)</td>
<td>“test text”</td>
<td>Takes the nine right most characters from the specified field and returns the result.</td>
</tr>
<tr>
<td>Right (“Complete Street Address”, 14)</td>
<td>“Street Address”</td>
<td>Takes the 14 right most characters from the specified field and returns the result.</td>
</tr>
</tbody>
</table>

See also

- String Functions on page 78
- Left on page 296
**ROOTNAME**

Use this function to extract and return the root name, or the original part of the name, of a string you specify. This function strips off the #nnn portion of a field name to get the root field name.

**NOTE:** Documaker requires that all fields on a section be uniquely named. Studio and Image Editor force a unique name if a field is duplicated. Appending #002 or #003, for example, to the end of the field name creates unique names. In some cases you may want to use the name of a field to supply the name of a data dictionary symbol to use to fill that field. If each unique instance of a field is to use the same name, this can present a problem.

### Syntax

```
RootName (Field)
```

### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>Enter the name of the field for which you want the system to return the root portion of that name.</td>
</tr>
</tbody>
</table>

### Example

Here are some examples:

```text
RootName("Street address #002")
```

This returns *Street address.*

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

This returns *Comment.*

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

This returns the root name of the current field.

### See also

[Name Functions on page 74](#)
**Round**

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

**Syntax**

`Round(Number, Places)`

The system returns the string value of a decimal number rounded to the number of places specified. The sign of the number is not changed. Decimal numbers maintain up to 14 digits of precision. The Round function returns the value with or without trailing zeros requested. If you use the result the Round function returns in a mathematical equation or to represent a decimal parameter, the string is implicitly converted as needed.

**Example**

Here are some examples:

(Assume the current field value is 23.5473)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Round ( )</code></td>
<td>23.55</td>
<td>Defaults to the current field and to two decimal places.</td>
</tr>
<tr>
<td><code>Round ( , 3)</code></td>
<td>23.547</td>
<td>Defaults to the current field and uses three decimal places.</td>
</tr>
<tr>
<td><code>Round (101.999, 0)</code></td>
<td>102</td>
<td>Rounds the given value to zero decimal places.</td>
</tr>
<tr>
<td><code>Round (101.999, 4)</code></td>
<td>101.9990</td>
<td>Rounds the given value to four decimal places.</td>
</tr>
</tbody>
</table>

**NOTE:** When using the result of the Round function to assign a section field value, make sure the numeric field is defined *without* a format. If the field has a format, it may override the text provided by this function.

**See also**

[String Functions on page 78](#)
**RouteWIP**

Use this procedure/function to send all the work contained in WIP to all the recipients specified in a routing slip.

**Syntax**

\[ \text{RouteWIP (Slip)} \]

**Parameter** | **Description**
---|---
Slip | Enter the name of a routing slip. The default is to let the system display a window that lets the user select a routing slip

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

If the WIP is already following a routing slip’s workflow, the form set is sent to the next recipient in the existing slip.

**Example**

Here are some examples:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>RouteWIP( )</td>
<td>1 or 0</td>
<td>Displays for the user the Routing Slip Selection window.</td>
</tr>
<tr>
<td>RouteWIP (&quot;manager&quot;)</td>
<td>1 or 0</td>
<td>This specifies the routing slip named, <em>manager</em>. 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.</td>
</tr>
</tbody>
</table>

**See also**  
WIP Functions on page 88
RPErrorMsg

Use this procedure to write an error message into Documaker Server's error file (ERRFILE.DAT). In addition, it increments the Documaker Server error count as necessary.

Syntax

RPErrorMsg (Message)

Parameter Description

| Message | (Optional) Enter the message you want the system to display. The message can consist of static text, DAL functions, and procedures. The default is a blank string. |

Example

Here is an example:

```plaintext
RPErrorMsg ( )
```

This example would cause the following to be written into your ERRFILE.DAT file:

```
Error : In DAL Script <.\deflib\iso_create.dal> at line <23>: (Text omitted)
```

Here is another example:

```plaintext
RPErrorMsg ("Failed to Open the INFO table in iso_create.")
```

This example would cause the following to be written into your ERRFILE.DAT file:

```
Error : In DAL Script <.\deflib\iso_create.dal> at line <18>: Failed to Open the INFO table in iso_create.
```

Here is another example:

```plaintext
RPErrorMsg (Time() & " " & Date() & " variable = " & table_name)
```

This example would cause the following to be written into your ERRFILE.DAT file:

```
Error : In DAL Script <.\deflib\iso_create.dal> at line <26>: 11:51:02 08/06/2003 variable = INFO
```

See also

RPErrorMsg on page 368
RPWarningMsg on page 369
Documaker Server Functions on page 58
RPLOGMSG
Use this procedure to write a message into Documaker Server’s log file
(LOGFILE.DAT).
Syntax

Example

RPLogMsg (Message)

Parameter

Description

Message

(Optional) Enter the message you want the system to display. The message can
consist of static text, DAL functions, and procedures. The default is a blank
string.

Here are some examples:
RPLogMsg ( )

This example would cause the following to be written into your LOGFILE.DAT file:
Message : In DAL Script <.\deflib\iso_create.dal> at line <23>: (Text
omitted)

Here is another example:
RPLogMsg ("Failed to Open the INFO table in iso_create.")

This example would cause the following to be written into your LOGFILE.DAT file:
Message : In DAL Script <.\deflib\iso_create.dal> at line <18>:
Failed to Open the INFO table in iso_create.

Here is another example:
RPLogMsg (Time() & " " & Date() & " variable =

" & table_name)

This example would cause the following to be written into your LOGFILE.DAT file:
Message : In DAL Script <.\deflib\iso_create.dal> at line <26>:
11:51:02 08/06/2003 variable = INFO

See also

RPErrorMsg on page 367
RPWarningMsg on page 369
Documaker Server Functions on page 58

368


**RPWarningMsg**

Use this procedure to write a warning message into the Documaker Server error file (ERRFILE.DAT). In addition, it increments the Documaker Server warning count as necessary.

**Syntax**

```
RPWarningMsg (Message)
```

**Parameter** | **Description**
--- | ---
Message | (Optional) Enter the message you want the system to display. The message can consist of static text, DAL functions, and procedures. The default is a blank string.

**Example**

Here are some examples:

```
RPWarningMsg ( )
```

This example would cause the following to be written into your ERRFILE.DAT file:

```
Warning : In DAL Script <\deflib\iso_create.dal> at line <23>: (Text omitted)
```

Here is another example:

```
RPWarningMsg ("Failed to Open the INFO table in iso_create.")
```

This example would cause the following to be written into your ERRFILE.DAT file:

```
Warning : In DAL Script <\deflib\iso_create.dal> at line <18>:
Failed to Open the INFO table in iso_create.
```

Here is another example:

```
RPWarningMsg (Time() & " " & Date() & " variable = " & table_name)
```

This example would cause the following to be written into your ERRFILE.DAT file:

```
Warning : In DAL Script <\deflib\iso_create.dal> at line <26>:
11:51:02 08/06/2003 variable = INFO
```

**See also**

- RPErrorMsg on page 367
- RPLogMsg on page 368
- Documaker Server Functions on page 58
**SAVEINIFile**

Use this procedure/function to save a set of INI control groups and options that were loaded into cache memory.

**Syntax**

```
SaveINIFile (Context, File)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context</td>
<td>(Optional) A name (valid string) associated with a set of INI control groups and options loaded into cache memory.</td>
</tr>
<tr>
<td>File</td>
<td>Enter the name of the file in which you want to store the specified set of INI control groups and options. If you omit the file extension, the system uses <code>INI</code>. If you omit the path, the system stores the file in the current directory.</td>
</tr>
</tbody>
</table>

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

If a context name is associated with the execution of this procedure, that set of INI control groups and options will be stored in the specified physical file name.

**Example**

Here are some examples:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SaveINIFile (&quot;DALRun&quot;);</td>
<td>The set of INI control groups and options are saved in a file.</td>
<td>The INI control groups and options are saved to the specified file. Execution of this procedure assumes that the file extension is <code>INI</code>.</td>
</tr>
<tr>
<td>SaveINIFile (&quot;Run_process&quot;, &quot;DALRun.ini&quot;);</td>
<td>The set INI control groups and options referenced by the context name, <code>Run_process</code>, are saved in a file.</td>
<td>The INI control groups and options are saved to the specified file.</td>
</tr>
</tbody>
</table>

**See also**

INI Functions on page 70
Using INI Options on page 8
**SaveWIP**

Use this procedure/function to save the WIP record being processing. Optionally, this procedure returns a one (1) on success or a zero (0) on failure. This procedure is needed in the DAL script called by the Documaker Workstation function, AFEBatchDALProcess, if you change any data in the WIP record being processed.

**Syntax**

```
SaveWIP ( )
```

There are no parameters for this procedure.

**Example**

Here is a sample DAL script.

```
desc_field   = WIPFld("DESC");
mod_data     = desc_field & " – 04/03/03";
rc_setwipfld = SetWIPFld("DESC", mod_data);
rc_savewip   = SaveWIP( );
```

This script appends the text, – 04/03/03, to the content of the DESC field in each WIP record in the WIP.DBF file.

**See also**

Executing a DAL Script from a Menu on page 7
**SECOND**

Use this function to extract the number of seconds in a time.

**Syntax**

```
Second (Time, Format)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time</strong></td>
<td>Enter a valid time string. The system assumes your entry is in the time format specified in the Format parameter. The default is the current time.</td>
</tr>
<tr>
<td><strong>Format</strong></td>
<td>Enter a valid time format string that describes the Time parameter. The default is time format 1 (HH:MM:SS).</td>
</tr>
</tbody>
</table>

**Example**

Here are some examples:

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Second( )</td>
<td>09</td>
<td>Defaults to the current time and extracts 09.</td>
</tr>
<tr>
<td>Second (&quot;09:20:20&quot;)</td>
<td>20</td>
<td>Reads the given time and extracts 20.</td>
</tr>
</tbody>
</table>

**See also**

Time Functions on page 80
**SetDeviceName**

Use this procedure to set a new output device file name which will be used the next time the output device is opened, assuming nothing overrides the name prior to that. You can use this function when splitting recipient batches into multiple print stream files.

**Syntax**

```
SetDeviceName (Device)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>Enter the new output device file name.</td>
</tr>
</tbody>
</table>

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 ` FullName` for information on using DAL functions to manipulate file names.

Keep in mind...

- The print drivers supported are: PCL5, PCL6, PST, MET, AFP, PDF, HTML, and RTF.
- These print drivers are not supported: EPT, MDR, and GDI.
- All platforms are supported, but note that while UniqueString is supported on z/OS, z/OS does not support PDF or long file names.
- Both multi-step and single-step processing are supported.

The only DAL function actually involved in splitting the print stream is BreakBatch. The others make it easier to implement this functionality. For example, since you need to name the new print stream, you use the SetDeviceName procedure. To find the name of the current device, you use the DeviceName function. If you need to create unique file names, you can use the UniqueString function.

**NOTE:** While you can call all of these DAL functions in Documaker Server or Entry, the BreakBatch and SetDeviceName functions are not applicable in Entry since it does not use the batch printing engine. The other functions, DeviceName and UniqueString, are applicable to both Entry and Documaker Server.
See also

- Printer and Recipient Functions on page 76
- BreakBatch on page 158
- DeviceName on page 217
- UniqueString on page 421
**SetEdit**

Use this procedure/function to determine which field should be the next active field during normal entry. Normal entry refers to tabbing from field to field. If a user mouse clicks a particular field or pages between sections, the field selected by the SetEdit procedure is ignored. This procedure optionally returns one (1) on success or zero (0) on failure.

**Syntax**

```
SetEdit (Field, Count, Section, Form, Group)
```

This procedure first locates the specified field. If you omit the Field parameter, the system uses the current field.

You can use a positive or negative number for the count parameter. A positive count moves forward from the located field. A negative count moves backward from the located field. Forward and backward refer to the order in which the field appears in the section’s edit list, not necessarily its physical position on the section. Do not include fields designated as display only in the count.

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

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

This procedure returns one (1) if it finds the specified field. Otherwise, it returns zero (0).

**NOTE:** The navigation logic you enter on the Navigation tab of the field’s Properties window overrides this procedure.

Here are some examples:
Assume the section has three fields named FIRST, SECOND, and THIRD. The fields occur in that order.

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetEdit(&quot;THIRD&quot;)</td>
<td>1</td>
<td>Locates the field named THIRD on the current section. If found and if editable, that field will be the next field to receive focus.</td>
</tr>
<tr>
<td>SetEdit(&quot;THIRD&quot;,-2)</td>
<td>1</td>
<td>Locates the field named THIRD on the current section. If found, moves two fields prior to THIRD--to the field named FIRST.</td>
</tr>
<tr>
<td>SetEdit(&quot;MyField&quot;, &quot;FRM&quot;)</td>
<td>1 or 0</td>
<td>Locates the form named FRM in the current form group. Then locates MyField on that form. If found, focus changes to that form and field.</td>
</tr>
</tbody>
</table>

See also

- **Documaker Workstation Functions on page 59**
- **Field Formats on page 62**
- **Locating Fields on page 64**
**SetFld**

Use this procedure/function to assign a value to a section field. Normally, this procedure is used to assign values to display only fields or to assign default values to fields which have not yet been edited.

**Syntax**

```
SetFld (String, Field, Section, Form, Group)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Enter a value appropriate for the field you are assigning. The default is an empty string.</td>
</tr>
<tr>
<td>Field</td>
<td>Enter the name of a field. The default is the current field.</td>
</tr>
<tr>
<td>Section</td>
<td>Enter the name of a section that contains the field named. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of a form that contains the section and/or field named. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of the form group that contains the form, section, and/or field named. The default is the current group.</td>
</tr>
</tbody>
</table>

The system returns one (1) if successful or zero (0) if the field cannot be changed or does not exist.

This procedure attempts to change the field’s text each time the script executes. Therefore, use the SetFld procedure with discretion. Do not use the SetFld procedure if the script should not execute each time a user highlights a new field.

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

If you are using the SetFld procedure in a batch system execution and you are trying to set a field other than the one which initiated the rule, you must load the FAP file. To do so, add the CheckImageLoaded rule to the sections to which you plan to assign fields.

Trailing spaces are deleted from the string to be stored. If you need the spaces, use a hard space (ALT + 0160). See the Rules Reference for more information about this rule.

**Example**

Here are some examples:

(Assume the section has three fields (First, Second, Third). The value of First is 123.)

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetFld(“N/A”, “SECOND”)</td>
<td>1</td>
<td>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.</td>
</tr>
</tbody>
</table>
IF (!SetFld(101, "MyField") )
  MSG("Field" & "MyField", "not assigned")
END

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")

This statement attempts to assign the value of the current field to MyField, located on the specified FRM. Since a section name was not given, the field may occur on any section on that form.

See also
Field Functions on page 61
Field Formats on page 62
Locating Fields on page 64
SetFont

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

Syntax

\[ \text{SetFont} \left( \text{FontID}, \text{Field}, \text{Section}, \text{Form}, \text{Group} \right) \]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FontID</td>
<td>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.</td>
</tr>
<tr>
<td>Field</td>
<td>(Optional) Enter the name of a field that identifies a multi-line text area. This is the field that receives the appended text. The default is the current field.</td>
</tr>
<tr>
<td>Section</td>
<td>(Optional) Enter the name of the section that contains the field you specified. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>(Optional) Enter the name of the form that contains the section and/or field you specified. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>(Optional) Enter the name of the form group that contains the form, section, and/or field you specified. The default is the current group.</td>
</tr>
</tbody>
</table>

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

The system applies the font change to the first field that matches the criteria.

See also  
Field Functions on page 61
**SetFormDesc**

Use this function to change the description of a form.

**Syntax**

SetFormDesc (NewDescription, Form, Group)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewDescription</td>
<td>Enter the new description. The text you enter replaces any existing form description.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of the form for which you want to change its description. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of the group which contains the form you specified in the Form parameter. The default is the current group.</td>
</tr>
</tbody>
</table>

The system returns one (1) if the form was found and the description was assigned. Otherwise, it returns zero (0) to indicate that no form was found based upon the parameters you provided.

**Example**

Here is an example:

SetFormDesc("Cover Page", Form1, Group2)

**See also**

Name Functions on page 74
SetGVM

Use this procedure/function to update the contents of a GVM variable. You can also use this procedure to create a GVM variable.

Syntax

\[
\text{SetGVM (Name, Data, Instance, Type, Size)}
\]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter a string which contains the name of the GVM variable.</td>
</tr>
<tr>
<td>Data</td>
<td>Enter the data you want to store in the GVM variable.</td>
</tr>
<tr>
<td>Instance</td>
<td>Enter the instance number of the GVM variable. The default is one (1)</td>
</tr>
<tr>
<td>Type</td>
<td>Indicate the type of GVM variable to create. You can choose from these options: C - Character array S - Short L - Long F - Float D - Double Q - Long double</td>
</tr>
<tr>
<td>Size</td>
<td>Enter the number of bytes to reserve when creating a GVM variable. This parameter is not used if the GVM already exists.</td>
</tr>
</tbody>
</table>

The system returns a one (1) if successful or a zero (0) if not.

**NOTE:** You can use this function to set a reserved GVM value, but be aware of how that reserved GVM is used. Some reserved GVM values should not be modified, such as NA_OFFSET and POL_OFFSET. Additionally, keep in mind that reserved GVM values may be changed by subsequent rule processing.

Example

Here are some examples:

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

See also

HaveGVM on page 270
DAL Script Examples on page 35
Documaker Server Functions on page 58
**SetImagePos**

Use this procedure/function to reposition a section on a page.

**Syntax**

```
SetImagePos (PrefixName, Section, Form, Group)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrefixName</td>
<td>A prefix name to be associated with the coordinates returned by the procedure.</td>
</tr>
<tr>
<td>Section</td>
<td>Enter the name of a section in the form set. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of a form in the form set that contains the section. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of the form group that contains the form and section you specified. The default is the current group.</td>
</tr>
</tbody>
</table>

This procedure repositions a section at the coordinates you specify in the PrefixName parameter: `prefix.name.top`, `prefix.name.left`.

**NOTE:** The section remains the same size.

This procedure retrieves these variables and sets the section’s top coordinate to `prefix.name.top` and its left coordinate to `prefix.name.left`. This procedure returns a bad variable error message if the `prefix.name.top` or `prefix.name.left` variables are not defined as DAL internal variables.

**Example**

For this example, assume the current section is *Image25*, the form is *Input_form*, and the form group is *Package1*. The coordinates are:

<table>
<thead>
<tr>
<th>Image25</th>
<th>For internal variables</th>
<th>Image50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top</td>
<td>25</td>
<td>125</td>
</tr>
<tr>
<td>Left</td>
<td>50</td>
<td>150</td>
</tr>
</tbody>
</table>

**NOTE:** The the Bottom-Right coordinate is automatically calculated from the new Top-Left coordinate by adding the section height and width, which are not changed by this DAL function.

**Procedure** | **Result** | **Explanation**
---|---|---
SetImagePos ("MyImage") | New coordinates for the current section, *Image25*, will be:
  - Myimage.top = 125
  - Myimage.left = 150
  - Myimage.bottom = 200
  - Myimage.right = 200 | Sets the coordinates for the current section to the internal DAL variables: Myimage.top, Myimage.left, Myimage.bottom, and Myimage.right.
This script takes the coordinates of the section named MyImage and sets them to the variables MyRect.Top, MyRect.Left, MyRect.Bottom, and MyRect.Right. Next, it increases MyRect.Top by 2400 FAP units then moves MyImage one inch (2400 FAP units) lower on the page.

See also Section Functions on page 77

ImageRect on page 278
**SetLink**

Use this function to update a hyperlink setting in a variable field, a graphic, or a text label.

**Syntax**

```
SetLink (Target, Parms, ObjectName, Section, Form, Key2, ObjectType)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Target**    | Enter the name of the target object (the HREF value). If the target object has a hyperlink type of *internal* or *target*, enter the name of the target object. If the target object has a hyperlink type of *external*, this parameter should contain a hypertext reference, such as:  
  - www.oracle.com and the Parms parameter should contain additional parameters to an HREF type link. Make sure this parameter contains valid HTML syntax. |
| **Parms**     | (Optional) Enter any link parameters (HREF parameters), such as a target frame or mouseover behavior. Here is an example:  
  - "target=new"  
  Make sure this parameter contains valid HTML syntax. |
| **ObjectName**| Enter the name of the variable field, graphic, or text label that contains the hyperlink. The system updates the first object found that matches your entry for this parameter. |
| **Section**   | (Optional) Enter the name of the section.                                                                                                     |
| **Form**      | (Optional) Enter the name of the form.                                                                                                         |
| **Key2**      | (Optional). Enter the name of the Key2 group.                                                                                                  |
| **ObjectType**| Enter the type of object, such as (variable) Field, Graphic, or Text (label). The default is Field.                                             |

Keep in mind...

- The object (variable field, graphic, or text label) referenced by SetLink must have an initial hyperlink setting.
- You must make sure the Target and Parms parameters contain valid HTML syntax.

**Example**

Here is an example:

```
SETLINK("http://www.oracle.com", "target=new", "Section2256", "FormQ1331TPG", , , "Text")
```

**See also**  
Field Functions on page 61
SetLogo

This function is obsolete and is no longer supported. Use the ChangeLogo function instead.

See also

ChangeLogo on page 163

Graphics Functions on page 71
**SetProtect**

Use this procedure/function to protect a specified field so it cannot be altered or to unprotect a field so that it can be edited.

**Syntax**

```
SetProtect (Mode, Field, Section, Form, Group)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>Enter a non-zero value to specify field protection mode. Enter zero (0) to leave the field unprotected. The default is one (1), which protects the field.</td>
</tr>
<tr>
<td>Field</td>
<td>Enter the name of the field. The default is the current field.</td>
</tr>
<tr>
<td>Section</td>
<td>Enter the name of a section that contains the field you specified. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of a form that contains the section and/or field named. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of the form group that contains the form, section, and/or field named. The default is the current group.</td>
</tr>
</tbody>
</table>

The system returns zero (0) if the field you specified could not be changed or does not exist in the section. The system returns one (1) if the field was successfully protected.

**Example**

Here are some examples:

(Assume the section has fields named First and Second. Assume First contains Y.)

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| IF (@(  ) ="Y")  
SetProtect( 
  1, "SECOND");  
END | 1 | Tests the value of the current field (First). Since it contains the letter Y, Second is protected. If you call SetProtect as a procedure, a one (1) is returned, indicating the field was successfully protected. |
| IF (@(  ) ="Y")  
SetProtect( 
  0, 
  "SECOND");  
END | 1 | Unprotects Second based on the same criteria. |

**See also**

Field Functions on page 61  
Field Formats on page 62  
Locating Fields on page 64
**SetRecip**

Use this procedure/function to assign the recipient copy count for a particular section, form, or group.

**Syntax**

```
SetRecip (Recipient, Count, Section, Form, Group)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recipient</td>
<td>Enter a valid recipient name for the sections you want to change.</td>
</tr>
<tr>
<td>Count</td>
<td>Enter the total number of copies of the designated sections that you want this recipient to receive. The default is zero (0).</td>
</tr>
<tr>
<td>Section</td>
<td>Enter the name of the section you want to locate. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of the form that contains the section you specified. The default is the current section.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of the group that contains the section or form you specified. The default is the current section.</td>
</tr>
</tbody>
</table>

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

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

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

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

**Example**

Here are some examples:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetRecip (&quot;Insured&quot;, 2)</td>
<td>1 or 0</td>
<td>Defaults to the current section. If this section includes Insured as a recipient, that copy count will be assigned 2.</td>
</tr>
<tr>
<td>SetRecip(&quot;HOME OFFICE&quot;, 1, &quot;FORM&quot;)</td>
<td>1 or 0</td>
<td>Locate FORM in the current group. Assign any section that specifies HOME OFFICE as a recipient the new copy count of one (1).</td>
</tr>
</tbody>
</table>

**See also** [Section Functions on page 77](#)
**SETREQUIREDFLD**

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

**Syntax**

```
SetRequiredFld (Required, Field, Section, Form, Group)
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td>Enter Yes if you want to make the field required. Enter No if you want to make the field optional.</td>
</tr>
<tr>
<td>Field</td>
<td>(Optional) Enter the name of the field. The default is the current field.</td>
</tr>
<tr>
<td>Section</td>
<td>(Optional) Enter the name of the section. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>(Optional) Enter the name of the form. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>(Optional) Enter the name of the group. The default is the current group.</td>
</tr>
</tbody>
</table>

**Example**

Here are some examples:

```
SetRequiredFld ("Yes", "Myfield", :MyImage", "Myform", "MyGroup");
SetRequiredFld ("Yes", "Myfield", :MyImage", "Myform", );
SetRequiredFld ("Yes", "Myfield", :MyImage", );
SetRequiredFld ("Yes", "Myfield", );
SetRequiredFld ("Yes", );
```

If you include the Section parameter, but omit the field parameter, the system uses the first field on that section. If you omit the Section and Field parameters, but include the Form, the system looks for the first field on the first section of the form you specified, and so on.

**See also**

Field Functions on page 61
Field Formats on page 62
**SETWIPFld**

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

**Syntax**

```
SetWIPFld (Field, Data)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>Enter the name of the variable field.</td>
</tr>
<tr>
<td>Data</td>
<td>Enter the data you want to store in the field.</td>
</tr>
</tbody>
</table>

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

The system returns one (1) if successful or zero (0) if the field cannot be changed or does not exist.

**Example**

Here are some examples:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetWIPFld (&quot;DESC&quot;, &quot;My Description&quot;)</td>
<td>1 or 0</td>
<td>Assigns to the WIP description field a new description.</td>
</tr>
<tr>
<td>SetWIPFld(&quot;DESC&quot;)</td>
<td>1 or 0</td>
<td>Clears the WIP description field.</td>
</tr>
</tbody>
</table>

**See also** [WIP Functions on page 88](#)
**Size**

Use this function to return the defined length of a specified field.

**Syntax**

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

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>Enter the name of the variable field. The default is the current field.</td>
</tr>
<tr>
<td>Section</td>
<td>Enter the name of the section that contains the field you specified. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of a form that contains the section and/or field you specified. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of the form group that contains the form, section, and/or field you specified. The default is the current group.</td>
</tr>
</tbody>
</table>

The system returns the length of the defined data area for the specified field.

The `Size` function is often confused with the `LEN` function. The `LEN` function returns the length of the actual data contained in a field or DAL variable.

**Example**

Here are some examples:

(Assume the current field contains the text `Your Name` and its defined length is 15.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Size ()</code></td>
<td>15</td>
<td>Returns the defined length of the current field.</td>
</tr>
<tr>
<td><code>Size (&quot;Myfield&quot;, &quot;FRM&quot;)</code></td>
<td>field size or zero</td>
<td>This example will look for MyField on the form, FRM. It may occur on any section. If the field is located, its size will be returned, otherwise the result is zero (0). Generally, you can assume that a zero result means the field is not defined, since it is unlikely that a field of zero length would be legitimate.</td>
</tr>
</tbody>
</table>

**See also**

- Field Formats on page 62
- Locating Fields on page 64
- LEN on page 297
- Field Functions on page 61
**SLIPAPPEND**

Use this procedure/function to add an email address to the end of the routing slip associated with the form set.

**Syntax**

```
SlipAppend (Address, Mode)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>Enter an email address.</td>
</tr>
</tbody>
</table>
| Mode      | Choose from these options:  
0 - linear recipient  
1 - CC recipient  
The default is zero (0). |

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

This procedure only works with scripts associated with routing slips. This procedure lets you, via scripts, direct workflow during the routing process. Do not use this procedure in a typical field script situation.

The address name is appended to the end of the current routing slip. If the mode parameter is not zero (0), the new entry is appended as a carbon-copy (CC) recipient. For example, assume the following routing slip is defined:

```
CC Recipient
@MyScript
EDJ
```

If the script executes the statements, `SlipAppend(“TOM”,1); SlipAppend(“CAR”)`, the slip will be adjusted to look as follows:

```
CC Recipient
* @MyScript EDJ
   EDJ
X  TOM
   CAR
```

**Example**

Here are some examples:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SlipAppend (&quot;TOM&quot;)</code></td>
<td>1 or 0</td>
<td>The email address is appended to the end of the current routing slip. The default is a linear recipient.</td>
</tr>
<tr>
<td><code>SlipAppend(“TOM”,1)</code></td>
<td>1 or 0</td>
<td>Appends the email address as a CC recipient.</td>
</tr>
</tbody>
</table>

**See also**

[WIP Functions on page 88](#)
SLIPINSERT

Use this procedure/function to insert another email address on a routing slip associated with the form set.

**Syntax**

```plaintext
SlipInsert (Address, Mode)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>Enter an email address.</td>
</tr>
<tr>
<td>Mode</td>
<td>Choose from these options: 0 - linear recipient 1 - CC recipient The default is zero (0).</td>
</tr>
</tbody>
</table>

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

This procedure only works with scripts associated with routing slips. This procedure lets you, via scripts, direct workflow during the routing process. It should not be used in a typical field script situation.

The address name is inserted immediately after the script reference in the routing slip. If two SlipInsert statements are executed in order, the second email address appears before the one inserted by the former statement. Think of this as last in, first out.

For example, assume the following routing slip is defined:

```
CC Recipient

@MyScript EDJ
```

If the script executes the statements, `SlipInsert("TOM", 1); SlipInsert("CAR")`, the slip will be adjusted to look as follows.

```
CC Recipient

* @MyScript CAR
X TOM EDJ
```

The asterisk (*) indicates the script has already been executed. If the mode parameter is not zero (0), the new entry is appended as a carbon-copy (CC) recipient.

**Example**

Here are some examples:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SlipInsert (&quot;TOM&quot;)</code></td>
<td>1 or 0</td>
<td>The email address will be inserted immediately after the script reference. The default is a linear recipient.</td>
</tr>
<tr>
<td><code>SlipInsert(&quot;TOM&quot;, 1)</code></td>
<td>1 or 0</td>
<td>Inserts the email address as a CC recipient.</td>
</tr>
</tbody>
</table>

**See also** [WIP Functions on page 88](#)
**SpanField**

Use this function/procedure to move a field horizontally and then resize it to span the distance between two other fields you specify. This function sets the span field’s contents to be enough of a fill character to span the distance.

This function only moves the field horizontally. It will not move the other two fields. The section designer must ensure vertical alignment between the fields.

**NOTE:** If you use this function with resources created prior to version 11.0, which had separate FAP and DDT files, this procedure automatically loads the section (FAP or compiled FAP) if it is not already loaded.

**Syntax**

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpanField</td>
<td>Use this parameter to specify the filler character you want the system to use to span the distance between the end of the left field text and the beginning of the right field text. If either field is empty, the left coordinate of the field is used. The system only uses the first character of the text contained in the field you specify as the filler character. In addition to the filler character, the field you specify also determines the font ID to be used for calculating the number of characters required to fill the width of the field. If there is fractional space remaining in the width, the filler character is duplicated. The extra white space will be placed to the left of the span field, so that the spanned field will be placed against the right-most field. The default is a period (.)</td>
</tr>
<tr>
<td>LeftField</td>
<td>Enter the name of the field on the left of the area you want to span.</td>
</tr>
<tr>
<td>RightField</td>
<td>Enter the name of the field to the right of the area you want to span.</td>
</tr>
<tr>
<td>Section</td>
<td>(Optional) Enter the name of a section that contains the fields you specified. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>(Optional) Enter the name of a form that contains the section and/or field you specified. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>(Optional) Enter the name of the form group that contains the form, section, or fields you specified. The default is the current group.</td>
</tr>
</tbody>
</table>

The SpanField parameter is always the first parameter, but you can specify the LeftField and RightField parameters in any order. The system automatically determines which of the two fields is to the right or left of the span field.
**NOTE:** If you are using the SpanField function in Documaker Server processing, the JustFld rule may be useful to right justify the right-most field to make sure the maximum distance is spanned. If you use the Move_It rule, or other rules that support right justification by padding the data with spaces, the results will be incorrect. The SpanField function calculates the width of a field based upon the entire contents and does not remove space, or any other white space or characters in the fields.

Example

Here is an example:

Assume LeftField contains ABCDEFG, RightField contains $123.45, and SpanField contains a dash (-).

```java
SpanField("SPANFIELD", "LEFTFIELD", "RIGHTFIELD")
```

Yields: ABCDEFG——$123.45

The horizontal location of the span field is adjusted to make sure it is positioned against the right edge of the left field, and then expanded with enough of the fill character to fill the gap between the left and right fields. The section designer is responsible for vertical alignment.

See also

Field Functions on page 61
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:

```plaintext
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**

```plaintext
SrchData (SearchCriteria, Offset, Length, Occurrence)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SearchCriteria</td>
<td>Enter the criteria you want the system to use to look for the data in the extract file.</td>
</tr>
<tr>
<td>Offset</td>
<td>For XML extract files, enter the offset into the data where the desired data starts. For flat files, enter the offset into the record where the data starts. The default is zero (0).</td>
</tr>
<tr>
<td>Length</td>
<td>Enter the number of characters to return. The default is zero (0).</td>
</tr>
<tr>
<td>Occurrence</td>
<td>This parameter is not valid for XML extract files. This parameter lets you specify which occurrence of the data to return. Entering one (1) or zero (0) returns the first occurrence of the data. The default is the first occurrence.</td>
</tr>
</tbody>
</table>

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:

```plaintext
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.

```plaintext
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.

```plaintext
value = SrchData("!/Diamond/Data/Client/Accounts/Account/
Policy/PolicyImages/Policy/premium_fullterm", 1, 7)
If Trim(SrchData("!/Diamond/Data/Client/Accounts/Account/
Policy/PolicyImages/Policy/premium_fullterm", 1, 4) = "2549"
  Then
    Return("equal - " & SrchData("!/descendant::Personalauto/
child::Vehicle[**vehovfsym**]/vehicle_num", 1,2)
  Else
    Return("not equal - " & value)
End
```

See also

GetData on page 253

Documaker Server Functions on page 58
Use this function to return the string value of a field. The @ function automatically converts a numeric format field into its number value. The STR function does not convert field data in any way and returns the value as it appears in the field.

**NOTE:** To consider case in the comparison, use the STRCompare function.

### Syntax

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

### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>Enter the name the field. The default is the current field.</td>
</tr>
<tr>
<td>Section</td>
<td>Enter the name of a section that contains the field named. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of a form that contains the section and/or field named. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of the form group that contains the form, section, and/or field named. The default is the current group.</td>
</tr>
</tbody>
</table>

The system uses the parameters you provide to search for one field on a section and return that field’s data value as formatted. The field can have any format type.

### Example

Here are some examples:

(Assume the current field value is $1,234.23 and is named MyField. Also, assume that a second occurrence of MyField appears on the form, MyForm, and contains the value `automobile`.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>STR( )</td>
<td>$1,234.23</td>
<td>Returns the string value in the current field. Notice that the formatting of the field is not removed.</td>
</tr>
<tr>
<td>STR (&quot;MyField&quot;)</td>
<td>$1,234.23</td>
<td>Returns the string value of the named field, located on the current section.</td>
</tr>
<tr>
<td>STR(&quot;MyField\2&quot;,&quot;MyForm&quot;)</td>
<td>automobile</td>
<td>The second occurrence of MyField already contained a string value.</td>
</tr>
</tbody>
</table>

### See also

- STRCompare on page 398
- Field Functions on page 61
- Field Formats on page 62
- Locating Fields on page 64
- @ on page 109
**STRCompare**

Use this function to compare two strings with case a consideration. In normal DAL string expressions, strings are compared in a case-insensitive manner. For example, the system would normally evaluate the following strings to be equal:

\[
ABC \quad abc
\]

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**

\[
\text{STRCompare} \ (\text{String1}, \text{String2}, \#\text{Count})
\]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String1</td>
<td>Enter the text for the first string you want to compare. The default is an empty string.</td>
</tr>
<tr>
<td>String2</td>
<td>Enter the text for the second string you want to compare. The default is an empty string.</td>
</tr>
<tr>
<td>#Count</td>
<td>(Optional) Enter the number of characters to compare. If you enter a value greater than zero, the system compares that number of characters. If you enter zero (0) or less, the system compares all characters. If you enter a value greater than the length of either string, the system pads the strings with blank characters to match the number of characters you specified. The default is -1 which indicates that all characters will be compared.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>This example</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>#RTN = STRCompare( string1 , string2 )</td>
<td>-1</td>
</tr>
<tr>
<td>#RTN = STRCompare( string2 , string1 )</td>
<td>1</td>
</tr>
<tr>
<td>#RTN = STRCompare( string1 , string2 , 3 )</td>
<td>0</td>
</tr>
</tbody>
</table>
See also  STR on page 397
String Functions on page 78
SUB

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

Syntax

```
SUB (String, Position, Length)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Enter a valid string. The default is the current field.</td>
</tr>
<tr>
<td>Position</td>
<td>Enter the position where sub should begin. The default is one (1).</td>
</tr>
<tr>
<td>Length</td>
<td>Enter the length to retrieve from the text. The default is the length of what remains of the String parameter value, beginning at the position indicated by the Position parameter.</td>
</tr>
</tbody>
</table>

The system returns a portion of the first specified parameter starting at the specified position for the length given.

If you omit the Position parameter, the system defaults to the first character of the string.
If the specified position is greater than the length of the string, the system returns an empty result.
If you omit the Length parameter, the remainder of the string following the specified position is included.

Example

Here are some examples:

(Assume the current field contains the text *Your Name.*)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUB ( , , 5)</td>
<td>“Your “</td>
<td>Defaults to position one of the current field and returns the first five characters.</td>
</tr>
<tr>
<td>SUB ( )</td>
<td>“Your Name”</td>
<td>Defaults to the current field; No length was specified, so the field remains the same.</td>
</tr>
<tr>
<td>SUB (“Complete Street Address”, 10, 6)</td>
<td>“Street”</td>
<td>Goes to position 10 of the specified field and returns six characters.</td>
</tr>
</tbody>
</table>

See also String Functions on page 78
SUM

Use this function to return the decimal sum of a group of fields which have names that begin with common characters.

Syntax

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

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

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

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

- MyField1
- MyField2
- MyField20

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

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

Example

Here are some examples:

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Section</th>
<th>Form</th>
<th>Group</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyField1</td>
<td>IMG A</td>
<td>FRM A</td>
<td>GRP</td>
<td>100.24</td>
</tr>
<tr>
<td>MyField2</td>
<td>IMG A</td>
<td>FRM A</td>
<td>GRP</td>
<td>200.16</td>
</tr>
<tr>
<td>MyField1</td>
<td>IMG B</td>
<td>FRM A</td>
<td>GRP</td>
<td>98.60</td>
</tr>
</tbody>
</table>
(Assume the current field is MyField1, on the first section of the first form. Reference the previous table for field values.)

<table>
<thead>
<tr>
<th>Field</th>
<th>Section</th>
<th>Form</th>
<th>Group</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyField2</td>
<td>IMG B</td>
<td>FRM A</td>
<td>GRP</td>
<td>* no value yet *</td>
</tr>
<tr>
<td>MyField1</td>
<td>IMG A</td>
<td>FRM B</td>
<td>GRP</td>
<td>0.00</td>
</tr>
<tr>
<td>MyField2</td>
<td>IMG A</td>
<td>FRM B</td>
<td>GRP</td>
<td>* no value yet *</td>
</tr>
<tr>
<td>MyField1</td>
<td>IMG B</td>
<td>FRM B</td>
<td>GRP</td>
<td>70.77</td>
</tr>
<tr>
<td>MyField2</td>
<td>IMG B</td>
<td>FRM B</td>
<td>GRP</td>
<td>* no value yet *</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUM ( )</td>
<td>100.24</td>
<td>Without any other information, the function assumes the current field and section. There will only be one value included in the sum.</td>
</tr>
<tr>
<td>SUM (&quot;Myfield2&quot;)</td>
<td>200.16</td>
<td>Again, there is only one field included in this result.</td>
</tr>
<tr>
<td>SUM(&quot;MyField&quot;)</td>
<td>300.40</td>
<td>In this example, the current section contains two fields that begin with the name &quot;MyField&quot;. The equation is as follows: (100.24 + 200.16).</td>
</tr>
<tr>
<td>SUM(&quot;MyField&quot;, &quot;IMG B&quot;)</td>
<td>98.60</td>
<td>Although two fields on IMG B have a matching name, only one field actually has a value.</td>
</tr>
<tr>
<td>SUM(&quot;MyField&quot;, &quot;FRM A&quot;)</td>
<td>399.00</td>
<td>No section is specified in this example, so the entire form is searched. Four fields match the name criteria, but only three have values: (100.24 + 200.16 + 98.60).</td>
</tr>
<tr>
<td>SUM(&quot;MyField&quot;, &quot;IMG B&quot;, &quot;GRP&quot;)</td>
<td>169.37</td>
<td>This example specifies a section and group, but no form. There are four fields that match the name criteria, but only two have values: (98.60 + 70.77).</td>
</tr>
<tr>
<td>SUM(&quot;MyField&quot;, &quot;GRP&quot;)</td>
<td>469.77</td>
<td>This example names the group without a form or section. Eight fields meet the naming criteria, but only five fields actually have values: (100.24 + 200.16 + 98.60 + 0.00 + 70.77).</td>
</tr>
</tbody>
</table>

See also
Mathematical Functions on page 72
Field Formats on page 62
Locating Fields on page 64
**SuppressBanner**

Use this procedure/function to suppress the printing of the banner page. This is useful when you are doing batch banner processing you need to combine several transactions within the same transaction banner pages.

**NOTE:** For information about processing banner pages, see the Documaker Administration Guide.

**Syntax**

```plaintext
SuppressBanner();
```

There are no parameters for this procedure.

**Example**

Here is an example.

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SuppressBanner();</td>
<td></td>
<td>Suppresses the current banner from printing.</td>
</tr>
</tbody>
</table>

You can use this procedure when you want to combine several transactions inside one set of banner pages, based on a flag the DAL script checks.

**See also**

- Printer and Recipient Functions on page 76
- DelBlankPages on page 208
- AddBlankPages on page 115
TABLE

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

Syntax

```
Table (RetCode, Key, Table, File)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RetCode</td>
<td>Enter a return code value designated by the letters K and D. For example: K - key code, D - code description, K + D - key code and code description, D + K - code description and key code. The default is the value of the current field table return value.</td>
</tr>
<tr>
<td>Key</td>
<td>Enter the table key code. The default is the value of the current field text.</td>
</tr>
<tr>
<td>Table</td>
<td>Enter the name of the table you want to search. Note that this parameter is case sensitive. The default is the current field table.</td>
</tr>
<tr>
<td>File</td>
<td>Enter the name of the file that contains the table you specified in the Table parameter. Note that this parameter is not case sensitive. The default is the value of current field table file name, or the current section table file name.</td>
</tr>
</tbody>
</table>

This procedure makes sure a given value (Key) is an entry in the specified table (Table). This procedure returns the string value identified in the RetCode parameter.

The table name in the Table parameter and file name in the File parameter must conform to the naming conventions used for naming tables in Studio or Image Editor. If the Key parameter does not occur within the named table, the return string is empty.

You can include one of these INI options to specify that entry table files will use the old or new format. (*Do not* include both options.)

```
< Tables >
  OldFormatOnly = Yes
  NewFormatOnly = Yes
```

For instance, if you are doing a lot of entry table lookups from the DAL code, your tables are located on a network drive, and the tables are a mix of both old and new format tables, performance can be affected because the system has to check the format of each table.

If, however, you can use one of these new options to tell the system that all tables are in the same format, it can omit that query and performance improves.

Specify *only* the option that applies. If you omit both options, the system first checks to see if the table is in the new format. If not, then it checks to see if the table is in the old format.

Keep in mind that if you include one of these options, all of your tables must be in that format. For instance if you set the OldFormatOnly option to Yes, all of your tables must be in the old format. If you later decide to convert your tables to the new format, you must remove this option and, to get the same performance gain, include the NewFormatOnly option.
Here are some examples:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table (&quot;D&quot;, &quot;GA&quot;, &quot;STATCOD&quot;, &quot;table1&quot;)</td>
<td>Georgia</td>
<td>Verifies that a table named STATCOD is contained in the file named table1. Then returns the description (Georgia) for the key code GA.</td>
</tr>
</tbody>
</table>

See also Documaker Workstation Functions on page 59
**Time**

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

**Syntax**

\[ \text{Time}\ (\text{Format, Hour, Minutes, Seconds}) \]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format</td>
<td>Enter a time format string. The default is time format 1 (HH:MM:SS).</td>
</tr>
<tr>
<td>Hour</td>
<td>Enter a number to indicate the hour. The default is the current hour.</td>
</tr>
<tr>
<td>Minutes</td>
<td>Enter a number to indicate the minute. The default is the current minute.</td>
</tr>
<tr>
<td>Seconds</td>
<td>Enter a number to indicate the second. The default is the current second.</td>
</tr>
</tbody>
</table>

The system returns a time string that contains a formatted time value.

If you omit one of the Hour, Minute, or Seconds parameters, the system uses the appropriate value from the current time.

**Example**

Here are some examples:

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time( )</td>
<td>07:07:32</td>
<td>No parameters entered. It defaults to the current time in format 1.</td>
</tr>
<tr>
<td>Time(2,13,30,5)</td>
<td>01:30:05 PM</td>
<td>Format 2 selected; time displays in 12-hour format using these values.</td>
</tr>
</tbody>
</table>

**See also** [Time Formats on page 80](#)
**Time2Time**

Use this function to convert a time from one format to another.

**Syntax**

```
Time2Time (OldTime, OldFormat, NewFormat)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OldTime</td>
<td>Enter a valid time string. The system assumes your entry is in the time format specified in the OldFormat parameter. The default is the current time.</td>
</tr>
<tr>
<td>OldFormat</td>
<td>Enter a valid time format that describes the OldTime parameter. The default is time format 1 (HH:MM:SS).</td>
</tr>
<tr>
<td>NewFormat</td>
<td>Enter a valid time format that describes the format you want the OldTime converted to. The default is time format 1 (HH:MM:SS).</td>
</tr>
</tbody>
</table>

**Example**

Here is an example:

(Assume $T1$ is 01:30:05 pm.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time2Time(“T1”, “2”, “1”)</td>
<td>13:30:05</td>
<td>Takes the time in $T1$ (which is in format 2) and converts it to format 1.</td>
</tr>
</tbody>
</table>

See also **Time Formats on page 80**
**TIMEADD**

Use this function to add time to a given time and return the new time. The resulting time is returned in the same format.

**Syntax**

\[
\text{TIMEADD (Time, Format, Seconds, Minutes, Hours)}
\]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>Enter a valid time string. The system assumes your entry is in the time format specified in the Format parameter. The default is the current time.</td>
</tr>
<tr>
<td>Format</td>
<td>Enter a valid time format that describes the Time parameter. The default is time format 1 (HH:MM:SS).</td>
</tr>
<tr>
<td>Seconds</td>
<td>Enter the number of seconds to be added. The default is zero (0).</td>
</tr>
<tr>
<td>Minutes</td>
<td>Enter the number of minutes to be added. The default is zero (0).</td>
</tr>
<tr>
<td>Hours</td>
<td>Enter the number of hours to be added. The default is zero (0).</td>
</tr>
</tbody>
</table>

**Example**

Here is an example:

(Assume the current time is 1:20:03 pm.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMEADD( , , &quot;10&quot;, &quot;20&quot;, &quot;3&quot;)</td>
<td>4:40:13</td>
<td>Defaults to the current time and adds 3 hours, 20 minutes, and 10 seconds. Returns the result in the same format.</td>
</tr>
</tbody>
</table>

**See also**  
[Time Formats on page 80](#)
**TimeZone**

Use this function to return the system’s time zone setting or to make sure a time zone is valid.

**Syntax**

```plaintext
TimeZone (TimeZone)
```

**Parameter** | **Description**
---|---
TimeZone (Optional) | If you include a time zone string, the system makes sure that string is valid. If it is invalid, the system returns an empty string. The default is to return the system’s current time zone setting.

**Example**

Here are some examples:

This example returns the system time zone, such as *America/New_York*:

```plaintext
T1 = TimeZone()
```

This example checks to see if a time zone string, such as *Europe/London*, is valid:

```plaintext
T1 = 'Europe/London'
T2 = TimeZone(T1)
if (T2 = '') then
    Print_It(T1 & ' is not a valid time zone string')
else
    Print_It(T1 & ' is a valid time zone string')
end
```

**See also**

- TimeZone2TimeZone on page 410
- Time Functions on page 80
- Using the Time Zone Functions on page 81
- ICU Time Zones on page 82
**TimeZone2TimeZone**

Use this function to convert date and time values from one geographic region into date and time values that are local to another geographic region. The function will also adjust for daylight savings time as needed.

**Syntax**

```
TimeZone2TimeZone (PrefixName, TimeZone, NewTimeZone)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrefixName</td>
<td>Enter the prefix name associated with variables that will be used to hold date and time settings. Here are some examples:</td>
</tr>
<tr>
<td></td>
<td>PrefixName.day</td>
</tr>
<tr>
<td></td>
<td>PrefixName.month</td>
</tr>
<tr>
<td></td>
<td>PrefixName.year</td>
</tr>
<tr>
<td></td>
<td>PrefixName.hour</td>
</tr>
<tr>
<td></td>
<td>PrefixName.minutes</td>
</tr>
<tr>
<td></td>
<td>PrefixName.seconds</td>
</tr>
<tr>
<td>TimeZone</td>
<td>(Optional) Enter the time zone used for the PrefixName variables.</td>
</tr>
<tr>
<td></td>
<td>If you enter an invalid time zone string, the system returns a value of zero (0) and sets variables associated with the PrefixName to zero (0).</td>
</tr>
<tr>
<td></td>
<td>The default is to return the system’s current time zone setting.</td>
</tr>
<tr>
<td>NewTimeZone</td>
<td>(Optional) Enter the time zone by which you want to adjust the values in the PrefixName variables.</td>
</tr>
<tr>
<td></td>
<td>If you enter an invalid time zone string, the system returns a value of zero (0) and sets variables associated with the PrefixName to zero (0).</td>
</tr>
<tr>
<td></td>
<td>The default is to return the system’s current time zone setting.</td>
</tr>
</tbody>
</table>

If you define these variables, the system uses the PrefixName and time you specified and converts that time to the equivalent time in the location you specified via the NewTimeZone parameter.

If you do not define these variables, the system creates these variables based on the PrefixName you entered and assigns values into these variables based on the current date and time.

If there are no errors, the system returns a non-zero value.

**Example**

Here are some examples:

This example creates date and time variables using `tz` as a prefix (`tz.day, tz.month, tz.year, tz.hour, tz.minute, tz.second`) and stores the current date and time values based on the system’s time zone:

```vbnet
TimeZone2TimeZone('tz', ,)
Print_It('Date:' & Date(, tz.day, tz.month, tz.year))
Print_It('Time:' & Time(, tz.hour, tz.minute, tz.second))
```

This example converts date and time variables (`tz.xxxx`) that use the system’s time zone into GMT date and time:

```vbnet
TimeZone2TimeZone('tz', , 'GMT')
Print_It('GMT Date:' & Date(, tz.day, tz.month, tz.year))
Print_It('GMT Time:' & Time(, tz.hour, tz.minute, tz.second))
```
This example converts a current America/New_York date and time into an Australia/Melbourne date and time:

```powershell
tz.day = ''
tz.month = ''
tz.year = ''
tz.hour = ''
tz.minute = ''
tz.second = ''
if (TimeZone2TimeZone('tz', 'America/New_York', 'Australia/Melbourne')) then
    Print_It('Australia/Melbourne Date:' & Date(, tz.day, tz.month, tz.year))
    Print_It('Australia/Melbourne Time:' & Time(, tz.hour, tz.minute, tz.second))
else
    Print_It('Error calling TimeZone2TimeZone')
end
```

See also
- TimeZone on page 409
- Time Functions on page 80
- Using the Time Zone Functions on page 81
- ICU Time Zones on page 82
**TOTALPAGES**

Use this function to return the number of pages that will print for a given recipient or for all recipients. A page is considered any side of paper that has a printable section for a recipient. A duplex sheet with front and back sections counts as two pages.

**Syntax**

```
TotalPages (Recipient)
```

**Parameter** | **Description**
---|---
Recipient | (Optional) If you include the Recipient parameter, the count only reflects the pages that print for that recipient. If you omit the Recipient parameter, the count includes all recipients.

The count considers copy-counts and reflects the total number of printed sides that will be referenced. A section may be empty (containing no text or discernible print objects) and still be designated to print. So, the count does not necessarily mean the pages will contain any real text.

**Example**

For example, assume you have a one-page document that has two recipients. Recipient1 gets one copy, while Recipient2 gets two copies.

With this command:

```
TotalPages("Recipient1")
```

The system returns one (1) as the page count

With this command:

```
TotalPages("Recipient2")
```

The system returns two (2) as the page count, since the one-page document will be printed twice. If you omit the Recipient parameter, the system returns three (3) as the page count.

**NOTE:** The count reflects when the function is called. The system cannot predict whether banner pages will be created or whether additional formatting or data entry will add or remove pages. Make sure you do not call this function until all page items have been created and formatted.

**See also**

TotalSheets on page 413

Documaker Workstation Functions on page 59
**TOTALSHEETS**

Use this function to return the total number of sheets of paper that will print for a recipient. A sheet is considered a physical piece of paper that may have print on one or both sides. Therefore a duplex sheet with a front and back sections will count as one sheet.

**NOTE:** Although the TotalSheets function does take duplex options into consideration, it has no knowledge of whether you will actually print to a printer that supports duplex commands. The count reflects what the document defines, not what the printer will support.

### Syntax

```
TotalSheets (Recipient)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recipient</td>
<td>(Optional) If you include the Recipient parameter, the count only reflects the sheets that print for that recipient. If you omit the Recipient parameter, the count includes all recipients.</td>
</tr>
</tbody>
</table>

The count takes into consideration recipient copy counts and duplex options. A section may be empty (containing no text or discernible print objects) and still be designated to print. So, the count does not necessarily mean that the sheets will contain any real text.

### Example

For example, assume you have a two-page document that is duplexed (prints front and back). Recipient1 gets one copy, while Recipient2 gets two copies.

With this command:

```
TotalSheets(Recipient1)
```

The system returns one (1) as the sheet count.

With this command:

```
TotalSheets(Recipient2)
```

The system returns two (2) as the sheet count, since the two-page document will be printed twice. if you omit the Recipient parameter, the system returns three (3) as the sheet count.

**NOTE:** The count reflects when the function is called. The system cannot predict whether banner pages will be created or whether additional formatting or data entry will add or remove pages. Make sure you do not call this function until all page items have been created and formatted.

### See also

- TotalPages on page 412
- Documaker Workstation Functions on page 59
**TriggerForm**

Use this function/procedure to add a form to your form set and still have the form’s section triggers evaluated.

For instance, you can use this function when users add forms to a policy and want endorsements to print in a sequence unique to each transaction. The RunTriggers rule evaluates each form level trigger and adds the form to the set using the order specified by the forms list in the MRL.

If you use an import file rule, the system adds the forms in the order specified by the associated input. If you have a situation where you want to control the form addition, but the data is not an import style file, you may want to use this function to do the form additions.

This function adds a form to the form set and evaluates the form’s section triggers. Section trigger and recipient processing are then performed as if the form had been triggered by the RunTriggers rule.

For each triggered form, the form’s section triggers are evaluated. If a section trigger returns a zero (0) count, the system removes the section from the form. If the section trigger returns a multiple count, the system inserts additional sections into the form until the section count matches the trigger count.

If the TriggerForm function is executed before the standard section processing rule (RULStandardImageProc), the standard field rules are also be executed on the inserted sections.

Here’s an example. Assume these entries are in your AFGJOB.JDT file:

```
;PreTransDAL;;TriggerForm("Myltr1");
;RunTriggers;;
;PreTransDAL;;TriggerForm("Myltr2");
;PreTransDAL;;TriggerForm("Myltr1","Myltr3");
;PreTransDAL;;TriggerForm("Myltr4",,"LB2");
```

<table>
<thead>
<tr>
<th>This form</th>
<th>Is added</th>
</tr>
</thead>
<tbody>
<tr>
<td>Myltr1</td>
<td>To the beginning of the form set (before any forms triggered by the RunTriggers rule)</td>
</tr>
<tr>
<td>Myltr2</td>
<td>After any forms triggered by the RunTriggers rule</td>
</tr>
<tr>
<td>Myltr3</td>
<td>After the first occurrence of Myltr1</td>
</tr>
<tr>
<td>Myltr4</td>
<td>To the group named LB2</td>
</tr>
</tbody>
</table>

**NOTE:** You can also use the AddForm procedure to add a form to a form set. The difference is TriggerForm also tells the system to evaluate the form’s section triggers, where AddForm does not.

**Syntax**

```
TriggerForm (Form, Insert, Group)
```
The system optionally returns one (1) on success or zero (0) on failure.

This procedure adds a copy of the specified form to the document set. The form named must be a valid form in the specified group. You cannot add a form defined for one group into another group. You can specify the name of the form you want to insert using the occurrence indicator.

If you include the Group parameter, make sure it references a group defined in the Application Definition (BDF) file.

Use this procedure with resources created using Documaker Studio and processed via the GenData program. If you use this procedure outside of GenData, such as with GenPrint or the entry system, or if you use it with legacy resources, the TriggerForm function will not cause the system to evaluate section triggers. In that scenario, you simply get the functionality of the AddForm procedure.

### Example

Here are some examples:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form</td>
<td>Enter the name of a form in the specified group.</td>
</tr>
<tr>
<td>Insert</td>
<td>Enter the name of a form after which the new form should be inserted. The default is to append after the last form in the group.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of a group to contain the specified form. The default is the current group.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>TriggerForm(&quot;FormA&quot;)</td>
<td>1 or 0</td>
<td>Add FormA after the last form in the current group and evaluate FormA's section triggers.</td>
</tr>
<tr>
<td>TriggerForm(&quot;FormA&quot;, &quot;FormA\1&quot;, GRP)</td>
<td>1 or 0</td>
<td>Add FormA after the first occurrence of that form within the named group and evaluate FormA's section triggers.</td>
</tr>
</tbody>
</table>

### See also

AddForm on page 119
AddForm_Propagate on page 120
CopyForm on page 174
DupForm on page 228
WIP Functions on page 88
**TriggerFormName**

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

**Syntax**

```
TriggerFormName ( )
```

There are no parameters for this function.

**Example**

Here is an example:

Assume your SETRECIPTB.DAT file has the following entries and a loaded DAL library file contains the DAL sub-routine function, *ILDScrk*. The forms are triggered if the conditions in the DAL script are met.

Here is an example of the SETRECIPTB.DAT file:

```plaintext
... ;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:

```plaintext
*** If driver's age, insured state, and form name are the specified conditions then trigger the form.

BeginSub ILDSchk

  trig_f_name  = TriggerFormName()
  If trig_f_name = "ILDS498" AND \
    ?("driver_age") <= 25 AND \
    ?("insure_st")  = "CA" Then
    Return(1)
  ElseIf trig_f_name = "ILDS598" AND \
    ?("driver_age") > 25 \
    ?("insure_st")  = "FL" Then
    Return(1)
  Else
    Return(0)
  End

EndSub
```

See also

- TriggerImageName on page 417
- TriggerRecsPerOvFlw on page 418
- Documaker Server Functions on page 58
**TriggerImageName**

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

**Syntax**

```
TriggerImageName ( )
```

There are no parameters for the function.

**Example**

Here is an example:

Assume your SETRECIPTB.DAT file has the following entries and a loaded DAL library file contains the DAL sub-routine function, *ILDSChk*. The forms are triggered if the conditions in the DAL script are met.

Here is an example of the SETRECIPTB.DAT file:

```
...;
Docu;CP;ILDS498;S004H;XLC;Agent(1);;0;0;0;1;;DALTrigger;ILDSChk;
;Docu;CP;ILDS598;S004L;XLC;Agent(1);;0;0;0;1;;DALTrigger;ILDSChk;
...
```

Here is an example of the DAL library file:

````
*** If driver's age, insured state, and section name are the
*** specified
*** conditions then trigger the section.

BeginSub ILDSChk

trig_f_name  = TriggerImageName()
If trig_f_name = "S004H" AND \
   ?("driver_age") <= 25 AND \n   ?("insure_st")  = "CA" Then
   Return(1)
ElseIf trig_f_name = "S00L" AND \
   ?("driver_age") > 25 \n   ?("insure_st")  = "FL" Then
   Return(1)
Else
   Return(0)
End

EndSub
```

See also  

- TriggerFormName on page 416
- TriggerRecsPerOvFlw on page 418
- Documaker Server Functions on page 58
**TRIGGERRECSPEROvFLW**

Use this function to retrieve the number of records per overflow section value which is stored in the SETRCPTBL.DAT entry being processed. Depending on the current trigger, this integer value can be the overflow record count for a form or section.

**NOTE:** This is only applicable in Documaker Server processing during DAL trigger processing.

**Syntax**

```plaintext
TriggerRecsPerOvFlw ( )
```

There are no parameters for this function.

**Example**

Assume you have the following entry in the SETRCPTBL.DAT file for the form trigger being processed. Also assume there are 30 records in the extract file that match the search mask.

```
;RP10;CIS;qa_f1550;;;Customer(1);1,M;25;0;1;;DALTrigger;FEATURE1550;
```

Here is an example:

```plaintext
BeginSub Feature1550
    #rec = CountRec("1,Feature1550,31,Data")
    #remaining = MOD(#rec, TriggerRecsPerOvFlw( ))
    While(#remaining > 0)
        * write addition records
            Write_fm( )
        #mod -= 1
    Wend
    Return(#rec)
EndSub
```

In this example, the TriggerRecsPerOvFlw function, returns a records per overflow section value of 25, which is used in the MOD function.

**See also**

- MOD on page 320
- TriggerFormName on page 416
- TriggerImageName on page 417
- Documaker Server Functions on page 58
**TRIM**

Use this function to remove leading and/or trailing spaces from a given string. The integer parameter determines whether spaces on the left, right, or both ends are to be removed. The resulting string is returned.

**Syntax**

```
Trim (String, Integer)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Enter a valid string. The default is the value of the current field.</td>
</tr>
<tr>
<td>Integer</td>
<td>Choose from these options: 0 - remove trailing spaces 1 - remove leading spaces 2 - remove leading and trailing spaces The default is two (2).</td>
</tr>
</tbody>
</table>

The system removes leading and trailing spaces from the string specified in parameter one. The Integer parameter determines which spaces are removed.

**Example**

Here are some examples:

(Assume the current field contains the text “    Your Name”)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trim (“ Value “)</td>
<td>“Value”</td>
<td>Defaults to trim leading and trailing spaces.</td>
</tr>
<tr>
<td>Trim (“ Value “, 0)</td>
<td>“ Value”</td>
<td>Removes trailing spaces.</td>
</tr>
<tr>
<td>Trim( )</td>
<td>“Your Name”</td>
<td>Use current field and remove leading and trailing spaces. See the note below.</td>
</tr>
</tbody>
</table>

**NOTE:** During field entry, the system automatically removes trailing spaces from values entered by the user. Only variables assigned during DAL scripts are likely to have trailing spaces.

**See also** String Functions on page 78
**Upper**

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

**Syntax**

`Upper (String, Length)`

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Enter a valid string. The default is the value of the current field.</td>
</tr>
<tr>
<td>Length</td>
<td>Enter the length of the output. The default is the length of the current field.</td>
</tr>
</tbody>
</table>

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*.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Upper ()</code></td>
<td>“YOUR NAME”</td>
<td>Defaults to the current field.</td>
</tr>
<tr>
<td><code>Upper (, 15)</code></td>
<td>“YOUR NAME”</td>
<td>Defaults to the current field and increases the length of the field to 15.</td>
</tr>
<tr>
<td><code>Upper (&quot;Street Address&quot;)</code></td>
<td>“STREET ADDRESS”</td>
<td>Uppercases the specified string.</td>
</tr>
</tbody>
</table>

**See also**

- String Functions on page 78
- Lower on page 305
**UNIQUESTRING**

Use this function to return a 45-character globally unique string.

**Syntax**

```
UniqueString() 
```

There are no parameters for this function.

Keep in mind...

- These print drivers are supported: PCL5, PCL6, PST, MET, AFP, PDF, HTML, and RTF.
- These print drivers are not supported: EPT, MDR, and GDI.
- All platforms are supported, but note that while UniqueString is supported on z/OS, z/OS does not support PDF or long file names, so the PDF example does not apply to z/OS.
- Both multi- and single-step processing are supported.

The only DAL function actually involved in splitting the print stream is BreakBatch. The others make it easier to implement this functionality. For example, since you need to name the new print stream, you use the SetDeviceName procedure. To find the name of the current device, you use the DeviceName function. If you need to create unique file names, you can use the UniqueString function.

**NOTE:** While you can call all of these DAL functions in Documaker Server or Entry, the BreakBatch and SetDeviceName functions are not applicable in Entry since it does not use the batch printing engine. DeviceName and UniqueString are applicable to both Entry and Documaker Server.

**Example**

Here is an example:

```
DataPath = GetINIString(,"Data","DataPath")
Drive = FileDrive(DataPath)
Path = FilePath(DataPath)
UniqueID = UniqueString()
Outputname = FullFileName(Drive,Path,UniqueID,".PDF")
SetDeviceName(Outputname)
```

See also

- Miscellaneous Functions on page 73
- BreakBatch on page 158
- DeviceName on page 217
- SetDeviceName on page 373
**UserID**

Use this function to return the user ID used to log on to the Entry module.

**Syntax**

```
UserID ( )
```

There are no parameters for this function.

This function is only useful if the system is set up to require user IDs.

**Example**

Here are some examples:

(Assume the current user is TOMJ.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>result = UserID( )</td>
<td>TOMJ</td>
<td>Identifies the current user ID as TOMJ.</td>
</tr>
<tr>
<td>SetFld (UserID ( ), ”MyField”)</td>
<td>TOMJ</td>
<td>First UserID determines that the current user ID is TOMJ, then the field named <em>MyField</em> is assigned the value TOMJ by the SetFld procedure.</td>
</tr>
</tbody>
</table>

**See also**

- WIP Functions on page 88
- UserLvl on page 423
**UserLvl**

Use this function to get the currently logged in user's access rights level. The value returned is in the range 0-9. Zero represents the highest level and nine represents the lowest level. Access rights levels are specific to each system implementation.

**Syntax**

```
UserLvl ( )
```

There are no parameters for this function.

This function is only useful if the system is set up to require user IDs and user rights.

**Example**

Here is an example:

(Assume the current user is TOMJ with an access rights level of 7.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>#result=UserLvl ( )</td>
<td>7</td>
<td>Determines that TOMJ's user rights are 7 and returns a 7.</td>
</tr>
<tr>
<td>IF (UserLvl( ) !=0) MSG(USERID( ), &quot;Remember to get a supervisor to approve this transaction.&quot;); END;</td>
<td>TOMJ</td>
<td>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.</td>
</tr>
</tbody>
</table>

See also

- WIP Functions on page 88
- UserID on page 422
**WeekDay**

Use this function to determine the day of the week in a given date and return the value as a number.

**Syntax**

```
WeekDay (Date, Format, Locale)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>Enter a valid date string. The system assumes your entry is in the format specified by the Format parameter. The default is the current date.</td>
</tr>
<tr>
<td>Format</td>
<td>Enter a valid date format that describes the format used by your entry in the Date parameter. The default is date format 1 (MM/DD/YY).</td>
</tr>
<tr>
<td>Locale</td>
<td>(Optional) Enter the locale code. If you omit this parameter, the system checks the Locale INI option. If the Locale INI option offers no value, the system defaults to USD (United States/English).</td>
</tr>
</tbody>
</table>

The system returns the number of the day of the week, from 1 to 7, as shown here:

<table>
<thead>
<tr>
<th>Number</th>
<th>Day of the week</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sunday</td>
</tr>
<tr>
<td>2</td>
<td>Monday</td>
</tr>
<tr>
<td>3</td>
<td>Tuesday</td>
</tr>
<tr>
<td>4</td>
<td>Wednesday</td>
</tr>
<tr>
<td>5</td>
<td>Thursday</td>
</tr>
<tr>
<td>6</td>
<td>Friday</td>
</tr>
<tr>
<td>7</td>
<td>Saturday</td>
</tr>
</tbody>
</table>

WeekDay is most often used with the DayName function. The DayName function extracts the name of the day of the week from a given date.

**Example**

Here are some examples:

(Assume the current date is Wednesday, July 5, 2009.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>WeekDay ()</td>
<td>4</td>
<td>Defaults to the current date.</td>
</tr>
<tr>
<td>Datestring = DateAdd(,, 1); WeekDay(datestring)</td>
<td>5</td>
<td>First the DateAdd function adds one day to the current date, resulting in a date of Thursday, July 6, 2009. Then WeekDay returns 5, which corresponds to Thursday.</td>
</tr>
</tbody>
</table>

**See also**

Date Functions on page 51
Locales on page 55
Using INI Options on page 8
Date Formats on page 52
DateAdd on page 184
DayName on page 189
**WHATFORM**

Use this function to return the name of the form that includes the item you searched for. Having the name of the form lets you manipulate that object using other DAL functions, which may require its name.

### Syntax

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>Enter the name of the field. The default is the current field.</td>
</tr>
<tr>
<td>Section</td>
<td>Enter the name of the section. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of the form. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of a group to contain the specified form. The default is the current group.</td>
</tr>
</tbody>
</table>

If nothing matches your criteria, the system returns a blank.

Keep in mind you can use an asterisk (*) as the object name to match parent objects. This lets you find objects without explicitly knowing the parent names.

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

### Example

Here is an example:

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>form = WhatForm(&quot;Total Field\d&quot;, , , &quot;*&quot;);</code></td>
<td>The name of the form or 0</td>
<td>Attempts to locate the third occurrence of a field in a form set and returns the name of the form that contains that field.</td>
</tr>
</tbody>
</table>

### See also

- WhatGroup on page 427
- WhatImage on page 428
- Name Functions on page 74
Use this function to return the name of the group that includes the item you searched for. Having the name of the form lets you manipulate that object using other DAL functions, which may require its name.

**Syntax**

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>Enter the name of the field. The default is the current field.</td>
</tr>
<tr>
<td>Section</td>
<td>Enter the name of the section. The default is the current section.</td>
</tr>
<tr>
<td>Form</td>
<td>Enter the name of the form. The default is the current form.</td>
</tr>
<tr>
<td>Group</td>
<td>Enter the name of a group to contain the specified form. The default is the current group.</td>
</tr>
</tbody>
</table>

If nothing matches your criteria, the system returns a blank.

Keep in mind you can use an asterisk (*) as the object name to match parent objects. This lets you find objects without explicitly knowing the parent names.

**NOTE:** For optimal performance, minimize the use of wildcards (*) when searching for field, section, or form names.

**Example**

Here is an example:

```
group = WhatGroup( , , "MyForm", ");
The name of the form or 0 Attempts to locate the group name that contains a specific form.
```

**See also**

- WhatForm on page 426
- WhatImage on page 428
- Name Functions on page 74
**WHATIMAGE**

Use this function to return the name of the section that includes the item you searched for. Having the name of the form lets you manipulate that object using other DAL functions, which may require its name.

**Syntax**

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

**Parameter** | **Description**
--- | ---
Field | Enter the name of the field. The default is the current field.
Section | Enter the name of the section. The default is the current section.
Form | Enter the name of the form. The default is the current form.
Group | Enter the name of a group to contain the specified form. The default is the current group.

If nothing matches your criteria, the system returns a blank.

Keep in mind you can use an asterisk (*) as the object name to match parent objects. This lets you find objects without explicitly knowing the parent names.

**NOTE:** For optimal performance, minimize the use of wildcards (*) when searching for field, section, or form names.

**Example**

Here is an example:

```
section = WhatSection("Total Field\12", "\0");
```

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>section = WhatSection(&quot;Total Field\12&quot;, &quot;\0&quot;);</td>
<td>The name of the form or 0</td>
<td>Attempts to locate the twelfth occurrence of a field in a form set and returns the name of the section that contains that field.</td>
</tr>
</tbody>
</table>

**See also**

- WhatForm on page 426
- WhatGroup on page 427
- Name Functions on page 74
**WIPExit**

Use this procedure/function to close work-in-process.

**Syntax**

```
WIPExit(SaveFlag)
```

**Parameter Description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SaveFlag</td>
<td>Enter a positive number, such as one (1), to save and close the current form set. Enter zero (0) to close the form set without saving and exit WIP. The default is to save and close the current form set.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIPExit(1)</td>
<td>Exits WIP and saves your work.</td>
<td>Work is saved with a valid positive flag.</td>
</tr>
<tr>
<td>WIPExit(0)</td>
<td>Exits WIP but does not save your work.</td>
<td>Work is not saved with a flag of zero.</td>
</tr>
</tbody>
</table>

**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

\[
\text{WIPFld} \ (\text{WIPfield})
\]

Parameter | Description
--- | ---
WIPfield | Enter the name of the field in the WIP record.

The system returns the value of an identified field within the current WIP record. WIP records are only defined within the Entry system and are implementation specific. If a request is made for a field that is not part of the WIP record definition, the system returns an empty string.

Example

Here are some examples:

(Assume the current WIP record has a field named OrigUser which contains the string David Harris.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>result = WIPFld (&quot;OrigUser&quot;)</td>
<td>David Harris</td>
<td>Determines that the current WIP record named OrigUser has the value David Harris and returns that value.</td>
</tr>
<tr>
<td>IF (WIPFld ('StatusCode') != 'W')</td>
<td>SetFld(&quot;N/A&quot;); END</td>
<td>If the current WIP record does not contain a StatusCode field that is equal to W, the SetFld statement executes.</td>
</tr>
</tbody>
</table>

See also WIP Functions on page 88
WIPKey1

Use this function to return the value of the Key1 field from the current WIP record.

Syntax

WIPKey1 ( )

There are no parameters for this function.

The system returns the value of the Key1 field within the current WIP record known as the Company field in the insurance market. WIP records are only defined within the Entry module and are specific for each implementation.

This is a short-cut method for WIPFld("KEY1"), which would return the same value.

Example

Here are some examples:

(Assume the current WIP record contains a Key1 field with the value Oracle.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>result = WIPKey1( )</td>
<td>Oracle</td>
<td>Determines the value contained in the WIP Key1 field and returns that value.</td>
</tr>
<tr>
<td>IF WIPKey1( ) = &quot;Oracle&quot; THEN SetFld(&quot;N/A&quot;); END</td>
<td>1 N/A</td>
<td>Determines that the Key1 field contains the value Oracle, then executes the SetFld procedure and places N/A in the current field. Also returns one (1) to indicate that the SetFld procedure was successful.</td>
</tr>
</tbody>
</table>

See also

WIP Functions on page 88
WIPFld on page 430
WIPKey2 on page 432
WIPKeyID on page 433
SetFld on page 377
WIPKey2

Use this function to return the value of the Key2 field from the current WIP record.

Syntax

\[ \text{WIPKey2}() \]

There are no parameters for this function.

The system returns the description of the Key2 field in the current WIP record, known as the Line of Business field in the insurance market. WIP records are only defined within the entry system and are implementation specific.

This is a short-cut method for WIPFld(“KEY2”), which would return the same value.

Example

Here are some examples:

(Assume the current WIP record contains a Key2 field with the value “Fire Insurance”.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{result = WIPKey2()}</td>
<td>Fire Insurance</td>
<td>Determines the value contained in the WIP Key2 field and returns that value.</td>
</tr>
</tbody>
</table>
| IF WIPKey2() = “Oracle” \;
  SetFld(“N/A”); \;
  END | Nothing | Determines that the Key2 field does not contain the value “Oracle”; therefore the SetFld procedure does not execute. |

See also

WIP Functions on page 88
WIPFld on page 430
WIPKey1 on page 431
WIPKeyID on page 433
SetFld on page 377
**WIPKeyID**

Use this function to replace the value of the KeyID field from the current WIP record.

**Syntax**

```plaintext
WIPKeyID( )
```

There are no parameters for this function.

The system returns the value of the KeyID field in the current WIP record, known as the *Policy Number* field in the insurance market. WIP records are only defined in the Documaker and are implementation specific.

This is a short-cut method for the WIPFld("KEYID") function, which would return the same value.

**Example**

Here are some examples:

(Assume the current WIP record contains a KeyID field with the value “1300”.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>result = WIPKeyID( )</code></td>
<td>1300</td>
<td>Determines the value contained in the WIP KeyID field and returns that value.</td>
</tr>
<tr>
<td><code>IF LEFT(WIPKeyID( ), 3) &gt; 100 SETFLD(&quot;N/A&quot;); END</code></td>
<td>1 N/A</td>
<td>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.</td>
</tr>
</tbody>
</table>

**See also**

- WIP Functions on page 88
- WIPFld on page 430
- WIPKey1 on page 431
- WIPKey2 on page 432
- SetFld on page 377
**XMLAttrName**

Use this function to return the name of the current attribute pointed to by the XMLFirstAttrib and XMLNextAttrib functions.

**Syntax**

```
XMLAttrName (%XMLTree)
```

**Parameter** | **Description**
--- | ---
%XMLTree | Enter a list type DAL variable that passes the XML tree handle.

The system returns the name of the current attribute pointed to by the XMLFirstAttrib and XMLNextAttrib functions.

**Example**

This example returns the second attribute name of the first form in the list.

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

**See also**

- XML Functions on page 89
- XMLFirstAttrib on page 438
- XMLNextAttrib on page 443
XMLAttrValue

Use this function to return the value of the current attribute pointed to by the XMLFirstAttrib and XMLNextAttrib functions. This function is similar to the XMLAttrName function.

Syntax

```plaintext
XMLAttrValue (%XMLTree)
```

**Parameter** | **Description**
---|---
%XMLTree | Enter a list type DAL variable that passes the XML tree handle.

The system returns the value of the current attribute pointed to by the XMLFirstAttrib and XMLNextAttrib functions.

**Example**

This example returns the second attribute name of the first form in the list.

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

**See also**

- XML AttrName on page 434
- XML FirstAttrib on page 438
- XML NextAttrib on page 443
Use this function to locate the XML path from the extracted XML tree and return a list of matched elements to either a:

- List type DAL variable, or a
- Matched text to a string type DAL variable

The result depends on the search request.

**Syntax**

```
XMLFind (%xXMLTree, SrchNode, XPath)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%xXMLTree</td>
<td>A list type DAL variable which is passed from either the XMLFileExtract rule or the LoadXMLList function. You can use the predefined %extract variable as a parameter here, as discussed in scenario 1.</td>
</tr>
<tr>
<td>SrchNode</td>
<td>A string type DAL variable that passes a node name from which the search starts. If you omit this parameter, the search starts from the root of the XML tree.</td>
</tr>
<tr>
<td>XPath</td>
<td>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.</td>
</tr>
</tbody>
</table>

The system returns a list type or a string type DAL variable.

**Example**

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

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

**See also**

- XML Functions on page 89
- Scenario 1 on page 90
- Scenario 2 on page 90
XMLFirst

Use this function to set the current pointer to the first element in the specified list.

**Syntax**

```plaintext
XMLFirst (%XMLTree)
```

**Parameter** | **Description**
--- | ---
%XMLTree | A list type DAL variable. This variable can be either an XML tree or a list of extracted elements.

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

**Example**

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

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

See also  | XML Functions on page 89
XMLFirstAttrib

Use this function to set the current pointer to the first element in the list you specify.

Syntax

```xml
XMLFirstAttrib (%XMLTree)
```

Parameter | Description
--- | ---
%XMLTree | Enter a list type DAL variable. You can enter either an XML tree or a list of extracted elements.

This function sets the attribute pointer to the first attribute for the current element in the element list or to the first attribute element in the attribute list.

If you input an element list, use these functions to retrieve the attribute name and value:
- XMLAttribName
- XMLAttribValue

If you input an attribute list, use these functions to retrieve attribute name and value:
- XMLNthAttribName
- XMLNthAttribValue

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

Example

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

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

See also
- XML Functions on page 89
- XMLAttribName on page 434
- XMLAttribValue on page 435
- XMLNthAttribName on page 445
- XMLNthAttribValue on page 446
XMLFirstText

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

Syntax

XMLFirstText (List)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>List</td>
<td>Enter the name of the list.</td>
</tr>
</tbody>
</table>

Example

Here is an example:

Mystring = XMLFirstText(List)

See also

XML Functions on page 89
**XMLGetCurName**

Use this function to get the name from the current element. This function is similar to the XMLGetCurText function.

### Syntax

```
XMLGetCurName (%XMLTree)
```

### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%XMLTree</td>
<td>A list type DAL variable. This variable can be either an XML tree or a list of extracted elements.</td>
</tr>
</tbody>
</table>

The system returns the element name from the current element. The return value is a string type DAL variable.

### Example

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

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

### See also

- XML Functions on page 89
- XMLFirst on page 437
- XMLGetCurText on page 441
**XMLGetCurText**

Use this function to get the text from the current element. This function is similar to the XMLGetCurName function.

**Syntax**

```
XMLGetCurText (%XMLTree)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%XMLTree</td>
<td>A list type DAL variable. This variable can be either an XML tree or a list of extracted elements.</td>
</tr>
</tbody>
</table>

The system returns the text from the current element. The return value is a string type DAL variable.

**Example**

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

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

**See also**

- [XML Functions on page 89](#)
- [XMLFirst on page 437](#)
- [XMLGetCurName on page 440](#)
**XMLNext**

Use this function to set the current pointer to the next node or element in the specified list. This function is similar to the XMLFirst function.

**Syntax**

```
XMLNext (%XMLTree)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%XMLTree</td>
<td>A list type DAL variable. This variable can be either an XML tree or a list of extracted elements.</td>
</tr>
</tbody>
</table>

The system sets the current pointer to the next node or element in the list you specified list and returns one (1) for success or zero (0) for failure.

**Example**

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

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

See also  
XML Functions on page 89  
XMLFirst on page 437
## XMLNextAttrib

Use this function to set the current pointer to the next element in the list you specify. This function is similar to the XMLFirstAttrib function.

### Syntax

```
XMLNextAttrib (%XMLTree)
```

### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%XMLTree</td>
<td>A list type DAL variable. This variable can be either an XML tree or a list of extracted elements.</td>
</tr>
</tbody>
</table>

This function sets the current attribute pointer to the next attribute for the current element in the list or to the next attribute element in the attribute list.

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

- XMLAttrName
- XMLAttrValue

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

- XMLNthAttrName
- XMLNthAttrValue

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

### Example

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

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

### See also

- XML Functions on page 89
- XMLAttrName on page 434
- XMLAttrValue on page 435
- XMLNthAttrName on page 445
- XMLNthAttrValue on page 446
**XMLNextText**

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

**Syntax**

```
XMLNextText (List)
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>List</td>
<td>Enter the name of the list.</td>
</tr>
</tbody>
</table>

**Example**

Here is an example:

```
MyString = XMLNextText(List);
```

**See also**

XML Functions on page 89
**XMLNthAttrName**

Use this function to return the nth attribute name, as indicated by an index number you specify.

**Syntax**

\[
\text{XMLNthAttrName} \ (\%\text{XMLTree}, \#\text{Index})
\]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%XMLTree</td>
<td>A list type DAL variable that passes a name list.</td>
</tr>
<tr>
<td>#Index</td>
<td>A integer type DAL variable that passes an index number.</td>
</tr>
</tbody>
</table>

The system returns the nth attribute name indicated by the index number.

**Example**

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

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

**See also**

XML Functions on page 89
XMLFind on page 436
**XMLNthAttrValue**

Use this function to return the nth attribute value, as indicated by an index number you specify. This function is similar to the XMLNthAttrName function.

**Syntax**

```
XMLNthAttrValue (%XMLTree, #Index)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%XMLTree</td>
<td>A list type DAL variable that passes a name list.</td>
</tr>
<tr>
<td>#Index</td>
<td>A integer type DAL variable that passes an index number.</td>
</tr>
</tbody>
</table>

The system returns the nth attribute value indicated by the index number.

**Example**

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

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

**See also**

- XML Functions on page 89
- XMLNthAttrName on page 445
Use this function to return the nth text value, as indicated by an index number you specify.

**Syntax**

```
XMLNthText (%XMLTree, #Index)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%XMLTree</td>
<td>A list type DAL variable that passes a name list.</td>
</tr>
<tr>
<td>#Index</td>
<td>A integer type DAL variable that passes an index number.</td>
</tr>
</tbody>
</table>

The system returns the nth text value indicated by the index number.

**Example**

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

```plaintext
ASTr="Text not found";
%xXMLTree=LoadXMLList("test.xml");
%XMLTree=XMLFind(%xXMLTree,"Forms","Form/text()");  
aStr=XMLNthText (%XMLTree, 1);  
#rt=DestroyList (%xXMLTree);
return(aStr);  
```

**See also**

XML Functions on page 89
LoadXMLList on page 302
**YEAR**

Use this function to determine the number of the year in a given date and returns the value as a four-digit number.

**Syntax**

```
Year (Date, Format, Locale)
```

**Parameter** | **Description**
--- | ---
Date | Enter a valid date string. The system assumes your entry is in the date format specified by the Format parameter. The default is the current date.
Format | Enter a valid date format that describes your entry in the Date parameter. The default is date format 1, (MM/DD/YY).
Locale | (Optional) Enter the locale code. If you omit this parameter, the system checks the Locale INI option. If the Locale INI option offers no value, the system defaults to USD (United States/English).

The system determines the year portion of the given date based on the format you specified in the Format parameter.

**Example**

Here are some examples:

(Assume the current date is 07/01/09.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year ()</td>
<td>2009</td>
<td>Defaults to the current date and returns a four-digit year.</td>
</tr>
<tr>
<td>Year (“2-5-09”, “1-2”)</td>
<td>2009</td>
<td>Returns a four-digit year for the given date.</td>
</tr>
</tbody>
</table>

**See also**

- Date Functions on page 51
- Locales on page 55
- Using INI Options on page 8
- Date Formats on page 52
- YearDay on page 449
**YearDay**

Use this function to determine the number of days from the beginning of the year (counting consecutively from January 1) to a given date and return the value as a number.

**Syntax**

```
YearDay (Date, Format, Locale)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>Enter a valid date string. The system assumes your entry is in the date format specified by the Format parameter. The default is the current date.</td>
</tr>
<tr>
<td>Format</td>
<td>Enter a valid date format that describes your entry in the Date parameter. The default is date format 1, (MM/DD/YY).</td>
</tr>
<tr>
<td>Locale</td>
<td>(Optional) Enter the locale code. If you omit this parameter, the system checks the Locale INI option. If the Locale INI option offers no value, the system defaults to USD (United States/English).</td>
</tr>
</tbody>
</table>

The system determines the day of the year portion of the given date based on the format you specified in the Format parameter.

**Example**

Here are some examples:

(Assume the current date is 07/01/09.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>YearDay()</td>
<td>182</td>
<td>Defaults to the current date and returns the day of the year (counting consecutively from January 1).</td>
</tr>
<tr>
<td>YearDay(“7-1-08”)</td>
<td>183</td>
<td>Returns the day of the year (counting consecutively from January 1) for the given date. (Since 2008 is a leap year the number is one greater.)</td>
</tr>
</tbody>
</table>

**See also**

- Date Functions on page 51
- Locales on page 55
- Date Formats on page 52
- Year on page 448
Chapter 3

Keyword Reference

This chapter contains a reference, in alphabetical order, of all the keywords you can use in your DAL scripts.

See the Keyword Table on page 452 for a list of the keywords. See Grammar and Syntax on page 14 for more information on using DAL.
This table lists each keyword and provides a description of the keyword. Click on the function name to jump to a discussion of that function.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>And</td>
<td>Include AND to perform a logical conjunction on two Boolean expressions.</td>
</tr>
<tr>
<td>BeginSub</td>
<td>Include a BeginSub statement at the beginning of each subroutine in a DAL subroutine library.</td>
</tr>
<tr>
<td>Break</td>
<td>Use a Break statement to exit a While..Wend statement block.</td>
</tr>
<tr>
<td>Continue</td>
<td>Use a Continue statement to restart a While..Wend statement loop.</td>
</tr>
<tr>
<td>Else</td>
<td>Include an Else statement if you want to pass control to the statement that follows this keyword if the logical expression is false.</td>
</tr>
<tr>
<td>ElseIf</td>
<td>If the first logical expression is false, the first ELSEIF logical expression is evaluated.</td>
</tr>
<tr>
<td>End</td>
<td>Include an End statement to end an IF, ELSEIF, or ELSE statement</td>
</tr>
<tr>
<td>EndSub</td>
<td>Include a EndSub statement to end each subroutine in a DAL subroutine library.</td>
</tr>
<tr>
<td>Goto</td>
<td>Include a Goto statement to move to a specific location within a calculation.</td>
</tr>
<tr>
<td>If...End</td>
<td>Use IF statements to execute commands based on the occurrence of a given condition.</td>
</tr>
<tr>
<td>Or</td>
<td>Include OR to perform a logical disjunction on two Boolean expressions.</td>
</tr>
<tr>
<td>Return</td>
<td>Use a Return statement to tell the calculation to return with or without a value.</td>
</tr>
<tr>
<td>While...Wend</td>
<td>Use While...Wend statements to execute a series of statements, as long as a given condition is true.</td>
</tr>
</tbody>
</table>
**AND**

When you have two Boolean expressions, use this keyword to have the system return True if both Boolean expressions evaluate to True. If either or expression evaluates to False, AND returns False.

**Syntax**

AND

There are no parameters for this keyword.

**See also**

- Or on page 464
- Keyword Table on page 452
**BEGINSUB**

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

**Syntax**

```
BEGINSUB Name
```

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

**Parameter**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter the name of the subroutine.</td>
</tr>
</tbody>
</table>

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

**Example**

Here is an example:

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

BEGINSUB Script2
  * This script returns a negative one if #y was equal to 5.
  IF (#y = 5) THEN Return (-1);
  END;
ENDSUB
```

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

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

**See also**

- Keyword Table on page 452
- EndSub on page 460
Break statements provide a way to exit a While...Wend statement block.

**Syntax**

```
Break (Levels)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Levels</td>
<td>(Optional) The value you enter defines how many nested While...Wend statement blocks you want to terminate. If you omit this parameter, control passes to the statement following the next Wend statement encountered.</td>
</tr>
</tbody>
</table>

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
  ...
Wend
```

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

See also [Keyword Table on page 452](#)
**CONTINUE**

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

**Syntax**

```plaintext
Continue
```

There are no parameters for this keyword.

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

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

**Example**

Here is an example:

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

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

**See also**  
Keyword Table on page 452
**ELSE**

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

**Syntax**

Else

There are no parameters for this keyword.

**Example**

Here is an example:

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

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

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

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

Use of the keyword connector THEN is optional.

**See also**

Keyword Table on page 452

If...End on page 462
**ELSEIF**

An IF statement with an ELSEIF condition is the most complicated type of IF statement:

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

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

**Example**

Here is a sample IF statement with ELSEIF condition:

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

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

**See also**

Keyword Table on page 452

If...End on page 462
An IF statement is executed based on the occurrence of a certain condition. IF statements must begin with the keyword IF and terminate with the keyword END.

**Syntax**

```
End
```

There are no parameters for this keyword.

**See also**

- Keyword Table on page 452
- If...End on page 462
**EndSub**

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

**Syntax**

```plaintext
EndSub
```

There are no parameters for this keyword.

BeginSub and EndSub must be paired for each script.

**Example**

Here is an example:

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

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

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

**See also**

- Keyword Table on page 452
- BeginSub on page 454
**GOTO**

A GOTO statement moves to a specific location within a calculation. The location has been named with a label. (See Labels on page 22 for more information.)

**Syntax**

GOTO Location

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>Specify the location you want to go to. For instance, enter the name of a section on a form.</td>
</tr>
</tbody>
</table>

A GOTO statement must begin with the keyword GOTO.

**Example**

Here is an example:

```
GOTO SECTION_ONE:
```

The control jumps to SECTION_ONE in a calculation.

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

GOTO will support retrieving the label from a target variable.

Here is another example:

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

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

**See also**

Keyword Table on page 452
**IF...END**

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

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

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

  ```
  IF (@("FirstAmount") < 1000.00) THEN
    $FinalAmount = @("FirstAmount") * .05;
  END;
  RETURN ($FinalAmount)
  
  CALCULATION: If the value of the section variable field FirstAmount is less than 1000.00 then the value is multiplied by .05 and entered in the target variable $FinalAmount. The value of the $FinalAmount target variable is then returned to the section variable field.
  
  The use of the keyword connector THEN is optional.
  ```

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

  ```
  IF (@("FirstAmount") < 1000.00) THEN
    $FinalAmount = @("FirstAmount") * .05;
  ELSE
    $FinalAmount = @("FirstAmount") + 10.00;
  END;
  RETURN ($FinalAmount)
  
  CALCULATION: If the value of the section variable field FirstAmount is less than 1000.00 then the amount is multiplied by .05 and entered in the target variable $FinalAmount. If the value of the section variable field FirstAmount is greater than or equal to 1000.00 then 10.00 is added to the amount and entered in the target variable $FinalAmount.
  
  The value of the $FinalAmount field is then returned to the caller or section variable field.
  
  The use of the keyword connector THEN is optional.
  ```

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

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

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

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

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

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

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

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

**See also**
- Keyword Table on page 452
- Else on page 457
- ElseIf on page 458
OR

When you have two Boolean expressions, use this keyword to have the system return True if either Boolean expression evaluates to True. If neither expression evaluates to True, OR returns False.

Syntax

OR

There are no parameters for this keyword.

See also

And on page 453
Keyword Table on page 452
A RETURN statement directs the calculation to return with or without a value. A RETURN statement must begin with the keyword RETURN. A RETURN statement may return the result of the calculation to be placed in the field that initiated the script.

A RETURN statement is also used to return results to one calculation script from another. Using a CALL statement temporarily suspends the current script calculation and sends control to another script file. A RETURN statement sends control back to the original script which may then continue processing.

Here are some sample RETURN statements:

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

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

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

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

See also
- Keyword Table on page 452
- Call on page 160
**WHILE...WEND**

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

**Syntax**

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

- **Condition**
  Required. The condition is any expression that evaluates to true or false. False is assumed to be a zero value. Any non-zero value is assumed to be true.

- **Statements**
  One or more statements executed while the condition is true.

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

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

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

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

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

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

See also  [Keyword Table on page 452](#)
Index

Symbols

" (quotation marks) 16, 17
& (ampersands) 199, 289, 378
( ) (parentheses) 16, 19
* (asterisks) 4, 65
: (colons) 6
; (semicolons) 3, 6
? function 111
@ function
defined 109
NUM function 325
\ (backslashes) 96
’ (apostrophes) 17

A

ABS function 113
accessing
database fields 49
ACIF 35
AddBlankPages function 115
AddComment function 117
AddDocusaveComment function 118
AddForm function
   CopyForm function 174
AddForm_Propagate function 120
AddImage function 122
AddImage_Propagate function 125
AddOvFlwSym function 127
AddressCount function 128
AFEBatchDALProcess 7, 371
AFELOG file
  AFELog function 129
  DelWIP function 215
AFELog function 129
AFEProcedures control group 12
alphabetic field format 62
alphanumeric field format 63
Always function 130
AND 147, 453
annuities 340
apostrophes (‘) 17
AppendText function 132
AppendTxm function
  AppendTxmUnique function 137
  defined 134
AppendTxmUnique function 136
AppIdxRec function 139
ApplyInserts function 140
archives
  adding comments 117
  Complete function 168
  retrieving records 139
ASCII files
  AddDocusaveComment function 118
  DAL script libraries 5
  scripts 4
Ask function 141
assignment statements 14
AssignWIP function 142
asterisks (*)
  comment lines 4
  wildcards 63
AutoKeyID
  option 12
  table 11
Avg function 143

B

backslashes in object names 96
BankRound function 145
banner processing
  RecipBatch function 353
  RecipName function 356
  SuppressBanner function 403
bar code fields
  format 62
  SetFont function 379
batch processing
  DDTSourceName function 205
Batch_DAL control group 7
Beep function 146
BeginSub 454
BeginSub function
  EndSub function 29
binding 179
bit logical shift operation 154
BitAnd function 147
BitClear function 148
bitmaps
  ChangeLogo function 163
  DelLogo function 214
  embedding logos 229
  HaveLogo function 272
  InlineLogo function 282
  Refresh function 357
  RenameLogo function 359
BitNot function 149
BitOr function 150
BitRotate function 151
BitSet function 153
BitShift function 154
BitTest function 156
bitwise
  AND operation 147
  exclusive OR operation 157
  inclusive OR operation 150
  logical NOT operation 149
  shift operation 154
  shift-and-rotate operation 151
BitXor function 157
blank lines
  formatting scripts 3, 4
blank pages 115, 208
Boolean values
  GetINIBool function 257
  PutINIBool function 349
Break 455
Break statements
  and While...Wend statements 26
built-in functions 39

C

cache
  GetINIString function 259
  LoadINIFile function 300
  LoadLib function 301
  SaveINIFile function 370
Calculation tab
  assigning a calculation 2
  comments 4
calculations
  entering in an external file 4
  POW function 340
CALL function 160
CALL statements
  defined 25
carbon-copy recipients 391, 392
carriage returns 4
case
  dates 52
  sensitivity 3, 4
  target variables 15
CaseSensitiveKeys option 11
century
  cut-off 54
  DateCnv function 186
CFind function 162
CHAIN function 161
CHAIN statements
  defined 25
ChangeLogo function 163
Char function 165
character strings
  converting to an integer value 275
  ListInList function 298
CharV function 166
CheckImageLoaded rule
  SetFld function 377
Class option 46
clearing
  BitClear function 148
CLIPSACES 200
CodeInList function 167
colons
  use of 6
commas
  numeric constants 16
  comma-separated value files 334
comment record processing
  RecipBatch function 353
  RecipName function 356
comments
  AddComment function 117
  AddDocusaveComment function 118
  creating strings 324
  lines 4
CompileWhenLoaded option 8
Complete function 168
CompressFlds function 169
compressing
  blank space 169
ConnectFlds function 171
Continue 456
Continue statements
  While...Wend statements 27
Control control group 8
control groups
  INI functions 70
control-z 4
converting
  character strings to integer values 275
  integer values to hexadecimal string values 206
coordinates
  ImageRect function 278
  SetImagePos function 382
copies
  counting recipient copies 354
  TotalPages function 412
CopyForm function 174
Count function 175
CountRec function 177
CreateIndex option 44
CreateTable option 44
custom field format 62
Cut function 178

DAL
  assignment statements 14
  calcs 2
  CALL statements 25
  CHAIN statements 25
  data flow statements 14
  DBUnloadDFD function 202
  debugger 31
  entering calculations in an external file 4
  examples 2, 41
  execution order 19
  flow control statements 14, 22
  format in external files 4
  GOTO statements 25
  IF statements 23
  implicit conversion 20
  keywords 22
  labels 22
  numeric constants 16
  operators 17
  punctuation 18
  retaining variables 362
  RETURN statements 23
  runtime error messages 33
  runtime options 8
  section variable fields 16
  source expression 16
  string constants 17
  target field 16
  using the Properties window 3
DAL control group 8
DAL rule
  ? function 111, 112
DAL scripts
  defined 2
  executing 344
  retrieving XML data 89
DALFunctions control group 8
DALLib option 9
DALLibraries control group 8
DALRun
  built-in function 10
  control group 9
  DBUnloadDFD function 202
  INI file 31
DALTriggers option 9
DALVAR built-in function 10
DashCode function 179
data flow statements 14
data storage statements 30
database functions
  accessing fields 49
  DB2/2 handler 45
  handlers for Excel 46
  list of 43
  ODBC handler 44
date field format 62
Date function 182
Date2Date function 183
DateAdd function 184
DateCnv function 186
DateFmt rule
  ? function 112
DateFmt2To4Year option 54
DateFmt2To4Year option 8
dates
  century cut-off 54
  data storage statements 30
  formatting 52
  list of functions 51
  locale considerations 87
Day function 188
daylight savings time 410
DayName function
  defined 189
  WeekDay function 424
DaysInMonth function 190
DaysInYear function 191
DB2/2 handler 45
DBAdd function 192
DBClose function
  defined 193
  memory tables 50
  record lengths 200
DBDelete function 194
DBFind function 195
DBFirstRec function 197
DBNextRec function 198
DBOpen function
  defined 199
  memory tables 50
  record lengths 200
DBPrepVars function 201
DBUnloadDFD function 202
DBUpdate function 203
DDT files
  DDTSourceName function 205
  storing information 6
DDTSourceName function 205
Debug_DAL_Rules option 9
Debug_Switches control group 9
Dec2Hex function 206
decimal target variables 15
DEFLIB directory 4
DeFormat function 207
DelBlankPages function 208
DelField function 209
DelForm function 211
DelImage function 212
DelWIP function 215
descriptions
  retrieving 246
DestroyList function 216
DDF files
  DBUnloadDFD function 202
DiffDate function 219
DiffDays function 220
DiffHours function 221
DiffMinutes function 222
DiffMonths function 223
DiffSeconds function 224
Index

DiffTime function 225
DiffYears function 226
directing workflow 391, 392
disabling scripts 2
divide by zero 33
dividing year 186
Docusave
   AddDocusaveComment function 118
      adding comments 117
DocusaveScript option 35
dot operator 49
double quotes 17
drives
   FileDrive function 240
dummy pages 115
DumpDAL option 9
DupForm function
   CopyForm function 174
      defined 228

E

EBCDIC
   AddDocusaveComment function 118
   either required 236
Else 457
ElseIf 458
e-mail addresses 392
EmbedLogo function 229
End 459
EndSub 460
EndSub function
   BeginSub function 28
ERRFILE.DAT file 367, 369
errors
   Beep function 146
      RPErroRMsg function 367
      runtime error messages 33
Excel
   databases 46
   exclusive OR operation 157
   execution order 19
   Exists function 230
   exponential power
      using the POW function 340
exporting
   Complete function 168
Ext option
   defined 8
      LoadLib function 301
external files
   using 4
extract files
   CountRec function 177
   retrieving data 395
extracting a field’s root name 364

F

FAP units
   Logo function 303
   positioning sections 383
field formats
   list of 62
      locating fields 64
FieldFormat function 231
FieldName function 232
FieldPrompt function 234
fields
accessing database fields 49
changing coordinates 171
compressing blank space 169
concatenating text 171
ConnectFlds function 171
date formats 52
deleting 209
extracting the root name 364
functions 61
JustField function 291
locating 64
moving horizontally 393
renaming 136
SpanField function 393
specifying 16
target fields 14
using the Properties window 3
FieldType function 237
FieldX function 238
defined 238
Logo function 303
FieldY function
defined 239
Logo function 303
FileDrive function 240
FileExt function 241
FileName function 242
FilePath function 243
files
DAL scripts 5
entering calculations in an external file 4
FileExt function 241
FileName function 242
FilePath function 243
FullFileName function 249
filler pages 115, 208
Find function 244
flow control statements
defined 14
keywords 22
FlushDALSymbols option 9, 362
FlushSymbols option 8
fonts
changing 379
form descriptions, retrieving 246
FORM PAGE NUM field 109
FORM PAGE NUM OF field 109
FORM.DAT file
RecipientName function 355
retrieving descriptions 246
Format function 245
formats
DAL format in external files 4
date 52
numeric 63
time 80
formatting functions
fields 61
string functions 78
FormDesc function 246
FormName function 247
forms
AddForm_Propagate function 120
changing the description 380
CopyForm function 174
DupForm function 228
WhatForm function 426
FORMSET PAGE NUM field 109
FORMSET PAGE NUM OF field 109
FormsetID field
SetWIPFld function 389
four-digit years 186
FrenchNumText function 248
### Index

**FSISYS.INI file**
- DAL script extensions 4
- executing DAL scripts 7
- GetINIBool function 257
- GetINIString function 259
- IgnoreInvalidImage option 123
- INI functions 70
- LogEnabled option 257
- options for Docusave 35
- options for OnDemand 35
- PutINIBool function 349
- runtime options 8

**FSIUSER.INI file**
- executing DAL scripts 7
- GetINIBool function 257
- GetINIString function 259
- INI functions 70
- LogEnabled option 257
- options for Docusave 35
- options for OnDemand 35
- PutINIBool function 349
- PutINIString function 351

**FullFileName function** 249

**functions**
- mathematical 70, 72
- miscellaneous 73, 76, 88
- object 94
- overview 41
- string 78
- time 80
- where used 97

**G**

**get field function** 109
- GetAddresseeValues function 250
- GetAttachVar function 252
- GetData function 253
- and the SrchData function 253, 395
- GetFormAttrib function 255

**GetINIBool function**
- defined 257

**GetINIString function** 259

**GetListElem function** 261

**GetOvFlwSym function** 262

**GetValue function** 263

**GoTo** 461

**GOTO statements**
- defined 25
- runtime error messages 34
- While loops 28

**graphics**
- applying 140
- deleting 214
- in-lining 282
- locating 272
- renaming 359

**GroupName function** 264

**groups**
- PrinterGroup function 344
- WhatGroup function 427

**GVM function**
- defined 265

**GVM variables**
- HaveGVM function 270
- printing 342
- SetGVM function 381

**H**

**HaveField function** 266

**HaveForm function** 268

**HaveGroup function** 269

**HaveGVM function** 270

**HaveImage function** 271

**HaveLogo function** 272

**HaveRecip function**
- defined 274
- RecipientName function 355

**Hex2Dec function** 275
hexadecimal values
  date formats 54
  Date2Date function 183
  Dec2Hex function 206
  Hex2Dec function 275
host required 236
Hour function 276
hyperlinks
  SetLink function 384

ICU system time zones 81
IF rule 235, 236
IF statements
  defined 23
  runtime error messages 33
IgnoreInvalidImage option 123
imageName function 277
ImageRect function
  AddImage function 123
  defined 278
implicit conversion 20
IncOvFlwSym function 280
INI files
  DAL options 8
  GetINIBool function 257
  GetINIString function 259
  LoadINIFile function 300
  PutINIBool function 349
  PutINIString function 351
INI function 281
INI functions 70
INIGroup control group 10
InlineLogo function 282
Input function 283
Insert function 284
inserting equipment 179
insertion text 63
Install option 44

INT function 285
integers
  BitAnd function 147, 148
  BitTest function 156
  BitNot function 149
  BitOr function 150
  BitRotate function 151
  BitSet function 153
  BitShift function 154
  BitXor function 157
  Char function 165
  CharV function 166
  converting to hexadecimal string values 206
  Dec2Hex function 206
  Hex2Dec function 275
  returning the remainder 320
  target variables 15
interest rates
  POW function 340
international
  alphabetic field format 62
  alphanumeric field format 62
  uppercase alphabetic field format 62
  uppercase alphanumeric field format 62
  IsPrintObject function 286
  IsXMLLError function 287

J

JCenter function 288
JLeft function 289
JRight function 290
JustField function 291

K

KeyID values 11
Keyword option 8
Index

keywords 22
    BeginSub and EndSub 28
KickToWIP function 293
KickToWIP rule 236

L

labels
    in scripts 22
    runtime error messages 34
leading signs 326
leading spaces 329, 419
leap years
    DateAdd function 184
    DaysInMonth function 190
    DaysInYear function 191
    DiffYears function 226
    LeapYear function 295
    YearDay function 449
LeapYear function 295
Left function 296
LEN function
    defined 297
    Size function 390
Lib option 8
libraries
    LoadLib function 301
    of DAL scripts 5
limits
    significant numbers 72
line breaks
    MLEInput function 314, 317
line feeds 4
ListInList function 298
LoadCordFAP option 163
LoadExtractData rule 253
LoadINIFile function 300
LoadLib function
    DAL libraries 6
    defined 301
LoadXMLList rule 90
locale 52
locales 55
times and dates 87
locating fields 64
locating objects 94
log files
    RPLLogMsg function 368
LOGFILE.DAT file 368
logical NOT operation 149
logical shift 154
Logo function 303
Lower function 305
lowercase
dates 52

M

MailWIP function 306
MajorVersion function 307
master resource library
    storing external script files 4
MasterResource control group 9
mathematical functions 72
MAX mathematical functions 72
memory
    GetINIString function 259
    LoadINIFile function 300
    LoadLib function 301
    runtime error messages 33
    SaveINIFile function 370
tables 50
MEN.RES file
    enabling the DAL debugger 32
    executing a DAL script 7
menus
    executing a DAL script from 7
messages
   AFELog function 129
   Ask function 141
   Beep function 146
   creating 323
metadata 255, 347
MIN function 310
MinorVersion function 312
minus signs 17
Minute function 313
miscellaneous functions 73, 76, 88
MLEInput function
defined 314
MLETranslate function 317
MLETranslate function
and MLEInput 314
defined 317
MLEInput function 317
MOD function 320
month abbreviations 54
Month function 321
MonthName function 322
Move_It rule
   ? function 111
moving data to compress blank space 169
MSG function 323
Multicopy option 174, 228
multi-line text area messages, creating 324
multi-line text field format 62
multi-line variable fields
   MLEInput function 314, 317
   MLETranslate function 317
MYPAGE variables 331

name
   FileDrive function 240
   FileName function 242
   FullFileName function 249
new line character
   MLEInput function 314, 317
NewFormatOnly option 404
NL function 324
NOT operation 149
not required 236
NUM function 325
numeric constants 16
numeric field format 62
numeric formats 63
Numeric function 326
NumText function
defined 327
   FrenchNumText function 248

O

object functions
   list of 94
   locating objects 94
occurrence
   counts 65
   PageImage function 330
ODBC
   DBUnloadDFD function 202
   handler 44
Ok buttons 323
OldFormatOnly option 404
OMR marks 208
OnCreate option 12
OnDemand, adding comments 117
OnDemandScript option 35, 343, 344
OnUpdate option 12
operator required 236

N

NAFILE.DAT files
   embedding graphics 229
operators
   defined 17
   dot 49
   source expressions 19
options
   setting using INI functions 70
OR 464
OR operation 150, 157
OutMode option 35
overflow
   AddOVFlwSym function 127
   AppendTxmUnique function 137
   FieldRule function 235
   GetOVFlwSym function 262
   IncOVFlwSym function 280
   ResetOVFlwSym function 361
overflow record count
   retrieving 418

PAD function 329
page numbering fields
   @ function 109
PageImage function 330
PageInfo function 331
pages
   PageImage function 330
   size 331
   TotalPages function 412
paginateForm function 333
paragraphs
   importing 136
parameters
   punctuation 18
   syntax of 41
parentheses
   specifying field names 16
ParseListCount function 334
ParseListItem function 336
partial names
   examples of 66, 143, 308, 310, 401
   object functions 95
PassWd option 44
PathCreate function 338
PathExist function 339
paths
   FileDrive function 240
   FileExt function 241
   FileName function 242
   FilePath function 243
   FullFileName function 249
POW function 340
Print function 341
print functions 76
print streams
   adding comments 117
   PrinterClass function 343
   PrinterGroup function 344
Print window 341
Print_It function
   defined 342
   NL function 324
PrinterClass function 343
PrinterGroup function 344
PrinterID function 345
PrinterOutputSize function 346
printing
   Complete function 168
   determining if a section will print 286
   in-lining graphics 282
PrintViewOnly option 343, 344
procedures 41
prompts
   AFELog function 129
   creating 314
   DBDelete function 194
   FieldPrompt function 234
   Input function 283
   ODBC drivers 44
propagate
   AddForm_Propagate 120
Properties window
   comments 4
   entering DAL calcs 2, 3
PrtType option 343, 344
punctuation 18
purging WIP 215
PutFormAttrib function 347
PutINIBool function 349
PutINIString function 351

Q

Qualifier option 44
quotation marks
   @ function 109
   date formats 52
   field formats 62
   specifying field names 16
   string constants 17

R

RecipBatch function
   defined 353
RecipCopyCount function 354
recipients
   HaveRecip function 274
   page size 332
   TotalPages function 412
RecipName function
   defined 356
record lengths
   trailing spaces 200
records
   minimum number 177
Refresh function
   AddImage function 123
   defined 357
   DelLogo function 214
   Logo function 303
remainder
   returning 320
RenameLogo function
   defined 359
reserved keywords 15
ResetFld function
   defined 360
ResetOvFlwSym function 361
Retain function 362
retrieving
   a string with a new line sequence 324
   the overflow record count 418
   the SourceName field 205
RETURN 465
return
   values 41
RETURN statements
   defined 23
   runtime error messages 34
   WIPExit function 429
Right function 363
RootName function 364
Round function 365
rounding with the BankRound function 145
RouteWIP function 366
routing slips
   RouteWIP function 366
   SlipInsert function 392
RPEErrorMsg function 367
RPLogMsg function 368
RPWarningMsg function 369
RunMode control group 9, 163
runtime
   error messages 33
   options 31
Index

S
SAMPCO sample resources 2, 41
SaveINIFile function 370
SaveWIP function 371
Script option 9, 12
ScriptFile option 7
scripts
creating libraries 5
defined 2
disabling 2
executing 344
executing from a menu 7
LoadLib function 301
maximum size 3
runtime error messages 33
runtime options 8
SlipInsert function 392
search criteria
including spaces 253, 395
search masks
CountRec function 177
searching
character string list 298
Second function 372
sections
adding 122
checking 2
PageImage function 330
re-paginations 333
repositioning 382
retrieving coordinates 278
variable fields 16
WhatImage function 428
semicolons
formatting calculations 3
use of 6
separators 52
sequence numbers
HaveRecip function 274
Server option 44, 46
SetEdit function 375
SetFld function 377
SetFont function 379
SetFormDesc function 380
SetGVM function 381
SetImagePos function
AddImage function 123
defined 382
SetLink function 384
SetOvFlwSym rule 127
SetProtect function 386
SetRecip function 387
SetRecipTb
triggering the form name 416
triggering the section name 417
SetRequiredFld function 388
setting the bit position 153
SetWIPFld function 389
shiftAmt value 151
shift-and-rotate operation 151
ShowWIPWarning option 293
signatures 140, 163
significant numbers 72
Size function 389, 390
SlipAppend function 391
SlipInsert function 392
source expressions
defined 16
operators 19
punctuation 18
Source Name field
DDTSourceName function 205
spaces
in scripts 3, 4
including 253, 395
trailing 200
SpanField function 393
spreadsheets 334
SrchData function 395
standard export format
Complete function 168
state stamps 140
statements
  data storage 30
  operators 19
  punctuation 18
  separators 6
STR function 397
STRCompare function 398
string functions 78
strings
  CodeInList function 167
  comparing 398
  constants 17
  ListinList function 298
  printing 342
  retrieving 324
  space and tab characters 3
  target variables 15
subroutines
  BeginSub function 28
  EndSub function 29
SUM function 401
SuppressBanner function
  defined 403
symbolic variable
  Exists function 230
  GetValue function 263
symbols
  runtime error messages 33
  statement continuation 18
syntax
  errors 41
  runtime error messages 33

tables
  setting up memory tables 50
  target fields 14, 16
  target variables
    decimals 16
    defined 14
    integers 16
    strings 16
TblLkUp rule 235
TblText rule 235
testing a specified bit 156
text
  concatenating 171
  creating a window for entering 314
  searching for 162
time
  formats 80
  functions 80
Time function 406
Time2Time function 407
TimeAdd function 408
times
  locale considerations 87
TimeZone function 409
TimeZone2TimeZone function 410
Title option 9
TLEs 35
TotalPages function 412
TotalSheets function 413
trailing signs 326
trailing spaces 329, 419
translating new line characters 317
Trigger2Archive control group 139
TriggerForm function
  defined 414
TriggerFormName function 416
TriggerImageName function 417
triggering
  form name 416
  section name 417
TriggerRecsPerOvFlw 418
Trim function 419

T

tab characters
  evaluating scripts 3
  in scripts 4
Table function 404
table only field format 62
two-digit year 186

U

Upper function 420
uppercase alphabetic field format 62
uppercase alphanumeric field format 63
uppercase dates 52
user ID
assigning WIP 142
User option 44
UserID function 422
UserLvl function 423
using DAL 1

V

values
absolute 113
POW function 340
variable fields
assign a calculation 2
field formats 62
functions 61
locating fields 64
mathematical functions 70, 72
miscellaneous functions 73, 76, 88
object functions 94
resetting 360
variables
deleting 8
prefix names 49
retaining 362
target variables 14
trailing spaces 419
VerifyKeyID
control group 9, 12
hook 11

version numbers
MajorVersion function 307
MinorVersion function 312

W

warnings
Beep function 146
RPWarningMsg function 369
WeekDay function 424
While 466
WhatForm function 426
WhatGroup function 427
WhatImage function 428
While 466
While...Wend statements 26
white space 3
wildcards 65
window
creating 314
WIP
DelWIP function 215
KickToWIP function 293
SaveWIP function 371
setting WIP fields 389
WIPExit function 429
WIPFId function 430
WIPKey1 function 431
WIPKey2 function 432
WIPKeyId function 433
work-in-process, assigning 142

X

X or space field format 62
XDB database
? function 111
file information 6
XML
  API functions 89
  GetData function 253
XML extract files 395
XMLAttrName function 434
XMLAttrValue function 435
XMLFileExtract rule 90
XMLFind function 436
XMLFirst function 437
XMLFirstAttrib function 438
XMLFirstText function 439
XMLGetCurName function 440
XMLGetCurText function 441
XMLNext function 442
XMLNextAttrib function 443
XMLNthAttrName function 445
XMLNthAttrValue function 445, 446
XMLNthText function 447
XOR operation 157
XPaths 91, 253, 395
XPATHW32 program 91

Y

Y or N field format 63
Year function 448
YearDay function 449
years
  DiffYear function 226
  forcing 2-digit 54
  sizes 52

Z

zero
  runtime error messages 33