ORACLE
INSURANCE

Oracle® Documaker

Internet Document Server Guide

version 2.2

Part number: E14902-01

May 2009

Copyright © 2009, Oracle. 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, MERCHANTABLE 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, MERCHANTABLE 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, MERCHANTABLE 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 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 BELIABLE 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, Processing Documents Using the Internet

2 Overview

5 Architectural Changes in Version 2.x

6 Required Components

6 Components Available from Oracle Insurance

Chapter 2, Using the Internet Document Server

11 Overview

13 Creating Front-End Solutions

13 Using JSP

14 Using the IDSJSP JavaBean

15 Using the JSP Samples

15 Handling Multi-Part/Form-Data Forms

16 Using ASP

17 Using the IDSASP Object

19 Sending and Receiving Attachment Fields

22 Showing a PDF File

25 Using the ASP Samples

25 Using the HTTP Parsing and Uploading APIs

30 Using the XMLSession Rules

30 IDSASP Methods

32 IDSJSP Methods

34 XMLSession Rules

37 Using IDSXML

38 XMLTransformErrors

39 XMLTransformErrors2

40 XMLLoadINI

41 XMLLoadXML

41 XMLLoadXSL

42 XMLGetGroupOptionValue

42 XMLGetValue

42 XMLGetGroup

44 XMLUpdateGroup
45 XMLBuffer
45 XMLLoadProcessor
46 XMLAddParameterToXSL
46 XMLTransformWithXSL
47 XMLProcessWithXSL
48 XMLUpdateFormset
48 XMLProcessFormset
50 Using Multiple Servers
51 Determining if Your Transactions are CPU or I/O Intensive
52 Performance Measurements when Using Multiple Servers
53 Setting Up Additional Servers
54 Setting Up a Windows NT Service
55 Handling Multi-threaded Requests
57 Using the Java Test Utility
58 Using Rules Written in Other Scripting Languages
59 Using IDS as a Client to Another IDS
60 Using the IDSClientRule
64 Monitoring IDS with SNMP Tools
65 Monitoring Requests
66 Managing IDS Instances
75 Sending Results and Receiving Requests in Multiple Formats
76 Configuring and Deploying Marshallers
78 Logging and Tracing
81 Naming Logging Messages
84 Using Logging Categories
87 Logging Information about Requests
91 Querying Transaction Information
92 getMetaData
93 QueryTranLogs
94 Monitoring Performance Statistics
95 Generating a Logging Configuration File
95 Using Logging Categories to Access the Internal Format of Requests
97 Configuring IDS
98 Running IDSConfig

98 Creating New Files
98 Adding Nodes
98 Adding Nodes with Text
99 Editing Nodes
99 Copying Nodes
99 Moving Nodes
99 Adding Attributes
100 Adding Comments
100 Adding Text
100 Adding a Request or Function
100 Adding an IDS Function
101 Converting DOCSERV.INI or DOCCLIENT.INI Files into XML Format
101 Adding a Section or Entry
101 Locating Text
101 Importing Configuration Information
102 Configuring MQSeries Buffer Sizes
103 Testing File Transmission

106 Referencing Attachment Variables

107 Using Unicode in Attachment Variables

108 Using the Message Queues

108 Choosing the Right Queuing Options
109 Understanding the Router Process
110 How HTTP Queues are Handled
110 Using the Router Section

111 Using Multiple Queuing Systems

113 Using the Java Message Service (JMS)

113 Setting up JMS

116 Using WebSphere MQ

117 Setting Up WebSphere MQ

118 Using MSMQ

123 Using Security Exits

124 Using Client Connection Definition Tables

124 Using SSL Connections

125 Using the ReplyToQueueName and ReplyToQueueManagerName Properties

126 Using HTTP

130 Using Multiple Bridges

132 Submitting Batch Requests

134 Printing in Duplex Mode to PCL Printers

135 Using IDS to Distribute Email

135 Modifying the docserv.xml Configuration File

136 Modifying the DAP.INI File

138 Attachment Variables Used by Email Rules

139 Using Email Rules

141 Using the Email Bus

144 Using IDS to Run Documaker

145 Setting Up IDS

146 Setting up Multiple Internet Document Servers

146 Controlling Documaker

149 Setting Up Documaker

151 Naming Conventions for Output Files

152 Creating DPW Files

153 Accessing IDS Attachment Variables in GenData

153 Using TCP/IP Communications

155 Customizing the Execution of Documaker

158 Using the XML Messaging System

160 Client Request Messages

163 Server XML Response Messages

164 Using XML SOAP Outside of Messaging Systems

166 Connecting to an SQL Database

167 Differences between Microsoft's ADO and IDSSQL

167 Setting up IDSSQL

167 IDSSQL Classes

167 IDSSQL.ADO

168 IDSSQL.IDSRC

168 Example Script

171 Using the Thin Client Forms Publisher

172 Pausing IDS

172 DSIQueryStatus

173 DSISetStatus

175 Executing Request Types at Run Time

177 Publishing Your Forms on the Web

177 FORMPUB
178 FD2HTW32
178 PTFMDW32
179 FAP2HTML
180 INI Options
182 Handling Multi-Part/Form-Data Forms
183 Formatting Text with XML Markup
184 Encrypting and Decrypting Data Files
185 Using Multiple Attachment Values with the Same Name
187 getEntries
188 Converting XML Files Using a Template
192 Customizing Your System
195 Handling Security Issues
195 Using Firewalls
195 Implementing Security for Web Applications
197 Using the FAP2XML Utility
198 Using LDAP Support
199 Using Default Time-outs for DSILIB-Based Client Applications
201 Running Timed Requests
202 In-Process Rendering for DPAView
202 DRLGetConfig
203 Using DAL Functions for WIP Column Access
205 Using Enterprise Web Processing Services

Chapter 3, Creating Output Files

208 Creating PDF Files
209 Setting Up the PDF Print Driver
212 Creating PDF Files with Unicode Support
212 Setting PDF Compression Options
213 Producing Optimal PDF Output
215 Handling Fonts
216 Not Embedding Fonts

217 Embedding Fonts
219 Handling Fonts with Multiple Width Tables
220 Using the PDF Print Driver with GenPrint
220 Changing the GenPrint Program
222 Generating Separate Files
223 Font Cross Reference File Tips
224 Embedding Fonts
226 Using the 14 Base Fonts Distributed with Acrobat Reader
227 Setting Up Bookmarks
229 Limitations
230 Creating HTML Files
235 Producing Table Information for TextMerge Paragraphs
236 Creating XML Output

Chapter 4, Using Print Preview

239 Touring Print Preview
245 Using the JSP Files
247 Customizing Print Preview
248 Using INI Options to Customize Print Preview Screens
249 Customizing the WIP List Page
250 Creating User Lists for Print Preview
251 Request Types for Print Preview

Chapter 5, Using Docucorp Publishing Services

256 DPS Object Properties
262 Setting Default Parameters
264 Sample VB Code
265 Sample C Code
267 Sample Java Code
269 Setting Up IDS
271 Setting Up Documaker

Chapter 6, Customizing iDocumaker, iPPS, and WIP Edit

274 Setting Up a Favorites List for iDocumaker

276 Attaching Files to Transactions as Forms

- 276 Specifying the File Name and Type in IDS Attachment Variables
- 277 Sending the File to IDS in a Message
- 277 Storing the File on a Disk Accessible to Documaker Bridge
- 278 Storing the File in a Documanage Repository
- 279 Error Messages
- 279 Specifying Duplex Options for the Attached Form
- 280 Debugging

282 Designating Read-Only Multi-Line Text Field Paragraphs

283 Printing on Your Workstation Printer

284 Preventing the Session from Expiring

285 Passing WIP Record IDs to the MergeWIP Rule

286 Automatically Updating iDocumaker

- 286 Configuring IDS to Update iDocumaker
- 287 Using the VERSUPD Utility
- 289 On the Client Side
  - 290 Additional Utilities
- 291 Checking Version Information

292 Using the WIP Edit Plug-in

- 296 Controlling the Interface
- 302 Setting Up Custom Functions
- 303 Changing the User Associated with a Document
- 303 Sending Passwords
- 304 Requesting a Dictionary
- 305 Trapping Events
- 305 Tracking Session Information
- 307 Setting Up Printers

Chapter 7, Using the DP.DLL ActiveX Interface

312 Requirements

313 Setting Up the Configuration File
315 Properties
316 Methods
317 AddNameValuePair
317 Bin2Unicode
317 CleanCache
318 FileExists
318 GetMsg
318 GetUniqueString
318 Initialize
319 InitializeDefaults
319 ProcessTrn
319 PutMsg
320 ReadIniOptions
320 RequestValue
321 ResultValue
321 SetGUID
321 SOAPAddAttachment
322 SOAPGetAttachment
322 SOAPGetAttachmentAsBuffer
322 SOAPLoadAttachment
323 SOAPUnloadAttachment
323 Terminate
323 Trace
323 Unicode2Bin
324 WriteBinFile
324 WriteToLog
325 Examples

Appendix A, System Files

330 IDS Configuration Files
333 Sample Output Files

Appendix B, Error Messages

340 Displaying Error Messages

344 Internet Document Server Error Messages

346 Documaker Bridge Error Messages

350 Java Error Messages

351 Printstream Bridge Error Messages

352 AFP Error Messages

Appendix C, Choosing a Paper Size

356 US Standard Sizes

357 ISO Sizes

360 Japanese Standard Sizes

361 Printer Support for Paper Sizes

365 Paper Sizes for AFP Printers

367 Index

Chapter 1

Processing Documents Using the Internet

Oracle Insurance offers a comprehensive range of scalable high-performance products for every step in the life cycle of a document. These include...

- Creation Solutions to capture data and create forms
- Publishing Solutions to volume produce personalized documents
- Archival Solutions to intelligently store and retrieve documents
- Management Solutions to control and network documents
- Development Tools to customize your Oracle Insurance solutions

Oracle Insurance's Management Solutions give you the ability to move and view your documents across the enterprise. In addition to advanced document networking communication products, Oracle Insurance has Internet solutions for managing your documents. Docupresentment's Internet Document Server (IDS) helps manage the flow of your documents.

IDS lets you access your documents with a web browser from your intranet or the Internet. The standard web browser interface includes security features, document database lookup, and document viewing in PDF format using the Adobe Acrobat Reader.

This chapter provides an overview of IDS, its concepts, what it can offer you, as well as how it fits into the Oracle Insurance's family of solutions.

OVERVIEW

For many years Oracle Insurance has been creating document solutions capable of handling the high-volume, automated assembly needs of customers like you.

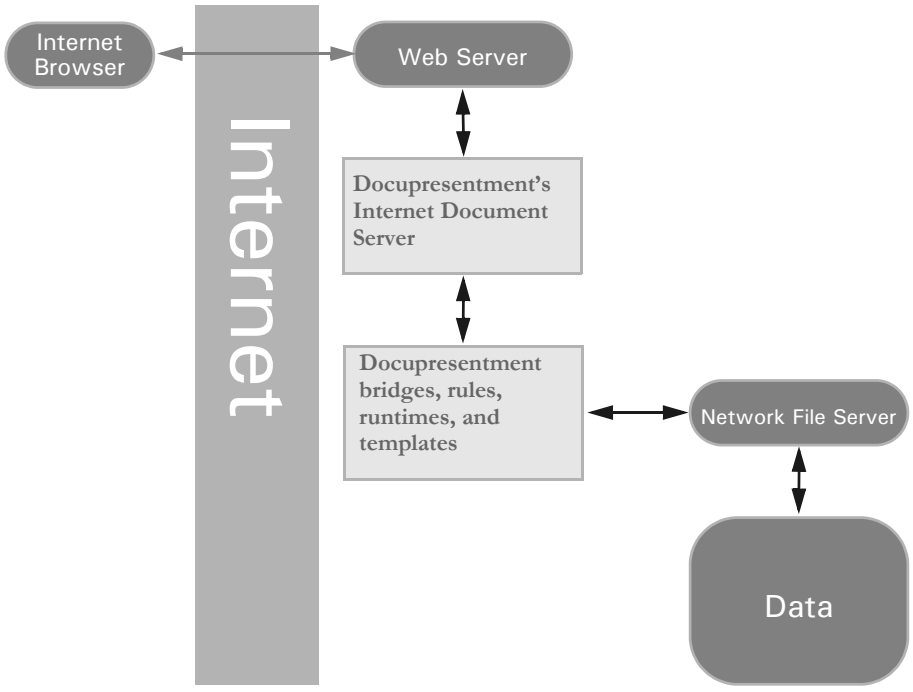
Through products such as Documaker and Documerge, Oracle Insurance has provided clients in industries as diverse as insurance, finance, and utilities, with high-volume document creation, processing, printing, and archiving solutions.

These solutions have typically concentrated on printed output although the real focus has always been to deliver the high quality documents in the most cost-effective manner, and to eliminate paper where ever possible.

The rapid acceptance of the Internet and in-house intranets has created a new and cost-effective way to provide the timely, on-demand delivery of critical documents to remote end-users equipped with only a minimum of standard software.

To address the need for Internet document processing, as well as other new technologies, Oracle Insurance developed a line of products which support *distributed documents*. These new products are collectively called Document Management Solutions and include interfaces to document storage and retrieval systems, as well as WYSIWYG document publishing and delivery via the Internet or your in-house intranet.

The foundation for document publishing and delivery, is Docupresentment's Internet Document Server. The server works with front-end thin clients via the Internet (or an intranet) and executes back-end document processing applications.



Docupresentment supports several installable components, called *bridges*. These bridges provide the software, interface document templates, and runtimes, necessary to process, store, publish, and deliver your documents.

Currently, Oracle Insurance has released several bridges, such as the one to the Documaker archive and the one to Documanage. These bridges provide retrieval and PDF publishing of archived Documaker document sets and a bridge to Metacode and AFP print streams.

In addition, Oracle Insurance also provides a way to distribute documents produced by the GenPrint program, the print component of the Documaker system. With the PDF Print Driver, Oracle Insurance gives Documaker users several ways to distribute documents via the Internet:

- From Documaker's archive component (the GenArc program)
- From Documaker's print component (the GenPrint program)

The Internet Document Server and the bridges to Documaker and Documanage are the first in a series of new products from Oracle Insurance. Over time, new product offerings from Oracle Insurance will provide additional solutions in these areas.

- Internet and intranet-based document processing
- Client-server processing
- Workflow management
- Integration with existing document management and storage subsystems

...as well as other specialized applications that integrate with Windows and Microsoft's BackOffice.

The product architecture uses a layered hierarchy that provides for backward compatibility to existing systems, while positioning for future product offerings.

Management Solutions	**Internet Document Processing**		**Future products**
Bridges	Bridge	Bridge	Bridge
Existing applications	Documanage	Documaker	Other products

- Management Solutions. Some components will provide totally new stand-alone systems and capabilities, while others focus on leveraging and extending existing Oracle Insurance applications. Internet Document Processing is an example of a new component that can be used to leverage existing applications and data with extended functionality.
- Docupresentment Bridges. These components are designed for use with existing applications or other custom built interfaces. Oracle Insurance offers a wide range of technical and professional services for designing and building custom bridges.
- Existing applications. These components cover a wide range of new and existing products and applications from Oracle Insurance's various divisions, as well as other in-house or legacy systems.

HTML vs. PDF

The standard underlying delivery mechanism of the Internet and the World Wide Web is HTML documents delivered via HTTP. HTML (Hyper-text Markup Language) files are essentially simple text files, *marked up* with formatting commands which appear alongside the text. The problem is that HTML focuses primarily on maintaining document *content* and not the exact look-and-feel of the document.

The challenge for Oracle Insurance is to deliver a standardized solution in an area that is Oracle Insurance's strength—reproducing the exact look-and-feel of a document set, not just the content, across multiple platforms.

In addition, another challenge is to provide a solution that supports the growing body of *thin client* workstations attached to the Internet, requiring only a minimum of end-user software.

To meet these and other challenges, Oracle Insurance stores and creates files in Portable Document Format (PDF). the PDF file fomat is the industry standard, providing a searchable, open format that maintains the look-and-feel of the original documents and works across a variety of platforms.

In addition, using Adobe Software's Acrobat Reader, a free application anyone can download from the Internet, thin-client end users can easily view and print complete document sets which are identical to the original documents.

NOTE: You can read more about Adobe's Portable Document Format and download the Adobe Acrobat Reader from Adobe's web site at www.adobe.com.

Architectural Changes in Version 2.x

The core architecture of Internet Document Server changed in version 2.0 to allow major enhancements of current and future functionality. The single-tasking architecture of Internet Document Server version 1.8 was replaced with a multi-tasking one that can handle several tasks at once, allowing greater throughput and fewer pauses. Tasks now handled concurrently include

- Handling of certain types of requests. Rules for requests that are written in a thread-safe manner can be run at the same time in one instance of IDS.
- Purging of cached files that have expired. When processing requests, IDS can produce temporary files, which are given a length of time to exist before they are automatically deleted. IDS version 1.8 had to stop processing requests to periodically purge these files; IDS version 2.***x*** does not pause request processing to purge files.
- Receiving requests from a messaging queue.
- Sending results back to a different messaging queue.
- Handling multiple incoming and outgoing HTTP requests.
- Watching the running rules to see if they are taking too long.
- Looking for changes to the configuration file and restart IDS.
- Looking for changes to logging configuration and incorporate changes without restarting IDS.

In addition to using HTTP as a transport of SOAP messages, IDS can respond to requests formatted as a URL from a browser and display results in HTML. The XML result produced by IDS is transformed by XSLT templates into HTML; there can be a different XSLT template for each request, or a default will be used.

In addition to HTTP, WebSphere MQ and MSMQ, IDS version 2.***x*** can use Java Messaging Service (JMS) queues for sending and receiving messages. JMS is a standard for messaging used by J2EE application servers, such as WebSphere, WebLogic and JBoss.

You can configure IDS version 2.***x*** to handle requests from both message queues and from HTTP in the same instance; version 1.8 could only do one or the other.

The format of requests coming in to IDS is configurable and extensible. If a third-party application wants to send requests in formats other than SOAP, custom translators can be installed in IDS. When IDS receives a request it will recognize the format and the result will be sent back in the same format as it was received.

IDS can be monitored by the Simple Network Management Protocol (SNMP). This is the standard protocol used by manufacturers of networking hardware (such as routers), printers and computers to monitor uptime and usage statistics. IDS appears as another piece of equipment to SNMP monitoring applications.

Version 2.***x*** also enhances IDS's error logging and tracing capabilities. Logging messages can be assigned a severity level (DEBUG, INFO, WARN, ERROR or FATAL) and logging messages can be routed to multiple destinations including the console screen, files, the Windows event logger, the UNIX syslog daemon, and email. Logging options can be changed without restarting IDS, making the diagnosing of problems easier.

Details on these features can be found in this manual and in the SDK Reference.

Required Components

To use Oracle Insurance's Internet document processing solution, you need several components. Some components are included in Oracle Insurance's Internet Document Server, while others are included in the various bridges. Other required components must be provided by the end-user or the license-holder. The basic components are:

Provided by the end-user

Web Browser. An end-user workstation must have a working web browser that supports Adobe Acrobat Reader version 7.0 or higher. Oracle Insurance has tested successfully with Microsoft Internet Explorer 6.0 and higher. To download a copy of Adobe Acrobat Reader, go to:

http://www.adobe.com

Internet Access. An installed and working Internet (or intranet) connection, including the necessary hardware and software, Internet provider account, modem, and so on. The access should provide acceptable performance when downloading large files.

Java Runtime Environment. Java and the Java runtime environment (JRE) are only required if you are using Java rules or a Java client. A Java Runtime Environment (JRE) or the Java Software Development Kit (JDK), version 1.5 ('Java 5') is recquired. You can download a free runtime environment at:

http://java.sun.com

To run certain rules, the Java Cryptography Extension is required. It is included in Java runtimes version 1.5 and later.

Provided by the Oracle Insurance license-holder

Server. An installed and working server, such as Microsoft Windows (2000 or XP).

Web Server. An installed and working web server, such as a web server with Windows 2000 Server or Windows 2003 Server and Microsoft Internet Information Server 5.0 (or higher). While you can use Windows XP for development and testing purposes, do not use it as a production web server.

Note: All of these components should be installed and working before you install IDS. In general, end-users will be supported and trained on these applications by their own experienced in-house Internet support group. Contact your Oracle Insurance sales representative to inquire about the services and consulting packages Oracle Insurance offers.

Components Available from Oracle Insurance

Internet Document Server

This component includes:

- Internet Document Server
- Service administration rules and supporting HTML templates
- Example test rules and HTML templates
- Sample programs written in Java and C++
- Sample programs, rules and ActiveX components written in Microsoft Visual Basic. (Windows only)

- Sample Microsoft Active Server Page (Windows only)
- Sample Java Server Page programs
- Documentation (see Using the Internet Document Server, beginning on page 9)

Documaker Bridge

This optional component includes:

- The bridge PDF generator
- Base bridge rules for archive/retrieve/HTML templates
- Subset Documaker runtime to support archive/retrieve rules
- Documentation (see the separate manual entitled, Using the Documaker Bridge)

Printstream Bridge (only for Windows)

This optional component includes:

- The bridge PDF generator
- Rules to support Documerge Metacode and AFP output and archive/retrieve/HTML templates
- Sample HTML templates
- Utility for creating logo files
- Utilities for creating and checking fonts
- Documentation (see the separate manual entitled, Using the Printstream Bridge)

Documanage Bridge (only for Windows)

This optional component includes:

- The bridge PDF generator
- Rules to support Documanage output archive/retrieve/HTML templates
- Sample archive/retrieve/HTML templates
- Documentation (see the separate Documanage manuals entitled, *General Reference for the Documanage Bridge* and *Rules Reference for the Doucmanage Bridge*)

Docuflex Bridge

This optional component includes:

- An IDS rule DLL (DFLXRULE.DLL)
- Documentation (see the separate manual entitled, Using the Docuflex Bridge)

PDF print driver

This component includes tools which let you convert output from Documaker's GenPrint program into PDF files, which can be viewed using an Internet browser. For more information, see Creating PDF Files, beginning on page 208.

HTML print driver

This component lets you create HTML files by simply printing to the HTML print driver. For more information, see Creating HTML Files on page 230.

DSI SDK Package

This component includes:

- Software Developer Kit for the various bridges
- Source code to archive retrieval rules
- API documentation and technical reference for writing custom rules, Visual Basic programs, Active X components and ASP components (see *Using the Internet Document Server SDK* in the SDK Reference).

NOTE: The sample dialog templates use standard HTML language features found in HTML version 2.0 and above. There are many ways you can customize these templates to meet your company's needs. The HTML language is well documented in many commercially available books, and is supported by many commercially available design tools. The HTML templates provided are designed to be able to be maintained with most commercial HTML editors. Oracle Insurance can provide services to customers needing assistance with HTML template customization.

Users and customers need to be aware that due to the varying nature of browsers and their continuously changing levels of support for the evolving HTML language, the use of certain HTML tags in the templates and dialogs might limit the ability of users to display certain aspects of the customized pages.

Chapter 2

Using the Internet Document Server

This chapter provides information on the capabilities of the Internet Document Server and its architecture. This chapter also tells you how to set up the Internet Document Server.

NOTE: For information on installing IDS, see the IDS Installation Guide.

You'll find this information:

- Overview on page 11
- Using Multiple Servers on page 50
- Setting Up a Windows NT Service on page 54
- Handling Multi-threaded Requests on page 55
- Using Rules Written in Other Scripting Languages on page 58
- Using IDS as a Client to Another IDS on page 59
- Monitoring IDS with SNMP Tools on page 64
- Managing IDS Instances on page 66
- Sending Results and Receiving Requests in Multiple Formats on page 75
- Logging and Tracing on page 78
- Configuring IDS on page 97
- Referencing Attachment Variables on page 106
- Using the Message Queues on page 108
- Using the Java Message Service (JMS) on page 113

- Using WebSphere MQ on page 116
- Using HTTP on page 126
- Using Multiple Bridges on page 130
- Submitting Batch Requests on page 132
- Printing in Duplex Mode to PCL Printers on page 134
- Using IDS to Distribute Email on page 135
- Using IDS to Run Documaker on page 144
- Using the XML Messaging System on page 158
- Connecting to an SQL Database on page 166
- Using the Thin Client Forms Publisher on page 171
- Pausing IDS on page 172
- Executing Request Types at Run Time on page 175
- Publishing Your Forms on the Web on page 177
- Handling Multi-Part/Form-Data Forms on page 182
- Formatting Text with XML Markup on page 183
- Encrypting and Decrypting Data Files on page 184
- Using Multiple Attachment Values with the Same Name on page 185
- Converting XML Files Using a Template on page 188
- Customizing Your System on page 192
- Handling Security Issues on page 195
- Using the FAP2XML Utility on page 197
- Using LDAP Support on page 198
- Using Default Time-outs for DSILIB-Based Client Applications on page 199
- Running Timed Requests on page 201
- In-Process Rendering for DPAView on page 202
- Using DAL Functions for WIP Column Access on page 203
- Using Enterprise Web Processing Services on page 205

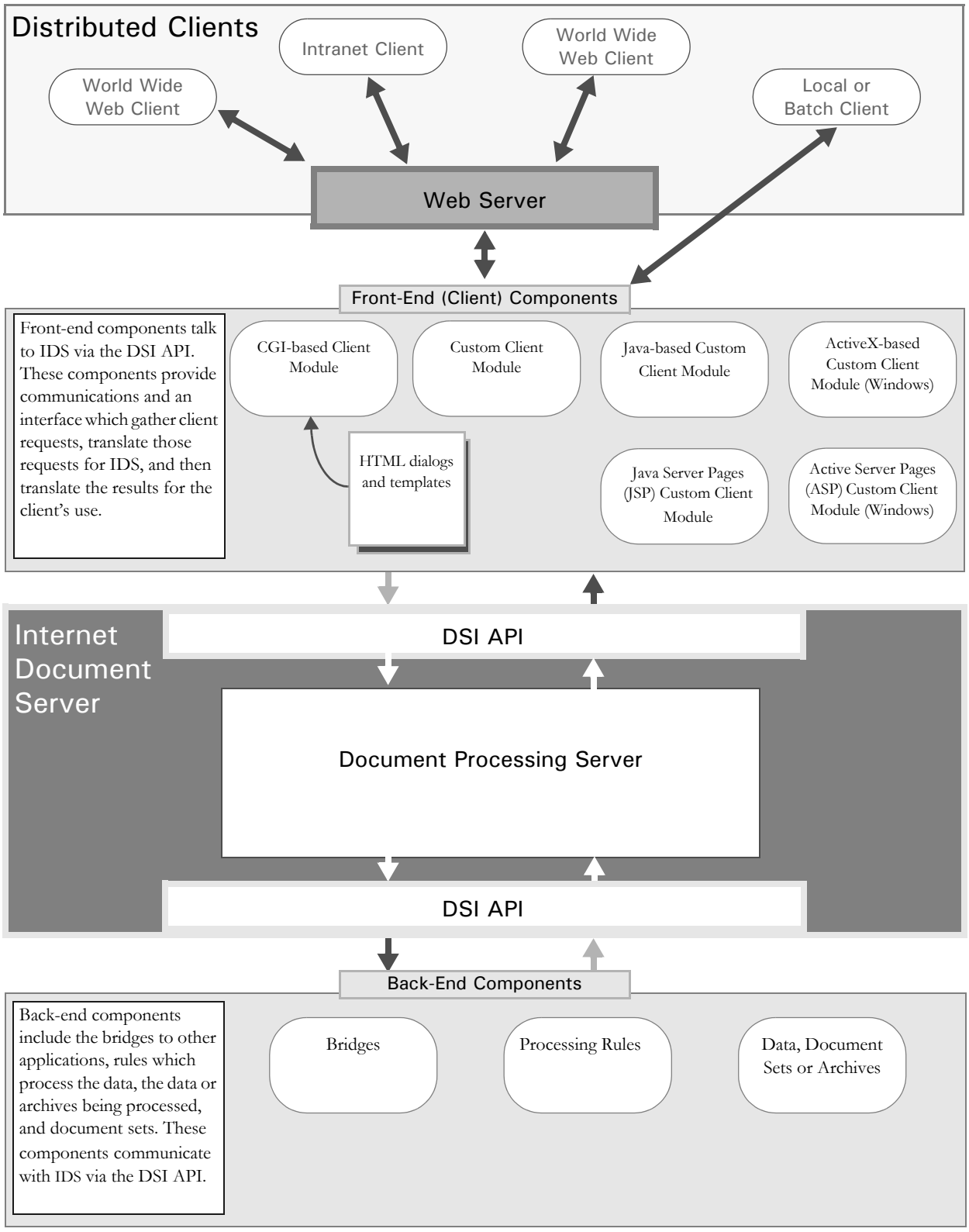
OVERVIEW

The Internet Document Server lets users connect to the server via the Internet. Executing back-end applications, however, requires additional components. These additional components are called *bridges*. These bridges provide bridge components, software rules, document templates, and other files necessary to process documents.

The Internet Document Server lets users communicate via standard Internet methods using a standard web browser. No other specialized client software is required to use the Internet Document Server; however, additional bridge components may require additional browser plug-ins, such as Adobe's Acrobat Reader. The Internet Document Server runs on a Microsoft Windows or Sun Solaris server running a web server package.

NOTE: See Processing Documents Using the Internet on page 1 for more information on Oracle Insurance's related products.

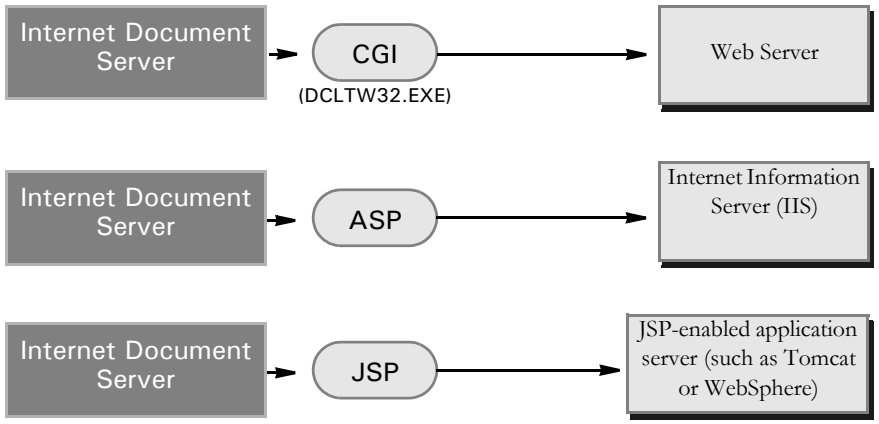
The following diagram shows you how the system operates.


Distributed Clients
Intranet Client
World Wide Web Client
World Wide Web Client
Local or Batch Client
Web Server
Front-End (Client) Components
Front-end components talk to IDS via the DSI API. These components provide communications and an interface which gather client requests, translate those requests for IDS, and then translate the results for the client's use.
CGI-based Client Module
Custom Client Module
Java-based Custom Client Module
ActiveX-based Custom Client Module (Windows)
HTML dialogs and templates
Java Server Pages (JSP) Custom Client Module
Active Server Pages (ASP) Custom Client Module (Windows)
Internet Document Server
DSI API
Document Processing Server
DSI API
Back-End Components
Back-end components include the bridges to other applications, rules which process the data, the data or archives being processed, and document sets. These components communicate with IDS via the DSI API.
Bridges
Processing Rules
Data, Document Sets or Archives

CREATING FRONT-END SOLUTIONS

You can use either JSP (Java server pages), ASP (active server page) or CGI (common gateway interface) to create front-end solutions, as shown in the previous illustration. While CGI is good for some situations, to handle a high volume of transactions, JSP or ASP is a more efficient choice.

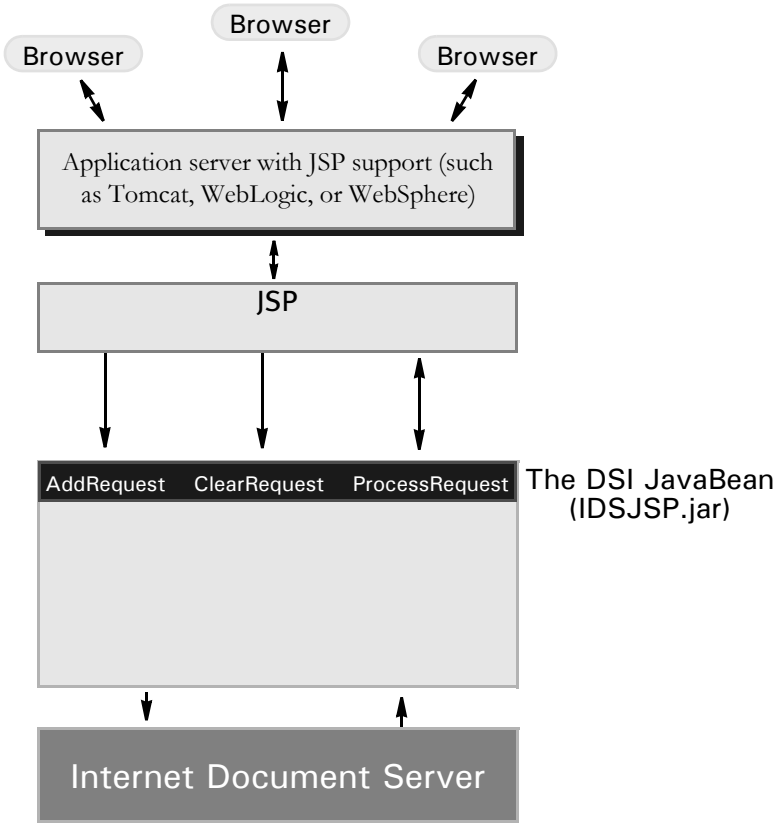
NOTE: Keep in mind ASP is a Microsoft product and is not available on Solaris.



USING JSP

With JSP, the HTML content is included in the JSP page. There is no separate template.

To help you more quickly create your front end solutions, we provide the DSI JavaBean to communicate with Internet Document Server. This bean is in IDSJSP.jar.



USING THE IDSJSP JAVABEAN

As shown in this illustration, the IDSJSP bean collects requests and results. The bean has the following properties and methods.

Properties

Property	Description
waittime	The time in between tries for ProcessRequests.
timeout	The total time to wait for the ProcessRequest.

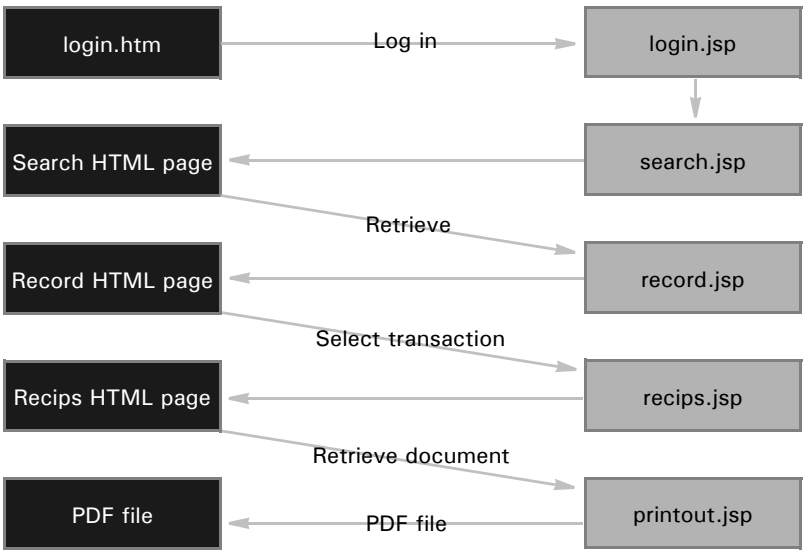
Methods

Method	Description
AddRequest	AddRequest(Object key, Object value) Adds name/value fields to the record to send to the IDS rule.
AddAllRequest	AddAllRequest(javax.servlet.ServletResponse request) Adds all name/value fields from the request objects to the records to send to the IDS rule.

Method	Description
ProcessRequest	ProcessRequest() Sends all the name/value and request types to IDS rules. Processes the IDS rule and gets return records from the IDS rule and returns them as type HashMap.
GetResult	GetResult(Object key) Gets the return record value from the IDS rule index using the key from the internal result.
ClearRequest	ClearRequest() Clears attachment variables out of the request. Use after ProcessRequest and before the next set of AddRequest calls.
ClearResult	ClearResult() Clears the result.

USING THE JSP SAMPLES

The system includes several JSP files which you can use as a guide when building your front-end solutions. You can also make copies of these JSP files and modify them to meet your needs. This illustration shows how these pages work together:



NOTE: The search, record, and recips HTML pages are dynamically generated.

Handling Multi-Part/Form-Data Forms

JSP pages can process HTML forms encoded as *multi-part/form-data*. This encoding enables file uploading from HTML pages to IDS for processing.

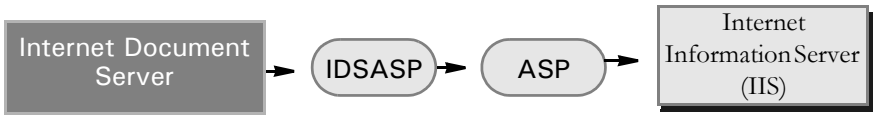
Here is a sample HTML form that includes uploading a file. The file is sent in the IMAFILE attachment named in binary format along with the usual attachment variables for a LGN request.

```
<FORM METHOD=POST ENCTYPE="multipart/form-data" ACTION="login.jsp">
  <INPUT TYPE="FILE" NAME="IMAFILE"/><BR>
  <INPUT NAME="REQTYPE" value="LGN" TYPE="HIDDEN"> <BR><BR><BR><BR>
  <FONT FACE="Arial,Lucinda Sans,Gill Sans" COLOR="#0000FF">
  <H3>
  <B>User ID: </B> <INPUT SIZE=10 MAXLENGTH=8 NAME="USERID"
value="FORMAKER"> <BR><BR>
  <B>Password: </B> <INPUT TYPE=PASSWORD SIZE=8 MAXLENGTH=8
NAME="PASSWORD" VALUE="FORMAKER"><P>
  <B>Archive: </B>
  <SELECT NAME="CONFIG">
   <OPTION VALUE = "UTILITY">Utility Company
   <OPTION VALUE = "FINANCE">Worldwide Financial
  </SELECT>
  <P>
  <INPUT TYPE="submit" VALUE="Login">   <INPUT
TYPE="reset" VALUE="Reset"><P>
  </H3>
  </FONT>
  <BR><BR>
  </FORM>
```

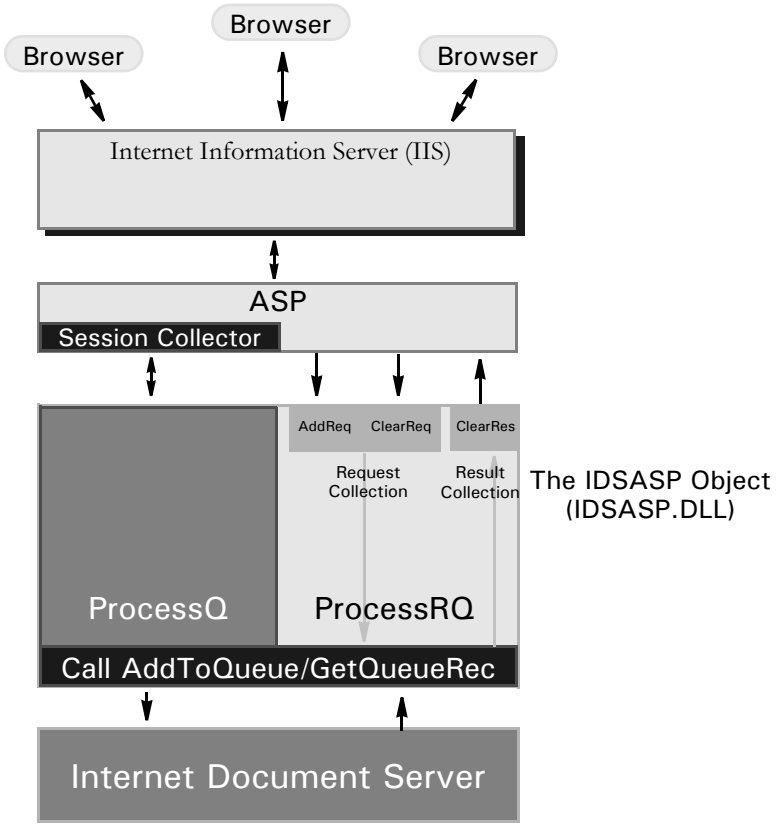
Using ASP

With ASP, the HTML page is included with the ASP page. There is no separate template. Furthermore, the processing is done in each ASP page instead of in DCLTW32.EXE. On the other hand, ASP also requires more coding effort.

To help you more quickly create your front end solutions, we created IDSASP (IDSASP.DLL).



With IDSASP, you can use the ASP Session Collector and ProcessQ or you can bypass the Session Collector using ProcessRQ, as shown in the following illustration.



Using the IDSASP Object

As shown in this illustration, the IDSASP object collects requests and results. To communicate with IDS using ASP through DSIAPI, the system includes an ActiveX DSO named *IDSASP.DLL*. This DSO has the following properties and methods.

Properties

Property	Description
hInstance	Type: Long DSI Instance Handle is created by InitSession() in IDSASP.
oDSI	Type: DSICoAPI DSI Handle.
Request	Type: Collection Request Collection in IDSASP.
Result	Type: Collection Result Collection in IDSASP.
ShowAtt	Type: Boolean When set to True, the system prints the request and result when you call the ProcessQ or ProcessRq method. Use this property for debugging.

Property	Description
WaitTime	Type: Long Retry time in the processing queue. The default is 1000 milliseconds.
TimeOut	Type: Long Time-out in milliseconds. The default is 15000 milliseconds.

Methods

Method	Description
AddReq	Syntax: AddReq (ByVal Name As String, ByVal Value As String) Call this method to add a request to IDSASP request collection.
ClearReq	Syntax: ClearReq() Call this method to clear the IDSASP request collection.
ClearRes	Syntax: ClearRes() Call this method to clear the IDSASP result collection.
OnEndPage	Called by ASP when the object is instantiated.
OnStartPage	Called by ASP when the object is released.
ProcessQ	Syntax: ProcessQ() Call this method to... - Retrieve the name/value pair from the ASP session collection and add it to the queue. - Release the record into the queue for IDS to process. - Read back the result from IDS. Retry time and time-out are defined in the WaitTime and TimeOut properties. - Store the results in the ASP session collection.
ProcessRq	Syntax: ProcessRq() Call this method to... - Retrieve the name/value pair from the IDSASP request collection and add it to the queue. - Release the record into the queue for IDS to process. - Read back the result from IDS. Retry time and time-out are defined in the WaitTime and TimeOut properties. - Store the results in the IDSASP result collection.
ReadBinFile	Syntax: ReadBinFile(ByVal bFileName As String) Call this method to read the binary file from the local hard disk and store as a binary array.

For both ASP and the DCLTW32.EXE file, fields are passed between the browser and ASP via a CGI query string, such as:

```
http://docucorp/asp/utility/search.asp?USERID=USERID&CONFIG=UTILITY
```

or in a form with the POST or GET method.

The differences between using ASP or the CGI-based client (DCLTW32.EXE) are:

- The HTML page is included with the ASP page instead of a separate HTML template
- Processing is done in each ASP page instead of only in the DCLTW32.EXE file

Sending and Receiving Attachment Fields

The IDSASP.DLL provides two ways to send and receive attachment fields to and from DSIAPI: using the ProcessQ method and using the ProcessRq method.

Using the ProcessQ method

Here is an example of using session collection with the ProcessQ method:

```
<%
    CrLf = Chr(13) + Chr(10)
    set DSI = Server.CreateObject("IDSASP.DSI")'create DSI handle
    session.Abandon            'Clear Session

    for i=1 to Request.Form.Count'Read Attachment and Create session
Collection
        session(Request.Form.Key(i))=Request.Form(i)
    next
    DSI.ProcessQ       'Execute Request From Attachment
%>
<%
    'Loop To display all of the records.
    For i=1 to session("RECORDS")
      alink = "recips.asp"
      alink = alink & "?USERID=" &
Server.URLEncode(session("USERID"))
     alink = alink & "&ArcKey=" & Server.URLEncode(session("RECORDS"
& cstr(i) & ".ArcKey"))
      alink = alink & "&REQTYPE=RCP"
      alink = alink & "&CONFIG=" &
Server.URLEncode(session("CONFIG"))
      alink = alink & "&COMPANY=" &
Server.URLEncode(session("RECORDS" & i & ".Company"))
     alink = alink & "&LOB=" & Server.URLEncode(session("RECORDS" &
cstr(i) & ".Lob"))
      alink = alink & "&POLICYNUM=" &
Server.URLEncode(session("RECORDS" & cstr(i) & ".PolicyNum"))
      alink = alink & "&RUNDATE=" &
Server.URLEncode(session("RECORDS" & cstr(i) & ".RunDate"))

      Response.Write "<TR><TD><B>"
      Response.Write "<A HREF=" & alink & ">" & session("RECORDS" &
i & ".Company") & "</A></TD>"
      Response.Write "<TD>" & session("RECORDS" & cstr(i) & ".Lob")
& "</TD>"
      Response.Write "<TD>" & session("RECORDS" & cstr(i) &
".PolicyNum") & "</TD>"
```

```
      Response.Write "<TD>" & session("RECORDS" & cstr(i) &
".RunDate") & "</TD>"
      Response.Write "</TR>" & CrLf
    next
%>
```

Using the ProcessRq method

Here is an example of using the IDSASP.DLL request and result collection properties with the ProcessRq method.

```
<%
    CrLf = Chr(13) + Chr(10)
    set DSI = Server.CreateObject("IDSASP.DSI")'create DSI handle
    'Read Input Parameter from INPUT FORM and send request to DSI
    for i=1 to Request.Form.Count
        DSI.AddReq Request.Form.Key(i),Request.Form(i)
    next
    DSI.ProcessRq  'Execute Request
%>
<%
    'Loop display Response Pair
    for i=1 to DSI.Result("RECORDS").Value
      alink = "recips.asp"
      alink = alink & "?USERID=" &
Server.URLEncode(DSI.Result("USERID").value)
      alink = alink & "&ArcKey=" &
Server.URLEncode(DSI.Result.Item("RECORDS" & cstr(i) &
".ArcKey").value)
      alink = alink & "&REQTYPE=RCP"
      alink = alink & "&CONFIG=" &
Server.URLEncode(DSI.Result("CONFIG").value)
      alink = alink & "&COMPANY=" &
Server.URLEncode(DSI.Result("RECORDS" & i & ".Company").value)
      alink = alink & "&LOB=" &
Server.URLEncode(DSI.Result.Item("RECORDS" & cstr(i) &
".Lob").value)
      alink = alink & "&POLICYNUM=" &
Server.URLEncode(DSI.Result.Item("RECORDS" & cstr(i) &
".PolicyNum").value)
      alink = alink & "&RUNDATE=" &
Server.URLEncode(DSI.Result.Item("RECORDS" & cstr(i) &
".RunDate").value)
      Response.Write "<TR><TD><B>"
      Response.Write "<A HREF=" & alink & ">" & DSI.Result("RECORDS"
& i & ".Company").value & "</A></TD>"
      Response.Write "<TD>" & DSI.Result.Item("RECORDS" & cstr(i) &
".Lob").value & "</TD>"
      Response.Write "<TD>" & DSI.Result.Item("RECORDS" & cstr(i) &
".PolicyNum").value & "</TD>"


      Mh=Left(DSI.Result.Item("RECORDS" & cstr(i) &
".RunDate").value,2)
      Dt=Mid(DSI.Result.Item("RECORDS" & cstr(i) &
".RunDate").value,3,2)
      Yr=Right(DSI.Result.Item("RECORDS" & cstr(i) &
".RunDate").value,2)
      Dat=Cdate(Mh & "/" & Dt & "/" & Yr)

      Response.Write "<TD>" & FormatDateTime(Dat,1) & "</TD>"
      Response.Write "</TR>" & CrLf
```

```
        next
    %>
```

Sample Pages

Here are some sample pages:

Page 1

This page sends a request from the browser with two file attachments.

```
<form name="form" enctype="multipart/form-data" action="test.asp" method="post">
<table>
<tr><input name="key1" value="12345678" /></tr>
<tr><input name="key2" value="456" /></tr>
<tr><input name="key3" value="789"/></tr>
<tr><input name="empty" value=""/></tr>
<tr><input name="file1" type="file"/></tr>
<tr><input name="file2" type="file"/></tr>
<tr><input name="submit" type="submit"/></tr>
```

Page 2

This page receives an HTTP request from page 1, parses the request, and uploads the files.

```
<%
    'create an instance of the object which calls parseData
    set o = server.CreateObject("IDSASP.DSI")
    'o.bDebug = true
    o.parseData()

    'write the element count in the request collection
    response.write "count=" & o.request.count & "<BR><BR>"

    'indicate if the request is a multipart request
    response.write "Multipart=" & o.bMultipart & "<BR>"

    'taverse through the request collection and write the name / value pairs
    for i = 1 to o.request.count
        name = o.request.Item(i).Name
        value = o.getRequest(name)
        response.write "(" & name & ") = (" & value & ")<br>"
    next

    'if the request is a multipart request, then process the attachments
    if o.bMultipart = true then
        for each attachment in o.attachments
            if IsObject(attachment) then
                name = attachment.name
                response.write "attachment name=" &name & "<BR>"
                file = attachment.file
                response.write "file name=" & file & "<BR>"
                ftype = attachment.ftype
                response.write "file type=" & ftype & "<BR>"
                encoding = attachment.encoding
                response.write "encoding=" & encoding & "<BR>"
                buffer = attachment.buffer
                response.write "buffer length=" & Len(buffer) & "<BR>"
                'write attachment to disk
                path = o.upLoad(name, "c:\inetpub")
                response.write "path returned by upLoad api = " & path & "<BR>"
```

Showing a PDF File

ASP provides two ways to show a PDF file: using the Response.Redirect method and using the Read ReadBinFile method.

Using the Response.Redirect method

Here is an example of showing a PDF file using the Response.Redirect method. You must enter a URL.

```
<%@ Language=VBScript %>
<%
    Set DSI = Server.CreateObject("IDSASP.DSI")'create DSI handle
    session.Abandon        'Clear Session
    For i=1 to Request.Form.Count'Read Attachment and Create session
Collection
        session(Request.Form.Key(i))=Request.Form(i)
    Next
    DSI.ProcessQ   'Execute Request From Attachment
```

```
    HostAddr=Request.ServerVariables("HTTP_HOST")'Get Host Name
    PrintFile=session("REMOTEPRINTFILE")'Get Full Printed Filename
with Path
    StartPoint=instr(1,PrintFile,"\")'Look for \ sign
    NameWidth=len(PrintFile)-StartPoint'Filename Length
    FileName=Mid(PrintFile,StartPoint+1,NameWidth)'Get Filename
    Url="http://" & HostAddr & "/doc-html/" & Filename'Construct
URL
    Set DSI = nothing
    If instr(1,Request.ServerVariables("HTTP_USER_AGENT"),"IE")<>0
then
    'Check IE Browser
%>
<HTML>
<BODY leftmargin=0 topmargin=0 scroll=no>
    <embed width=100% height=100% fullscreen=yes src="<%=Url%>">
</BODY>
</HTML>
<%
    Else
        Response.Redirect Url
    End If
%>
```

Using the Read ReadBinFile method

Here is an example of showing a PDF file using the Read ReadBinFile method in the IDSASP.DLL file, you must enter the local path.

```
<%@ Language=VBScript %>
<%
    Dim Stream
    set DSI = Server.CreateObject("IDSASP.DSI")'create DSI handle
    'Attach Input Parameter from INPUT FORM
    for i=1 to Request.Form.Count
        DSI.AddReq Request.Form.Key(i),Request.Form(i)
    next
    DSI.ProcessRq  'Send Queue to DSI
%>
<%
    Response.Buffer=True
    Pth=Request.ServerVariables("PATH_TRANSLATED")
    DsiPath=left(pth,instr(4,pth,"\"))'Get Path of Docserv
    PrintFile=DsiPath & DSI.Result("REMOTEPRINTFILE").Value
        Response.ContentType = "application/pdf"
        Stream = DSI.ReadBinFile(PrintFile)
    Response.BinaryWrite(Stream)
    Response.End

    set DSI = nothing
%>
```

Using the ASP Samples

IDS includes several ASP files which you can use as a guide when building your own front-end solutions. You can also make copies of these ASP files and modify them to meet your needs. Keep in mind that all samples use the...

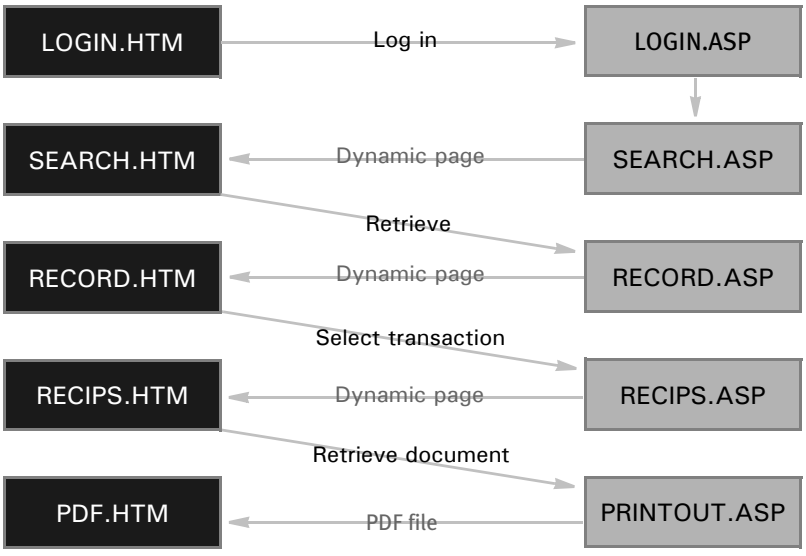
- ASP session collection method except for the World Wide Combo (Insure) sample which uses the Request/Result collection in the IDSASP.DLL.
- Response.Redirect Method to view PDF except for the World Wide Combo (Insure) sample which uses the ReadBinFile method.

NOTE: We use Chili!Soft to work with third-party vendors. For more information, visit their web site: www.chilisoft.com.

The sample ASP files include:

- LOGIN.ASP
- SEARCH.ASP
- RECORD.ASP
- RECIPS.ASP
- PRINTOUT.ASP

This illustration shows how these pages work together:



For more information about Active Server Pages, consult your Microsoft documentation.

Using the HTTP Parsing and Uploading APIs

The following APIs in IDSASP let you parse HTTP requests into separate request and attachments collections and provide a way to process multipart/form-data form requests.

- parseData
- getRequest
- getAttachment
- getBuffer
- upLoad

Multiple file attachments are parsed into attachments collections in IDSASP. Files can then be uploaded (written to disk) to the web server via the upLoad API.

Attachment objects can also be retrieved from the attachments collection via the getAttachment API. Attachment objects contain properties for each attachment as well as an attachment buffer that contains the actual file attachment contents.

You can also retrieve file attachment contents as buffers from the attachments collection via the getBuffer API. The parseData API can also parse non multi-part/form-data HTTP requests. Use the getRequest API to retrieve name/value pairs from the request collection.

In addition, you can also see Sample Pages on page 21.

parseData

Use this API to parse HTTP requests into separate request and attachments collections. Regular name/value pairs in an HTTP request are parsed into request collection. File attachments are parsed into an attachments collection. The parseData API can parse multipart/form-data HTTP requests as well as non multipart/form-data requests. Call this API at the beginning of an ASP script to parse the HTTP request from a submitted form.

Parameters None

Returns Nothing

getRequest

Use this API to retrieve name/value pairs from the request collection instead of the Request.Form API calls.

Parameters

Parameter	Description
name	A string value that represents the name of a key in the request collection.

Returns A string value with the value in request collection for key name.

getAttachment

Use this API to return an attachment object from the attachments collection. This API retrieves not only the file attachment contents, but also its properties. The attachment object returned contains these properties:

Property	Description
name	This is the actual name of the attachment in the HTTP request. Its value corresponds to the value of the file form tag used to submit the attachment in the HTTP request.
File	A string value that contains the file name and extension of the attachment. Its value corresponds to the file name of the File form tag used to submit the attachment.
Ftype	A string value that contains the extension (file type) of the attachment.
Buffer	A string buffer holding the file attachment contents.
Encoding	The actual encoding type used by the browser when the file attachment was submitted.

Parameters

Parameter	Description
name	A string value that represents the name of a key in the attachments collection.

Returns

An attachment object containing the file attachment contents as well as properties for the attachment.

getBuffer

Use this API to retrieve attachments as buffers.

Parameters

Parameter	Description
name	A string value that represents the name of the attachment in the Attachment object within the attachments collection. This value should be the same as that of the file form tag name used to send an attachment in an HTTP request.

Returns

A string buffer containing the contents of the file attachment in the attachments collection.

upLoad

Use this API to write an attachment object's buffer contents from the attachments collection to disk. This API lets you upload file attachments to disk on the web server.

Parameters

Parameter	Description
name	A string value that represents the name of the attachment in the attachment object within the attachments collection. This name is the same as the file form tag used to send the attachment from the browser in an HTTP request.
Path	A string value that specifies the full path where you want the attachment written.

Returns

The full path and file name of the file uploaded, if successful.

Sample Pages

Here are some sample pages:

Page 1

This page sends a request from the browser with two file attachments.

```
<form name="form" enctype="multipart/form-data" action="test.asp" method="post">
<table>
<tr><input name="key1" value="12345678" /></tr>
<tr><input name="key2" value="456" /></tr>
<tr><input name="key3" value="789"/></tr>
<tr><input name="empty" value=""/></tr>
<tr><input name="file1" type="file"/></tr>
<tr><input name="file2" type="file"/></tr>
<tr><input name="submit" type="submit"/></tr>
```

Page 2 This page receives an HTTP request from page 1, parses the request, and uploads the files.

```
<%
    'create an instance of the object which calls parseData
    set o = server.CreateObject("IDSASP.DSI")
    'o.bDebug = true
    o.parseData()

    'write the element count in the request collection
    response.write "count=" & o.request.count & "<BR><BR>"

    'indicate if the request is a multipart request
    response.write "Multipart=" & o.bMultipart & "<BR>"

    'taverse through the request collection and write the name / value pairs
    for i = 1 to o.request.count
        name = o.request.Item(i).Name
        value = o.getRequest(name)
        response.write "(" & name & ") = (" & value & ")<br>"
    next

    'if the request is a multipart request, then process the attachments
    if o.bMultipart = true then
        for each attachment in o.attachments
            if IsObject(attachment) then
                name = attachment.name
                response.write "attachment name=" &name & "<BR>"
                file = attachment.file
                response.write "file name=" & file & "<BR>"
                ftype = attachment.ftype
                response.write "file type=" & ftype & "<BR>"
                encoding = attachment.encoding
                response.write "encoding=" & encoding & "<BR>"
                buffer = attachment.buffer
                response.write "buffer length=" & Len(buffer) & "<BR>"
                'write attachment to disk
                path = o.upLoad(name, "c:\inetpub")
                response.write "path returned by upLoad api = " & path & "<BR>"
```

Using the XMLSession Rules

Use the XMLSession rules to save state information across multiple IDS servers and across multiple web servers. Session information for each client session is saved as an XML file on the server side. This also increases security as session information no longer resides on the web server.

You can store, retrieve, and save files using the XMLSession rules. You can retrieve the session information as rowset on the client side and it is also accessible using session methods available in IDSASP and IDSJSP.

For more information about these methods and rules, see

- IDSASP Methods on page 29
- IDSJSP Methods on page 31
- XMLSession Rules on page 33

IDSASP Methods

Here are the IDSASP methods:

addSessionVar

Use this method to add a name/value pair to the session collection. You can include these parameters:

Parameter	Description
name	The name of the name/value pair to add to the session.
Value	The value of the name/value pair to add to the session.

Here is an example:

```
dsi.addSessionVar "USERID", "FORMAKER"
```

getSessionVar

Use this method to return a string containing the value of name in the session collection. You can include these parameters:

Parameter	Description
name	The name of the name/value pair to retrieve from the session.

Here is an example:

```
userid = dsi.getSessionVar("USERID")
```

removeSessionVar

Use this method to remove a name/value pair from the session collection. You can include these parameters:

Parameter	Description
name	The name of the name/value pair to remove from the session.

Here is an example:

```
dsi.removeSessionVar "USERID"
```

addSessionObject

Use this method to add a binary or text buffer to hold the contents for a file to the session collection. You can include these parameters:

Parameter	Description
name	The name of the object name to add to the session.
Buffer	A binary or text buffer holding the contents of a file to add to the session.

Here is an example:

```
dsi.addSessionObject "FILE1", buffer
```

getSessionObject

Use this method to retrieve a buffer that holds the contents of a file from the session collection. You can include these parameters:

Parameter	Description
name	The name of the object to retrieve from the session.
Opt	An integer value that indicates whether the object should be retrieved as a binary or text buffer (1=text, 2=binary)

Here is an example:

```
buffer = dsi.getSessionObject("FILE1", 1)
binBuf = dsi.getSessionObject("FILE2", 2)
```

removeSessionObject

Use this method to remove an object from the session collection. You can include these parameters:

Parameter	Description
name	The name of the object to remove from the session.

Here is an example:

```
dsi.removeSessionObject "FILE1"
```

IDSJSP Methods

Here are the IDSJSP methods:

addSessionVar

Use this method to add a name/value pair to the session map. You can include these parameters:

Parameter	Description
name	Enter a string that contains the name of the name/value pair to add to the session map.
Value	Enter a string that contains the value of the name/value pair to add to the session map.

This method has no return value. Here is an example:

```
dsimsg.addSessionVar("USERID", "FORMAKER");
```

removeSessionVar

Use this method to remove a name/value pair from the session map. You can include these parameters:

Parameter	Description
name	Enter a string that contains the name of the name/value pair to remove from the session map.

This method has no return value. Here is an example:

```
dsimsg.removeSessionVar("USERID");
```

getSessionVar

Use this method to retrieve a name/value pair from the session map. You can include these parameters:

Parameter	Description
name	Enter a string that contains the name of the name/value pair to retrieve from the session map.

This method returns a string containing the value of the name/value pair in the session map. Here is an example:

```
String userid = dsimsg.getSessionVar("USERID");
```

addSessionObject

Use this method to add a file buffer to the to the session map. You can include these parameters:

Parameter	Description
name	Enter a string that contains the name of the file buffer to add to the session map.
Buffer	A byte array holding the contents of a file.

This method has no return value. Here is an example:

```
dsimsg.addSessionObject("FILE1", buffer1);
```

removeSessionObject

Use this method to remove a file buffer from the session map. You can include these parameters:

Parameter	Description
name	Enter a string that contains the name of the file buffer to remove from the session map.

This method has no return value. Here is an example:

```
dsimsg.removeSessionObject("FILE1");
```

getSessionObject

Use this method to retrieve a file buffer from the session map. You can include these parameters:

Parameter	Description
name	Enter a string that contains the name of the file buffer to retrieve from the session map.

This method returns a byte[] array containing the buffer retrieved from the session map. Here is an example:

```
byte[] buff = dsimsg.getSessionObject("FILE1");
```

XMLSession Rules

Here are the XMLSession rules:

initSession

Use this rule to initialize a session. This rule generates a unique session ID and generates a session file with unique ID. If a SESSION rowset is present in the request, the rule also adds the rowset to the session file. If the SESSION rowset is missing in the request, a blank unique session rowset is generated and added to the session file.

The SESSION rowset is returned in the result.

Input variables

Variable	Description
XMLSESSION	(Optional) This variable contains a unique identifier for the session. If present, it is used to generate a new session. Otherwise, the rule generates a unique identifier for the new session.
XMLSESSIONTIMEOUT	(Optional) Specifies the session timeout in seconds. The default is 1800 seconds.

Use this INI option to specify a global share for multiple IDS processes:

```
< GlobalData >
    Path =
```

Output variables

Variable	Description
XMLSESSION	This variable contains the unique identifier for the session if the session is valid. Otherwise, the value will be *INVALID*.
RESULTS	Success or failure

termSession

Use this rule to terminate a session. This rule removes the session file associated with the unique ID.

Input variables

Variable	Description
XMLSESSION	This variable contains the unique ID for the session to be removed.

Use this INI option to specify a global share for multiple IDS processes:

```
< GlobalData >
    Path =
```

Output variables

Variable	Description
XMLSESSION	This variable contains the unique identifier for the session. This attachment variable contains *REMOVED* if the session was terminated successfully. Otherwise, the value will be *INVALID*.

Variable	Description
RESULTS	Success or failure

updateSession

Use this rule to update the unique session file with the SESSION rowset in the request message and return the updated rowset in the result.

Input variables

Variable	Description
XMLSESSION	This variable contains the unique ID for the session to be updated.
XMLSESSIONTIMEOUT	(Optional) Specifies the session timeout in seconds.The default is 1800 seconds.

Use this INI option to specify a global share for multiple IDS processes:

```
< GlobalData >
    Path =
```

Output variables

Variable	Description
XMLSESSION	This variable contains the unique identifier for the session if the session is valid. Otherwise, the value will be *EXPIRED* if the session has expired, or *INVALID* if the session is no longer valid.
RESULTS	Success or failure

purgeXMLSessions

Use this rule to remove expired sessions.

Input variables

None. Use this INI option to specify a global share for multiple IDS processes:

```
< GlobalData >
    Path =
```

Output variables

Variable	Description
RESULTS	Success or failure.

saveFile

Use this rule to save the contents of an XML node from a session file as a new file to disk and to add an attachment variable to the result message with the full path and file name of the file saved.

Input variables

Variable	Description
XMLSESSION	This variable contains the unique identifier for the session.
INPUTVAR	The name of the SESSION rowset variable containing the data that is to be saved to disk.
OUTPUTVAR	The name of the attachment variable to add to the output message indicating the full path and file name of the file saved.
PRINTPATH	(Optional) Specifies the output path for the output file. If omitted, the output file is written to the current IDS directory.
FILETYPE	(Optional) Specifies the file type for the output file. If omitted, the default extension DAT is used.

Use this INI option to specify a global share for multiple IDS processes:

```
< GlobalData >
    Path =
```

Output variables

Variable	Description
(variable)	An attachment variable whose name is specified by OUTPUTVAR input attachment variable - will hold a String value indicating the full path and file name of the file saved to disk.
XMLSESSION	This variable contains the unique identifier for the session if the session is valid. Otherwise, the value will be will be *EXPIRED* if the session has expired, or *INVALID* if the session is no longer valid.
RESULTS	Success or failure

USING IDSXML

You can use IDSXML as a guide to processing the errors.xml file in a Microsoft ASP environment.

NOTE: IDSXML is not a COM+ component so do not register it under the Component Services Microsoft Management Console snap-in. The component should only be registered under the current IDS directory on the IDS client (the web server).

Keep in mind IDSXML requires Microsoft XML parser 4.0 (MSXML 4.0). Please make sure you have this before you use IDSXML.

IDSXML is a Win32 COM component. IDSXML provides XML parsing and XSL processing APIs for ASP. Here is a description of the properties and APIs provided by this component:

Properties

This table shows you the properties:

Name	Type	Description	See also
Formset	collection	A collection of form objects	XMLProcessFormset
FormsetSelectionList	string	A comma-delimited string of options selected from a form set. These values are expected: - form - form.copycount.recipient - form.image - form.image.copycount.recipient	XMLUpdateFormset

Here is an example:

```
DEC PAGE,DEC PAGE.1.AGENT,DEC PAGE.1.COMPANY,
DEC PAGE.1.INSURED,DEC PAGE.q1snam,DEC PAGE.q1mdc1,DEC
PAGE.q1mdc2,DEC PAGE.q1mdc3, DEC PAGE.q1mdc3.1.INSURED
```

Methods

IDSXML includes these methods:

- XMLTransformErrors on page 37
- XMLTransformErrors2 on page 38
- XMLLoadINI on page 39
- XMLLoadXML on page 40
- XMLLoadXSL on page 40
- XMLGetGroupOptionValue on page 41
- XMLGetValue on page 41
- XMLGetGroup on page 41
- XMLUpdateGroup on page 42

- XMLBuffer on page 44
- XMLLoadProcessor on page 44
- XMLAddParameterToXSL on page 44
- XMLTransformWithXSL on page 45
- XMLProcessWithXSL on page 46
- XMLUpdateFormset on page 47
- XMLProcessFormset on page 47

XMLTransformErrors

Use this method to transform a result message into useful HTML output. This method takes an XML buffer from a result message that contains errors, an errors XML file that contains error descriptions, and an XSL template which is used to transform the XML message into HTML output that describes errors returned by IDS.

Syntax

```
XMLTransformErrors xmlbuf, xmlFile, xslFile
```

Parameter	Description
xmlBuf	Enter the name of the XML buffer that contains the message returned by IDS. This message contains errors returned by IDS.
xmlFile	Enter the name of the errors file that contains all error codes recognized by IDS. This file is produced by IDS and contains additional information, causes, and resolutions for each error. The errors file is used by this method to transform the message returned by IDS into useful HTML output. This is a file that ships with IDS (errors.xml).
xslFile	Enter the name of the XSL template you want the method to use to transform the XML buffer and errors XML file into HTML information about errors returned by IDS. This is a template that ships with IDS (errors.xsl).

Example

In this example, page one detects an error, captures the buffer that contains the error, and redirects to the error processing page, which is page 2.

Page1: processRequest.asp

```
<%
set DSI = server.CreateObject("IDSASP.DSI")

For i=1 to Request.Form.Count
    DSI.AddReq Request.Form.Key(i), Request.Form(i)
Next

On Error Resume Next

DSI.ProcessRq

If Err.Number <> 0 Then
    Err.Clear
```

```
End if

path = Request.ServerVariables("APPL_PHYSICAL_PATH")


results = DSI.Result("RESULTS").Value
errors = DSI.Result("ERRORS").Value


If Len(results) = 0 OR results <> "SUCCESS" OR CInt(errors) > 0 then
   Session("xmlbuf") = DSI.GetSOAPMessage
   set dsi = nothing
   Response.Redirect "error.asp"
End if


Set DSI = Nothing
%>
```

Page2: error.asp

```
<%
set o = Server.CreateObject("IDSXML.XML")

xmlbuf =  Session("xmlbuf")
xmlFile = Server.MapPath("xml\errors.xml")
xslFile = Server.MapPath("xsl\errors.xsl")


o.XMLTransformErrors xmlbuf, xmlFile, xslFile
%>
```

XMLTransformErrors2

Use this method to transform a result message into useful HTML output. This method takes as input an XML buffer from a result message from IDS that contains errors and an XSL template which is used to transform the XML message into HTML output that describes the errors returned.

Syntax

```
XMLTransformErrors2 xmlbuf, xslfile
```

Parameter	Description
xmlBuf	Enter the name of the XML buffer that contains the message returned by IDS. This message contains errors returned by IDS.
xslFile	Enter the name of the XSL template you want the method to use to transform the XML buffer into HTML information that contains the errors returned by IDS. This is a template that ships with IDS (default.xsl).

Example

In this example, page one detects an error, captures the buffer that contains the error, and redirects to the error processing page, which is page 2.

Page1: processRequest.asp

```
<%
```

```
set DSI = server.CreateObject("IDSASP.DSI")

For i=1 to Request.Form.Count
    DSI.AddReq Request.Form.Key(i), Request.Form(i)
Next

On Error Resume Next

DSI.ProcessRq

If Err.Number <> 0 Then
    Err.Clear
End if

path = Request.ServerVariables("APPL_PHYSICAL_PATH")


results = DSI.Result("RESULTS").Value
errors = DSI.Result("ERRORS").Value


If Len(results) = 0 OR results <> "SUCCESS" OR CInt(errors) > 0 then
   Session("xmlbuf") = DSI.GetSOAPMessage
   set dsi = nothing
   Response.Redirect "error.asp"
End if


Set DSI = Nothing
%>
```

Page2: error.asp

```
<%
set o = Server.CreateObject("IDSXML.XML")

xmlbuf =  Session("xmlbuf")
xslFile = Server.MapPath("xsl\default.xsl")

o.XMLTransformErrors2 xmlbuf, xslFile
%>
```

XMLLoadINI

Use this method to load an XML INI file into memory. Use this method before calling other methods that retrieve information from an XML document.

Syntax

```
XMLLoadINI sIni
```

Parameter	Description
sINI	Enter the full path and file name of the XML document you want to load. This can also be a buffer that contains an XML document.

Here is an example of the format of the XML document:

```
<GROUPS>
   <GROUP NAME="MQSERIES">
    <QUEUEMANAGER>QALAB1</QUEUEMANAGER>
    <CLIENT>YES</CLIENT>
    <REQUESTQ>REQUESTQ</REQUESTQ>
    <RESULTQ>RESULTQ</RESULTQ>
   </GROUP>
   <GROUP NAME="PRINTOPTIONS">
    <ALLRECIPIENTS></ALLRECIPIENTS>
    <PRTDOWNLOADFONTS></PRTDOWNLOADFONTS>
    <PRTTYPE>PDF</PRTTYPE>
    <PRTSENDCOLOR></PRTSENDCOLOR>
    <PRTPAGENUMBERS></PRTPAGENUMBERS>
    <PRTPRINTVIEWONLY></PRTPRINTVIEWONLY>
    <PRTTEMPLATEFIELDS></PRTTEMPLATEFIELDS>
   </GROUP>
   <GROUP NAME="TEST">
    <policy>
        <num>1</num>
    </policy>
   </GROUP>
</GROUPS>
```

Example Here is an example:

```
set o = Server.CreateObject("IDSXML.XML")
sIni = Server.MapPath("ini.xml")
o.XMLLoadIni(sIni)
```

XMLLoadXML

Use this method to load an XML document into memory before an XSL transformation occurs.

Syntax

```
XMLLoadXML sXML
```

Parameter	Description
sXML	Enter the full path and file name of the XML document you want to load. This can also be a buffer that contains an XML document.

Example Here is an example:

```
sXML = Server.MapPath("xml\test.xml")
o.XMLLoadXML(sXML)
```

XMLLoadXSL

Use this method to load an XSL template into memory before an XSL transformation occurs.

Syntax

```
XMLLoadXSL sXSL
```

Parameter	Description
sXSL	Enter the full path and file name of the XSL template you want to load. This can also be a buffer that contains an XSL template.

Example

Here is an example:

```
sXSL = Server.MapPath("xsl\test.xsl")
o.XMLLoadXSL(sXSL)
```

XMLGetGroupOptionValue

Use this method to return a value from an XML node. Use the XMLLoadINI method before you use this method.

Syntax

```
XMLGetGroupOptionValue sGroup, sOption
```

Parameter	Description
sGroup	Enter the group name to use for retrieving a value.
sOption	Enter the option name to use for retrieving a value.

Example

Here is an example:

```
set o = Server.CreateObject("IDSXML.XML")
sIni = Server.MapPath("ini.xml")
o.XMLLoadINI(sIni)
val = o.XMLGetGroupOptionValue("MQSERIES", "CLIENT")
set o = nothing
```

XMLGetValue

Use this method to return a value from a node in an XML tree using xPath. Use the XMLLoadINI method before you use this method.

Syntax

```
XMLGetValue xPath
```

Parameter	Description
xPath	Enter a fully qualified xPath value to the node in the XML document tree.

Example

Here is an example:

```
set o = Server.CreateObject("IDSXML.XML")
sIni = Server.MapPath("ini.xml")
o.XMLLoadINI(sIni)
val=o.XMLGetValue("//GROUP[@NAME='TEST']/policy/num")
set o = nothing
```

XMLGetGroup

Use this method to return an INI group as a buffer, as an object, or as a new file. Use the XMLLoadINI method before you use this method.

Syntax

```
XMLGetGroup sGroup, iOption, sDir
```

Parameter	Description
sGroup	Enter the name of the INI group you want returned.
iOption	Choose one of these options: 1 - Return the INI group as a buffer. 2 - Return the INI group as an object. 3 - Save the INI group as a new file.
sDir	Only include this parameter if you entered three (3) for the iOption parameter. Enter the name of the directory in which you want the system to save the new XML file.

Example Here is an example:

```
<%
set o = server.createobject("IDSXML.XML")
sIni = Server.MapPath("ini.xml")
o.XMLLoadINI(sIni)

'return MQSeries group as buffer
Buffer = o.XMLGetGroup("MQSERIES", 1, "")



'return MQSeries group as object
Set MQSeriesGroup = o.XMLGetGroup("MQSERIES", 2, "")
'traverse through the group object and print all pairs
For i = 1 to MQSeriesGroup.Count
   name = MQSeriesGroup(i).name
   value = MQSeriesGroup(i).Value
   response.write name & "=" & value & "<BR>"
Next
'access a particular value
val = MQSeriesGroup("CLIENT").Value



'write the MQSeries group as a new file into cache directory
o.XMLGetGroup "MQSERIES", 3, "Cache"

'cleanup
set MQSeriesGroup = nothing
set o = nothing
%>
```

XMLUpdateGroup

Use this method to update a group in an XML document via a Group Object parameter. This method reads the group object's properties and updates the XML document's matching group with the object's properties. The method then returns the updated XML document as a buffer or saves it to disk. Use the XMLLoadINI method before you use this method.

Syntax

```
XMLUpdateGroup sGroup, oGroup, iOption, sOut
```

Parameter	Description
sGroup	Enter the name of the group you want to update.
oGroup	Enter the name of the group object you want to use to update a matching group in an XML document.
iOption	Choose one of these options to indicate what you want done with the updated file: 1 - Return the updated XML INI file as a buffer. 2 - Save the updated XML INI file as a file. 3 - Return only the updated group as a buffer. 4 - Save only the updated group as a file.
sOut	Only use this parameter if the iOption parameter is set to two (2) or four (4). Enter the full path and file name for saving the XML document.

Example

Here is an example:

```
<%

Set o = Server.CreateObject("IDSXML.XML")
sIni = Server.MapPath("ini.xml")
o.XMLLoadINI(sIni)

Set PrtOpt = o.XMLGetGroup("PRINTOPTIONS", 2, "")
PrtOpt("ALLRECIPIENTS").Value = "YES"
PrtOpt("PRTDOWNLOADFONTS").Value = "YES"
PrtOpt("PRTTYPE").Value =  "XML"
PrtOpt("PRTSENDCOLOR").Value =  "NO"
PrtOpt("PRTPAGENUMBERS").Value =  "NO"
PrtOpt("PRTPRINTVIEWONLY").Value = "NO"
PrtOpt("PRTTEMPLATEFIELDS").Value = "YES"

'update the xml ini file and return the updated file as a buffer
Buffer = o.XMLUpdateGroup("PRINTOPTIONS", PrtOpt, 1, "")

'update the xml ini file and save it to disk
o.XMLUpdateGroup "PRINTOPTIONS", PrtOpt, 2, "POpt.xml"

'update the group and return the updated group as a buffer
Buffer2 = o.XMLUpdateGroup ("PRINTOPTIONS", PrtOpt, 3, "")

'update the group and save the updated group as a new file
o.XMLUpdateGroup "PRINTOPTIONS", PrtOpt, 4, "newPOpt.xml"
```

```
'cleanup
Set PrtOpt = Nothing
Set o = Nothing

%>
```

XMLBuffer

Use this method to return a buffer for an XML document. Use XMLLoadINI method before you use this method.

Syntax

```
XMLBuffer sFile
```

Parameter	Description
sFile	Enter the full path and file name of the XML document.

Example

Here is an example:

```
<%
set o = Server.CreateObject("IDSXML.XML")

InputFormset = Server.MapPath("xml\OriginalFormset.xml")

Buffer = o.XMLBuffer(InputFormset)

Response.write Buffer

set o = nothing
%>
```

XMLLoadProcessor

Use this method to load an XSL template into memory and to load the XSL processor based on that template. Use this method before you call the XMLAddParameterToXSL or XMLProcessWithXSL methods.

Syntax

```
XMLLoadProcessor sXSL
```

Parameter	Description
sXSL	Enter the full path and file name to an XSL template or a buffer that contains an XSL template.

Example

Here is an example:

```
set o = server.createobject("IDSXML.XML")
template = Server.MapPath("xsl\test.xsl")
o.XMLLoadProcessor(template)
```

XMLAddParameterToXSL

Use this method to add a parameter to internal XSL processor. For instance, you can use this method to add parameters before processing with an XSL template that expects parameters. Use the XMLLoadProcessor method before you call this method.

Syntax

```
XMLAddParameter2XL name, value
```

Parameter	Description
name	Enter the name of the parameter to add to the XSL template.
value	Enter the value of the parameter to add to the XSL template.

Example

Here is an example:

```
<%


set o = Server.Createobject("IDSXML.XML")
sXML = Server.MapPath("xml\test.xml")
sXSL = Server.MapPath("xsl\test.xsl")

o.XMLLoadXML(sXML)

o.XMLLoadProcessor(sXSL)

o.XMLAddParameterToXSL "color", "blue"

o.XMLProcessWithXSL 1 , ""

set o = nothing

%>
```

XMLTransformWithXSL

Use this method to transform an XML document with an XSL template. Use the XMLLoadXML and XMLLoadXSL methods before you call this method. Use this method to process XML documents with XSL templates that do not expect parameters.

Syntax

```
XMLTransformWithXSL Option, sPath
```

Parameter	Description
Option	Choose one of these options: 1 - Return the transformation result as a string and write the result to the screen. 2 - Return the transformation result as a string. 3 - Write the transformation result to disk using the path specified by sPath parameter. This also returns the transformation result as a string.
sPath	Only include this parameter if you entered three (3) for the Option parameter. Enter the full path and file name of the file you want to use for writing the transformation result to disk.

Example Here is an example:

```
<%

set o = Server.Createobject("IDSXML.XML")
sXML = Server.MapPath("xml\test.xml")
sXSL = Server.MapPath("xsl\test.xsl")

o.XMLLoadXML(sXML)

o.XMLLoadXSL(sXSL)
o.XMLTransformWithXSL 1, ""

set o = nothing

%>
```

XMLProcessWithXSL

Use this method to transform an XML document using an XSL template that expects parameters. You should use the XMLAddParameterToXSL method to add the parameters expected by the style sheet before you call this method. Also use the XMLLoadProcessor method before you call this method.

Syntax

```
XMLProcessWithXSL Option, sPath
```

Parameter	Description
Option	Choose one of these options: 1 - Return the transformation result as a string and write the result to the screen. 2 - Return the transformation results as a string. 3 - Write the transformation result to disk using the path specified by sPath parameter. This also returns the transformation result as a string.
sPath	If you set the Option parameter to three (3), enter the full path and file name of the file you want to use for writing the transformation result to disk.

Example Here is an example:

```
<%


set o = Server.Createobject("IDSXML.XML")
sXML = Server.MapPath("xml\test.xml")
sXSL = Server.MapPath("xsl\test.xsl")

o.XMLLoadXML(sXML)

o.XMLLoadProcessor(sXSL)

o.XMLAddParameterToXSL "color", "blue"

o.XMLProcessWithXSL 1 , ""

set o = nothing
```

```
%>
```

XMLUpdateFormset

Use this method to update an XML form set. This method takes as input a buffer that contains an XML form set or a string that specifies the full path and file name for an XML form set. The method uses the FormsetSelectionList property, which contains a comma-delimited string of forms, images, and recipients, to modify the form set.

This method generates a unique name for the updated form set and saves it as a new XML document. It returns the full path and name of the new XML document.

Syntax

```
XMLUpdateFormset sXML
```

Parameter	Description
sXML	Enter the full path and file name of an XML form set or a buffer that contains an XML form set.
sDir	Enter the name of a directory into which you want to save the updated form set.

Example

Here is an example:

```
<%
set o = Server.CreateObject("IDSXML.XML")
InputFormset = Server.MapPath("original.xml")

o.FormsetSelectionList = "DEC PAGE,DEC PAGE.1.AGENT,DEC
PAGE.1.COMPANY,DEC PAGE.1.INSURED,DEC PAGE.q1snam,DEC
PAGE.q1mdc1,DEC PAGE.q1mdc2,DEC PAGE.q1mdc3"

OutFormset = o.XMLUpdateFormset(InputFormset, "Cache")
set o = nothing
%>
```

XMLProcessFormset

Use this method to process a form set. This method takes as input a buffer that contains an XML form set or a string that specifies the full path and file name of an XML form set. The method parses the XML form set and converts it into a public form set collection property. The form set collection contains form objects and each form object contains images and recipients.

Syntax

```
XMLProcessFormset xmlBuffer
```

Parameter	Description
xmlBuffer	Enter the full path and file name of an XML form set or a buffer that contains the XML form set you want loaded and returned as a collection.

Example

Here is an example:

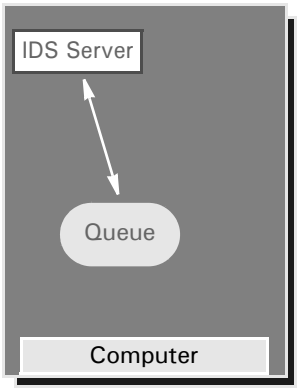
```
set o = Server.CreateObject("IDSXML.XML")
InputFormset = Server.MapPath("xml\OriginalFormset.xml")
```

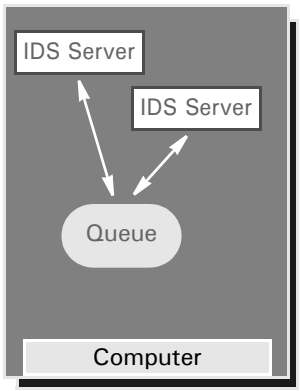
```
Buffer = o.XMLBuffer(InputFormset)
o.XMLProcessFormset Buffer


For i = 1 To o.Formset.Count
    formName = o.Formset.Item(i).NAME
    formID = Pad(formName)
    html = "<input type=checkbox name=SELECTION value=" & formID & _
           " onclick='FormSelect(this);'>" & formName
    form = "FORM." & CStr(i)
    oTree.Add "root", form, html, bExpand, "page.gif"
    html = "Recipients"
    FRPCS = "FRECIPIENTS" & CStr(i)
    oTree.Add form, FRPCS, html, bExpand, "mydoc.gif"

    For k = 1 To o.Formset.Item(i).Recipients.Count
        recipientName = o.Formset.Item(i).Recipients.Item(k).NAME
       recipientCnt = o.Formset.Item(i).Recipients.Item(k).CopyCount
       recipientID = formID & "." & recipientCnt & "." & recipientName
        html = "<input type=checkbox name=SELECTION value=" &
recipientID & _
               ">" & recipientName
        recipient = "RECIPIENT" & "." & CStr(i) & "." & CStr(k)
        oTree.Add  FRPCS, recipient, html, bExpand, "n.gif"
    Next

    For j = 1 To o.Formset.Item(i).Images.Count
        imageName = o.Formset.Item(i).Images.Item(j).NAME
        imageID = formID & "." & imageName
       html = "<input type=checkbox name=SELECTION value=" & imageID
& ">" & _
               "<font color=blue>" & imageName & "</font>"
        image = "IMAGE." & CStr(i) & "." & CStr(j)
        oTree.Add  form, image, html, bExpand, "help_page.gif"

       For k = 1 To o.Formset.Item(i).Images.Item(j).Recipients.Count
            recipientName =
o.Formset.Item(i).Images.Item(j).Recipients.Item(k).NAME
            recipientCnt =
o.Formset.Item(i).Images.Item(j).Recipients.Item(k).CopyCount
         recipientID = formID & "." & imageName & "." & recipientCnt
& "." &  recipientName
            html = "<input type=checkbox name=SELECTION value=" &
recipientID & ">" & _
            recipientName
           recipient = "RECIPIENT" & "." & CStr(i) & "." & CStr(j)
& "." & CStr(k)
            oTree.Add  image, recipientID, html, bExpand, "n.gif"
        Next
    Next
Next
```

Using Multiple Servers

To further increase performance, you can set up multiple servers. Each server you set up helps process client requests. You can set up additional servers in a variety of ways, as this diagram shows:

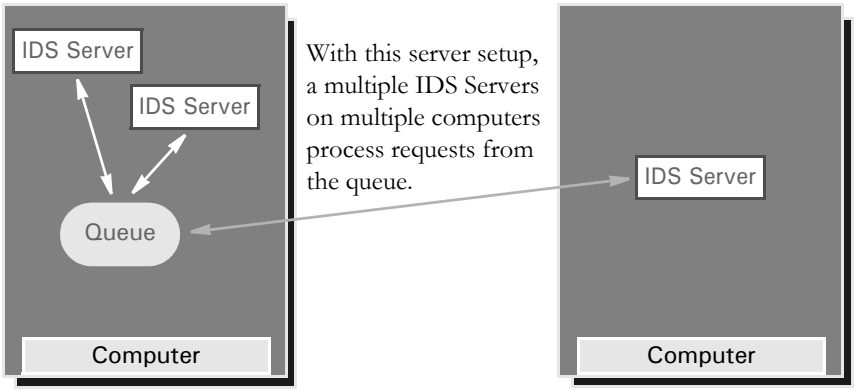
With this server setup, a single IDS Server processes requests from the queue. Both are physically located on the same computer.





With this server setup, a multiple IDS Servers processes requests from the queue. Both the servers and the queue are physically located on the same computer.

With this server setup, a multiple IDS Servers on multiple computers process requests from the queue.



To determine which server setup will work best for you, first determine if your transactions are CPU or I/O (input/output) intensive. Then take a look at the test results we have compiled.

Determining if Your Transactions are CPU or I/O Intensive

To determine if the transactions the server is processing are CPU or I/O intensive, look at the Windows Task Manager:

- If the CPU gauge shows around 100% CPU usage with no other applications running, the transactions are *CPU intensive*.
- If the CPU gauge shows less than 80% CPU usage, the transactions are, most likely, *I/O intensive* (this includes network I/O).

Here are some scenarios and recommendations:

Scenario	Recommended Server Setup
Low transaction volume. Each transaction takes a few seconds to process.	No changes in configuration are required. One server should be able to process all of the requests within reasonable period of time.
High transaction volume. Each transaction takes a few seconds to process.	If clients are getting the *time-out waiting for Server* error message, increase the time-out value for the clients. To do this, set TimeOut INI option in the ReqType:XXX control group. You can set this option for each request type. This lets you set it to a higher value for requests which take longer to process. The default time-out for each request type is 60 seconds. To increase total throughput, try adding a second server. Keep in mind that adding a second server does not let you process twice as many transactions. You will probably see a performance increase of around 10-20 percent.
Any volume. Each transaction takes a few *minutes* to process. Mostly I/O intensive.	You may see this scenario when the rules have to retrieve data from a mainframe computer via ODBC or DB2, or when the rules have to do a lot of file processing.*1 In this situation, try adding additional servers on the same computer as the original server.
Any volume. Each transaction takes *minutes* to process. Mostly CPU intensive.	You may see this scenario if you use a lot of calculation-intensive rules. In this scenario, the best solution is to add a second server on a *separate* computer. If you want to add more servers, add them on separate computers so you end up with a computer for each server.

*1 The number of ODBC connections to MVS is limited by MVS and ODBC drivers. You cannot exceed this limit.

Performance Measurements when Using Multiple Servers

To help you choose the right server setup for your needs, here are some test results compiled from multiple server runs.

These tests were run with a specified number of servers and clients. The servers were started from the command line, so there is no built-in web server overhead or limitations. For these tests, all clients ran at the same time. In typical implementations, you seldom have all users working on the server at the same time.

Clients	Number of servers	Number of transactions per hour
Short transactions, each takes about 1 second or less		
20	1	1680
20	2	1825
20	3	1583 (note the performance degradation)
40	1	1211
Not CPU intensive transactions, each takes about 40 seconds		
-	1	77
-	3	263
CPU intensive transactions (100% CPU usage in the NT task manager), each takes about 10 seconds		
-	1	372
-	2 (same PC)	372 (no difference)
-	2 (different PC)	754

Setting Up Additional Servers

You can start multiple instances of IDS by default. Running multiple instances of IDS is generally required for performance reasons.

You control the number of instances IDS starts using this Configuration option:

```
<section name="DocumentServer">
    <entry name="Instances">2</entry>
</section>
```

The default is two (2). You can enter a number from 1 to 10.

Specifying the INI file to use

You can specify the name and location of the DAP.INI file you want to use in the DPRInit rule as shown here:

```
<section name="REQTYPE:INI">
    <entry name="function">dprw32-
&gt;DPRInit,500,d:\docserv\dap.ini</entry>
</section>
```

Separate parameters with commas.

The first parameter specifies the file cache. The default FAP file cache is 1000. The second parameter specifies where to find the INI file. *DAP.INI* is the default file name.

NOTE: This approach does not work with the DPRCoLogin rule. Use the DPRLogin rule instead.

Setting Up a Windows NT Service

You can configure the Internet Document Server to run as a Windows service.

NOTE: Do not install the Internet Document Server and Internet Document Master Server as a Windows NT service until you have checked to make sure the system was properly installed.

To set up the Internet Document Server as a Windows service, go to the directory where it is installed and run the batch file, DS-SERVICE.BAT. This will install IDS as a service called *Docupresentment Server*.

To uninstall Internet Document Server as a Windows service, go to the directory where it is installed and run the batch file, DS-SERVICE-UNINSTALL.BAT.

When running as a service, messages usually written to the console's standard output are written to a text file named DS-STDOUT.TXT. The messages usually written to the console's standard error are written to a text file named DS-STDERR.TXT.

Handling Multi-threaded Requests

IDS can run multiple requests at the same time in separate threads of execution. If the requests run are safe to run in multiple threads and are mixed between I/O-based and computation-based, then running some requests in multiple threads can be an alternative to running multiple instances of IDS.

An instance of IDS has a main thread that initializes global data and is the default for running all requests. You can configure IDS to start extra threads to run some requests. This is done in the docserv.xml configuration file, in the 'BusinessLogicProcessor' section:

```
<entry name="RequestProcessors”>1</entry>
```

Entry	Description
RequestProcessors	Indicates how many extra threads to set up to run requests. If the entry is set to zero (0), no extra threads are set up and all requests are run serially.

To specify that a request can be run in an extra thread, in the docserv.xml configuration file, create a 'MultiThreadedRequests' subsection in the 'BusinessLogicProcessor' section:

```
<section name="MultiThreadedRequests">
    <entry name="Request">FTPSEND</entry>
</section>
```

All requests mentioned in this section will be run in multiple threads if the 'RequestProcessors' entry is set up.

Each thread has its own separate input and output state to keep track of message variables and attachments per request being run, so code in rules that read or change message variables or attachments will not interfere with other requests running at the same time. This includes calls to:

- DSIMessage.getMsgVar, DSIMessage.setMsgVar, and so on in the Java and scripting rules.
- DSIJQueue.LocateAttachVar, DSIJQueue.AddAttachVar, and so on in the IDS version 1.x Java rules.
- DSILocateAttachVar, DSIAddAttachVar, and so on in the C rules.

This does not mean all rules are safe to run in multiple threads, just that calls to the DSI API code do not prevent rules from being run in multiple threads. For example, Documaker code is not safe to run in multiple threads. If you need to run multiple Documaker-related requests at the same time, you must run multiple instances of IDS.

INI vs. THREADINI control sections

IDS version 1.x has a INI control section where global data is created and destroyed. Since IDS version 2.x can have multiple threads, you need a way to create and destroy data needed by each thread. This is done with the THREADINI control section. The THREADINI control section is run once for each thread as it is started and once for each thread when it is stopped. An example of code needing this is COM setup in Windows; every thread in Windows that will be running COM code needs to initialize COM for that thread. Here is a sample INI and THREADINI control section:

```
<section name="ReqType:INI">
    <entry name="function">irlw32->;IRLInit</entry>
    <entry name="function">dprw32->;DPRInit</entry>
</section>
<section name="ReqType:THREADINI">
    <entry name="function">DSICoRul->;Init</entry>
</section>
```

Threads and Inter-Rule data in C rules

IDS rules written in C use the functions DSICreateValue, DSILocateValue, and DSIDestroyValue to create data in one part of code to use in other parts of code.

A common usage of these functions is to allocate data in the DSI_MSGINIT part of a function, use it in the DSI_MSGRUNF and DSI_MSGRUNR parts of a function, and free the data in the DSI_MSGTERM part of a function. All this is done inside a single request.

A less common usage is to set up data in the INI request used for the entire runtime of IDS. Rules in the INI request type have their DSI_MSGINIT code run when IDS starts and their DSI_MSGTERM code run when IDS shuts down. This means data allocated with DSICreateValue in the DSI_MSGINIT part of a INI request rule function is available for all other requests run by IDS. This data is usually read-only, for example configuration information from the DAP.INI file and MRLs set up by the DPRInit function.

Since IDS can run requests in multiple threads simultaneously, each thread needs it's own set of data for running requests plus access to the global configuration information. And, at the same time, each thread needs to remain compatible with C rules using the DSICreateValue, DSILocateValue and DSIDestroyValue functions. This is done by having two data contexts for data: *global* and *thread-local*.

Global context is when IDS is running rules in the INI request type. The INI rules are run with DSI_MSGINIT before the other rule threads are created, and the rules are run with DSI_MSGTERM after the other threads are destroyed. Thread-local context is when all other requests are run.

When the functions DSICreateValue and DSIDestroyValue are called during global context the values are put in global data; in thread-local context the values are put in thread-local data.

When DSILocateValue is called in thread-local context, the thread-local data is first checked to see if has the data. If the value is in thread-local data, it is returned. If not, then global data is checked, and returned if it is there. Only if the value is missing in both places will DSILocateValue indicate that the data is not found.

This allows C rules using DSICreateValue, DSILocateValue, and DSIDestroyValue to run in multiple threads but remain compatible with previous versions of IDS.

Threads and Inter-Rule data in Java and scripting rules

Since Java rules are based on objects that have their own state, this use of the C DSI-Value functions is not required. When Java rules are used with transaction scope, an instance of the class will remain for the run of the request, so data can be put in member variables and will remain. There are functions available that do the same thing for passing data from one rule to another or for use in one rule if it is run in static scope.

The RequestState object passed in to a Java rule has the methods putObject, getObject, and removeObject. Since each thread has its own RequestState object, you can use these functions to keep track of data for each request. These functions let you store any type of object, not just byte arrays like the C functions.

To pass data between C and Java rules, use the RequestState methods createVar, locateVar, and destroyVar. These correspond to the C functions DSICreateValue, DSILocateValue, and DSIDestroyValue, so they can only use byte arrays for passing data. The data context setup is also the same as for these C functions.

To set up and use Java global data, use the GlobalVarStorage class from the DocuCorpUtil library.

USING THE JAVA TEST UTILITY

IDS includes a Java threads test utility you can run to send requests to IDS using single or multiple threads. It also supports attachments and rowsets. You can also feed it a debug message file, such as a receive.msg file generated by IDS (see the ReceiveMessage log4j category in the logconf.xml file under the docserv directory) which can contain more than one transaction that was previously processed. This can be useful in recreating or duplicating a set of transactions for testing.

You can invoke the test utility via the threads script shipped with IDS.

NOTE: If you invoke the test utility without any arguments, it displays usage information. For more information, see the HTML documentation for the com.docucorp.test.threads class that is included with Java SDK.

You must have Java version 5 or later installed to use this test utility.

Using Rules Written in Other Scripting Languages

IDS can run rules written in scripting languages. In addition to rules written in Java, C, and Visual Basic, IDS adds rules written in scripting languages (Java Script, Python, and so on) for debugging, fast prototyping, or rarely run rules.

In a REQTYPE section, add a rule entry such as:

```
<entry name="function">script;test.py;runRule</entry>
```

This rule entry uses the runRule function in the Jython script file (TEST.PY).

The following example programs do the same thing. The first is in JavaScript, the second in Jython, a dialect of Python that runs under Java Virtual Machines. You can find information about which languages are available and where to get them at:

http://jakarta.apache.org/bsf/index.html

Here is a JavaScript example rule:

```
function runRule(requestState, idsArgs, idsMessage) {

    switch (idsMessage) {
        case IDSConstants.init :
            break;
        case IDSConstants.runForward :
            break;
        case IDSConstants.runReverse :
            text = requestState.getOutput().getMsgVar("LANGUAGE");
            if (text == null) {
                text = "JavaScript";
            } else {
                text = text + " and JavaScript";
            }
            requestState.getOutput().setMsgVar("LANGUAGE", text);
            break;
        case IDSConstants.terminate :
            break;
    }

    return IDSConstants.success;
}
```

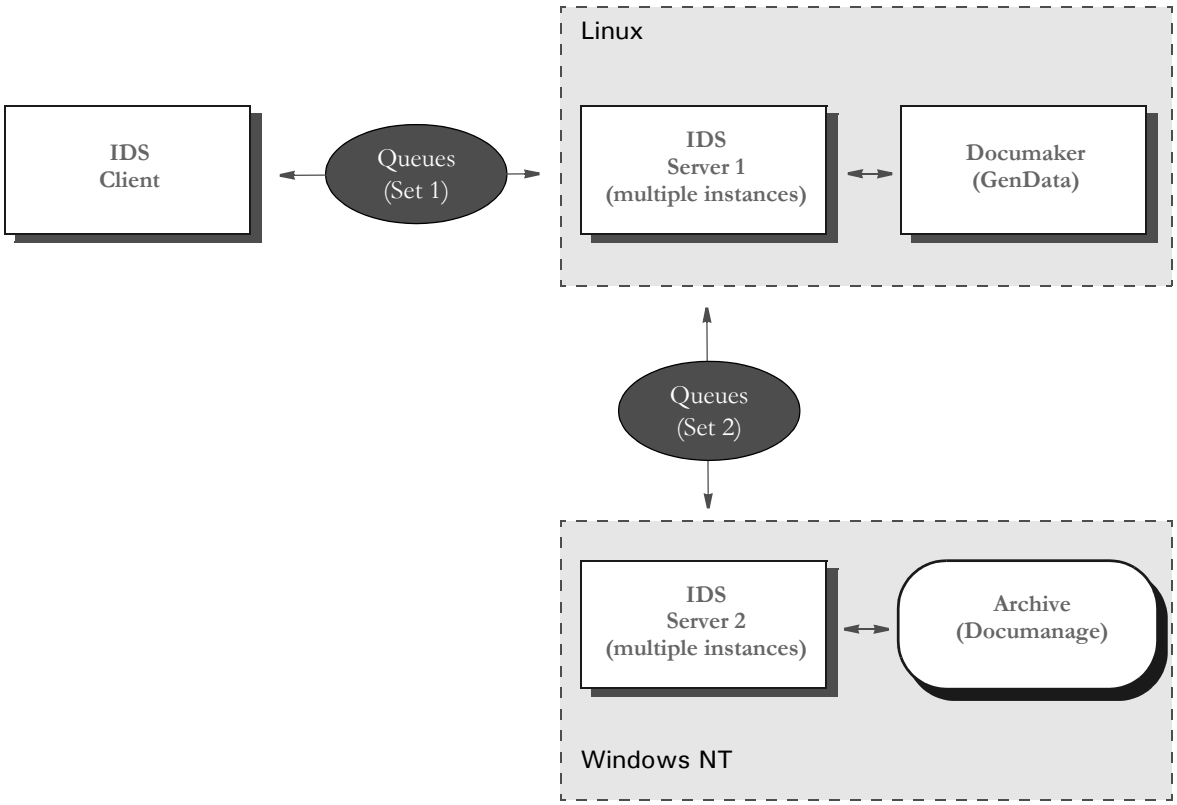
Here is a Jython example rule:

```
def runRule(requestState, idsArgs, idsMessage):

    if idsMessage == IDSConstants.runReverse:
        text = requestState.getOutput().getMsgVar("LANGUAGE")
        if text is None:
            text = "Jython"
        else:
            text = text + " and Jython"
        requestState.getOutput().setMsgVar("LANGUAGE", text)

    return IDSConstants.success
```

Using IDS as a Client to Another IDS

You can set up an IDS installation running under Linux and use the RunRP rules to archive data from Documaker output into a Documanage archive stored on a Windows NT computer. To do this, you must set up your system as explained below.

NOTE: This solution can also be used for other situations, such as when you need to execute rules across platforms.



The archive process

The IDS client submits an XML extract file to IDS (IDS Server 1). On this request IDS Server 1 executes the RunRP rules. After Documaker executes and prior to this transaction being complete, IDS Server 1 has access to NA file, POL file, and NEWTRN file. These files are attached to the request to IDS Server 2 (on Windows NT).

IDS Server 2 archives data into Documanage and returns the error code. IDS Server 1 receives the return code from IDS Server 2, adds all the attachment variables from IDS Server 2 to the output attachment, and replies to the IDS client. IDS Server 2 does not use GenArc or single-step GenData to archive, which reduces the number of resource and setup files needed on Windows NT.

The retrieval process

The IDS client submits the request to retrieve data to IDS Server 1. IDS Server 1 submits the request to get the DPA file to IDS Server 2. The IDS Server 2 gets the DPA file from Documanage, attaches it to the SOAP message and sends the reply to IDS Server 1. IDS Server 1 receives the Documaker file, runs rules to produce a PDF file and sends the PDF file to the client (this can be done as a file on disk or attached to the SOAP message).

Keep in mind:

- MQSeries is used as a queuing system in both places
- Java rules are used to send messages from IDS Server 1 to IDS Server 2 so JVM has to be available on Linux platform to execute Java rules.
- Some of the Documaker resources (like DFD files, INI files) must be available to both IDS implementations and must be kept in sync. One way to synchronize resources is by using mounted volumes from Windows NT to Linux to IDS Server 2 on Windows NT has access to the same physical files as IDS Server 1 on Linux.
- You use the IDSClientRule to make IDS act as a client to another implementation of IDS. This rule is explained below:

Using the IDSClientRule

Use this rule to have IDS act as an IDS client to send a request to a second IDS. You can set communications parameters to talk to the second IDS and request parameters to set up request types and other attachment variables to send to the second IDS.

The attachment variables and files you want to send can be hard-coded or retrieved as attachment variables from the first IDS. Attachment variables returning from the second IDS can be put in the input or output queue of the first IDS.

If the second IDS returns files in its result, the files can be written to disk with unique names and cached. The unique names of the files are stored in attachment variables for use by other rules. The rule can be run on the run forward or run reverse message.

Keep in mind that only MQSeries setups are supported. This rule has transaction scope and the method in the Java class is *callRequest*. The syntax of the function line in a request is shown here:

Syntax

```
function = dsijrule->JavaRunRule,;com/docucorp/ids/rules/
IDSClientRule;CALL;transaction;callRequest;ARG,MESSAGEFILE=call.pro
perties&REQUESTFILE=call.txt&TIMEOUTSEC=15&RUN=RUNF&DEBUG=Y&DESTINA
TION=OUTPUT
```

Parameter	Description
MESSAGEFILE	This parameter points to the file that holds the settings for communicating with the second copy of IDS.
REQUESTFILE	This parameter points to the file that holds attachment variables and files that are sent as a request to the second IDS.
TIMEOUTSEC	Indicates the number of seconds to wait for a reply from the second IDS.
RUN	Indicates the request message during which the rule will run. This parameter is either *RUNF* for the run forward message or *RUNR* for the run reverse message.

Parameter	Description
DEBUG	Determines if debugging messages are put in the Java rule log file. The default is No.
DESTINATION	This parameter says which queue will get the results back from the second copy of IDS. The default is OUTPUT.

You can set communications parameters to talk to the second IDS. This is done by setting up the file referenced in the MESSAGEFILE rule parameter.

Here is a sample IDSClientRule communications properties file. Parameters are explained by comment lines above the parameters:

This indicates the class that implements queuing, in this case MQSeries:

```
queuefactory.class=com.docucorp.messaging.mqseries.DSIMQMessageQueu
eFactory
```

This indicates the class that formats the DSIMessage to send to IDS. Uncomment this line to communicate with IDS version 1.6, leave it commented for subsequent versions:

```
#marshaller.class=com.docucorp.messaging.data.marshaller.LegacyByte
ArrayDSIMessageMarshaller
```

The following are sample MQSeries parameters. This is the queue manager for system hosting MQSeries:

```
mq.queue.manager=QM_pdtest
```

This specifies the MQSeries channel the messaging client and IDS use to communicate:

```
mq.queue.channel=SCC_pdtest
```

The MQSeries communication can be either in *bindings* mode (the program is running on the same machine as the MQSeries server) or in *client* mode (the program is running on a different machine and communicates with the MQSeries server through TCP/IP). If the setting *mq.tcpip.host* is defined, the system uses client mode, otherwise it uses bindings mode:

```
mq.tcpip.host=10.10.10.10
```

This specifies the TCP/IP port the MQSeries server is listening to. *1414* is most commonly used.

```
mq.tcpip.port=1414
```

A client program sends requests out and gets results in:

```
mq.outputqueue.name=requestq
mq.inputqueue.name=resultq
```

This determines how long, in seconds, the MQSeries server keeps a message in the queue if a program does not get it.

```
mq.outputqueue.expiry=120
```

Here is the complete example:

```
queuefactory.class=com.docucorp.messaging.mqseries.DSIMQMessageQueu
eFactory
#marshaller.class=com.docucorp.messaging.data.marshaller.LegacyByte
ArrayDSIMessageMarshaller
mq.queue.manager=QM_pdtest
```

```
mq.queue.channel=SCC_pdtest
mq.tcpip.host=10.10.10.10
mq.tcpip.port=1414
mq.outputqueue.name=requestq
mq.inputqueue.name=resultq
mq.outputqueue.expiry=120
```

Attachment variables and files to send can be hard coded or retrieved as attachment variables from the first IDS. This is done in the file mentioned in the REQUESTFILE parameter. The file can have these sections:

- [MESSAGES] for attachment variables in *name=value* pairs
- [TEXTFILES] for sending text files
- [BINARYFILES] for sending binary files
- [RECEIVEDFILES] for the handling of files returned from the second IDS

[TEXTFILES] and [BINARYFILES] have *file ID=file location* pairs, listing the name that the file data can be identified by and where to get the file's information. Here is a sample REQUESTFILE:

```
[MESSAGES]
REQTYPE=CUSTOMREQUEST
USERID=GUEST
[TEXTFILES]
TEXT1=/home/fap/text1.txt
TEXT2=/home/fap/text2.txt
[BINARYFILES]
BIN1=/home/fap/binary1.bin
BIN2=/home/fap/binary2.bin
```

You can use attachment variables from the current running state of IDS as values in any of the above sections. For example:

```
~GetAttach VARIABLENAME,INPUT
```

will use an attachment variable from the input queue

```
~GetAttach VARIABLENAME,OUTPUT
```

will use an attachment variable from the output queue. If instead of the name of an attachment variable, you use an asterisk (*), every attachment variable from that queue will be sent. Here is an example

```
~GetAttachment *,INPUT
```

This tells the system to ignore the attachment variable name. If you use an asterisk (*), the request type of the request to send is not changed. An explicit *REQTYPE* is required in the REQUESTFILE file.

For the [RECEIVEDFILES] section, the string to the left of the equals sign (=) is the name of the file coming back from the second IDS. To the right of the equals sign is the name of the attachment variable that contains the unique name and path of the generated file, the directory where the file will be stored, and, optionally, the number of seconds to cache the file. The default cache is 3600. For text files, the system appends *.txt*. For binary files, it appends *.bin*.

Here is a complete example of a request file:

```
[MESSAGES]
REQTYPE=CUSTOMREQUEST
DUMMY = ~GetAttach *, INPUT
[TEXTFILES]
TEXT1=c:\fap\text1
[BINARYFILES]
BIN1= ~GetAttach BINFILE, OUTPUT
[RECEIVEDFILES]
ZZZT=MYTEXT, ., 1800
ZZZB=MYBIN, c:\fap, 600
```

In this example after the run, if the current directory is c:\docserv, then the file sent in ZZZT would be stored in

```
c:\docserv\1729530022133082002.txt
```

That file name would be stored in the attachment variable MYTEXT and the file sent in ZZZB would be stored in:

c:\fap\6229680022133082002.bin

That file name would be stored in the attachment variable MYBIN. The text file, (c:\docserv\1729530022133082002.txt) would be cached by IDS for 1800 seconds. The binary file (c:\fap\6229680022133082002.bin) would be cached by IDS for 600 seconds.

MONITORING IDS WITH SNMP TOOLS

You can connect IDS to SNMP agents (SNMP server programs) so performance data can be viewed by SNMP monitors (SNMP client programs). The connection is done with the SNMP AgentX protocol, so you can use any SNMP agent program that supports AgentX.

If an AgentX-enabled SNMP agent is not available for a particular platform, IDS includes a Java-based SNMP agent application you can use. This version includes a Management Information Base (MIB) file, that can be used by SNMP monitor programs to map text names and data types to the MIB number addresses for SNMP objects.

The primary server reports the number of instances of IDS that are running. Each instance will report:

- The amount of time since it started running.
- The amount of time since the last restart.
- The number of successful transactions.
- The number of transactions that caused errors.
- The number of times the instance has been restarted.
- The amount of time needed to run the last transaction.
- The request type of the last transaction.
- The amount of time needed to run the longest transaction so far.
- The request type of the longest transaction so far.
- The number of transactions that have occurred in the last minute.
- The number of transactions that have occurred in the ten last minutes.
- The number of transactions that have occurred in the last hour.

To monitor IDS with SNMP tools, in the docserv.xml configuration file, create an SNMP subsection under the DocumentServer, messaging subsection, as shown here:

```
<section name="DocumentServer">
    <section name="SNMP">
        <entry name="Enabled">yes</entry>
        <entry name="MasterAddress">10.1.10.100</entry>
        <entry name="MasterProtocol">UDP</entry>
        <entry name="MasterPort">705</entry>
    </section>
</section>
```

Entry	Description
Enabled	Determines whether or not SNMP support is enabled. The default is No.
MasterAddress	Is the IP address of where the master SNMP agent program is running. The default is localhost.
MasterProtocol	Is the communications protocol for communicating with the SNMP agent program. You can enter *UDP* for communicating with IDS's included agent or *TCP* for communicating with other AgentX-based SNMP agent programs, such as net-snmp. The default is UDP.

Entry	Description
MasterPort	Is the port the master SNMP agent program uses for the AgentX protocol. The default is 705.

MONITORING REQUESTS

The SNMP monitoring capabilities in IDS allow the monitoring of extra requests and rules by performance monitoring applications, such as LoadRunner.

You can have up to five statistics monitors to measure performance by SNMP. Each monitor measures either an entire request or an individual rule in a request. For each monitor, the time it takes to execute each *message* part (initialization, run forward, run reverse, and terminate) as well as the total time for execution is available.

NOTE: The MIB file can use these monitors, but it is not required.

To enable the statistics monitors, in the docserv.xml configuration file, in the DocumentServer section, add a StatisticsMonitors subsection, as shown here:

```
<section name="DocumentServer">
   ...
    <section name="StatisticsMonitors">
        <entry name="Monitor">SCH</entry>
        <entry name="Monitor">RCP</entry>
        <entry name="Monitor">PRT</entry>
        <entry name="Monitor">PRT/7</entry>
        <entry name="Monitor">PRT/8</entry>
    </section>
</section>
```

Each Monitor entry can be the name of a request or the name of a request followed by a slash (/) and a number. The number is the number of the active entry in the request section. For example, for this request type...

```
<section name="ReqType:PRT">
    <entry name="function">atcw32-&gt;ATCLogTransaction</entry>
    <entry name="function">atcw32-&gt;ATCLoadAttachment</entry>
    <!-- This comment line is skipped -->
    <entry name="function">dprw32-&gt;DPRSetConfig</entry>
    <entry name="function">dprw32-&gt;DPRTermDB</entry>
    <entry name="function">dprw32-&gt;DPRInitLby</entry>
    <entry name="function">atcw32-&gt;ATCUnloadAttachment</entry>
    <entry name="function">dprw32-&gt;DPRRetrieveFormset</entry>
    <entry name="function">dprw32-&gt;DPRPrint</entry>
    <entry name="function">dprw32-&gt;DPRProcessTemplates</entry>
    <!-- -->
</section>
```

The monitor entry *PRT/7* would refer to the DPRRetrieveFormset rule and *PRT/8* would refer to the DPRPrint rule.

Managing IDS Instances

You can use the Watchdog process to manage and monitor IDS instances. The Watchdog process is the one that is now started, stopped, and configured as a service and it in turn is responsible for managing and monitoring the IDS instances. It monitors the health of each instance and restarts it or stops it when needed. You can also configure the Watchdog process through log4j to send email notifications when an instance encounters a fatal or mission-critical error.

Watchdog also monitors the idle time for each instance and starts additional ones when all running instances are under load. These options are supported:

Options	Description
Instances	(Optional) The number of IDS instances Watchdog should start at startup. The default is two (2).
UseLoadBalancing	(Optional) This option controls whether Watchdog checks the idle time of the instances that are running and starts additional ones when all of them are busy. Instances are considered busy when their idle time is less than the value provided in the MinIdleTimeSeconds option. Watchdog uses the value provided in the IdleTimeChecks option to determine the number of idle time checks to run before it starts additional instances. When additional instances are started for load balancing purposes, they are shut down by Watchdog if their idle time exceeds the value in the MaxIdleTimeSeconds option. The maximum number of instances running is the value for the ***MaxInstances*** option (including the instances configured in the ***Instances*** option). Watchdog checks the idle time of the current instances at the interval specified in the IdleTimeCheckIntervalSeconds and if all are busy, it starts an additional number of instances equal to the value provided in the IncrementCount option. Please note that Watchdog does not start checking the busy time of the current instances until the time provided in the IdleTimeCheckDelaySeconds option is reached. Make sure the value for the delay is ample enough to provide for all instances to start and reach an idle time equal to or greater than the value provided for the MinIdleTimeSeconds option. You can enter Yes (or True) or No (or False). The default is Yes.
MaxInstances	(Optional) This option controls the maximum number of instances that can run when the UseLoadBalancing option is enabled. The default is the number of processors times two.
IncrementCount	(Optional) This option controls how many additional instances are started during the current check when all instances running are busy and the UseLoadBalancing option is enabled. The default is two (2).

Options	Description
IdleTimeCheckIntervalSeconds	(Optional) This option controls how often Watchdog checks the idle time of the instances that are running to determine if they are busy so it can start additional ones when the UseLoadBalancing option is enabled. The default is 60 seconds.
IdleTimeCheckDelaySeconds	(Optional) This option controls the initial delay before the first idle time check is performed by Watchdog when the UseLoadBalancing option is enabled. This time should be ample enough to allow all instances to start and reach an idle time equal to or greater than the value provided for the MinIdleTimeSeconds option. The default is 120 seconds.
IdleTimeChecks	(Optional) This option defines the number of consecutive Idle time checks that must fail, meaning all instances were busy during each check, before more instances are started when the UseLoadBalancing option is enabled. Each check takes place at the IdleTimeCheckIntervalSeconds interval. The default is two (2).
MinIdleTimeSeconds	(Optional) This option controls the minimum idle time for each instance. The idle time represents how long it has been since an IDS instance processed the last request. If Watchdog detects an instance has an idle time less than the value provided for this option, it considers it busy for the purpose of load balancing. The default is five (5) seconds.
MaxIdleTimeSeconds	(Optional) This option controls the maximum idle time for an additional instance. The idle time represents how long it has been since an IDS instance processed the last request. If Watchdog detects an instance which was started for the purpose of load balancing has reached an idle time greater than the value provided for this option, it sends the instance a shutdown request. The default is 120 seconds.
MaxTransactions	(Optional) This option controls the maximum number of transactions an instance can process before it is restarted by Watchdog. Enter **-1** to disable this option. The default is 10000.
MaxReportIntervalSeconds	(Optional) This option controls the maximum time interval that can elapse without an instance reporting back to Watchdog before it is restarted. The default is 120 seconds.
MaxUpTimeSeconds	(Optional) This option controls the maximum time interval an instance can run before it is restarted by Watchdog. Enter **-1** to disable this option. The default is 28800 seconds (8 hours).

Options	Description
MaxRestarts	(Optional) This option controls the maximum number of restart attempts that can occur within a time interval specified by the RestartIntervalSeconds option before Watchdog shuts down. Use this option to prevent Watchdog from attempting to restart instances infinite times when they cannot be started due to configuration errors and so on. The default is five restarts.
RestartIntervalSeconds	(Optional) This option controls the interval used with the MaxRestarts option to determine if Watchdog is having a problem starting instances and to prevent continuous or infinite restart attempts. The default is 60 seconds.
MaxMemoryUsagePercent	(Optional) This option controls the maximum percentage of the total JVM memory that can be used by an instance before Watchdog will restart it. Note that the total memory used in this calculation does not include any memory used by native code. This option is used with the MemoryChecks option. The default is 95.
MemoryChecks	(Optional) This option controls the total count of consecutive memory checks that must be present, where the memory usage by an instance exceeds the value provided for the MaxMemoryUsagePercent option for each check, at which point Watchdog will restart it. The interval for each memory check is controlled by the CheckIntervalSeconds option. The default is -1, which disables this option.
CheckIntervalSeconds	(Optional) This option controls the time interval used by Watchdog to check the health of each instance. The default is one (1) second.
UseJMX	(Optional) This option controls whether JMX is used to monitor additional health metrics for each instance. Enabling this option lets Watchdog also monitor class loading, memory usage, garbage collection, and deadlocks in Java code for each instance. Note that enabling this option requires an additional and separate TCP/IP port for each instance so that it can be started with a JMX agent. You can enter Yes (or True) or No (or False). The default is No. Only use this option for debugging or testing purposes. Do not use this option in production mode because it causes extra overhead and it requires additional ports be used.

Options	Description
JMXPort	(Optional) This option controls the starting JMX port to use when starting each instance with a JMX agent if the UseJMX option is enabled. Note that the starting port value should consider that each additional instance that is started will try to use a continuous/incremental port number. The default starting port value is 49163.
JMXCheckIntervalSeconds	(Optional) This option controls the time interval used to run JMX checks for each instance when the UseJMX option is enabled. The default is 60 seconds.
JMXMemoryChecks	(Optional) This option controls the total count of consecutive JMX memory checks that must be present, where the memory usage by an instance exceeds the value provided for the MaxMemoryUsagePercent option for each check, at which point Watchdog will restart it. The interval for each check is controlled by the JMXCheckIntervalSeconds option. The default is -1, which disables this option.
JMXVerboseMemory	(Optional) This option controls whether Watchdog turns on verbose memory to output GC statistics for each IDS instance when the UseJMX option is enabled. You can enter Yes (or True) or No (or False). The default is No.
JMXVerboseClassLoader	(Optional) This option controls whether Watchdog turns on verbose class loading for each IDS instance when the UseJMX option is enabled. You can enter Yes (or True) or No (or False). The default is No.
WaitForShutdownSeconds	(Optional) This option controls how long Watchdog waits for an instance to shut down after it issues a shutdown command and before it terminates the instance. The default is 20 seconds.
OrderedRestartIntervalSeconds	(Optional) This option controls the interval used for restarting each of the IDS instances in a sequential/ ordered manner when the MaxTransactions or MaxUpTime options are used. Watchdog restarts one instance at a time and waits for an amount of time equal to the value specified for this option before it restarts the next one and so on until it has restarted all of them. If you set this option to less than 60 seconds, you can negatively affect performance. The default is 60 seconds.

Determining the instance number of a server

You can determine the exact instance number of the (IDS) server the rule is running on. For instance, you can use this to determine which TCP/IP port to use when IDS has to talk to the GenData process. This DSI variable can be accessed from an IDS rule:

```
IDSINSTANCE
```

The value is a character array of a zero-based value. For example, on the primary IDS the value will be zero (0), on first secondary instance it will be one (1), and so on. To get the value, the rule has to call DSILocateValue().

Categories and appenders used by Watchdog

Here are the Log4J categories and appenders used by Watchdog (see logconf.xml file included in the docserv directory):

- These mail categories and appenders are used to send email notifications during mission critical errors, such as when IDS has a fatal exception:

```
<!--Used by Watchdog to send email notifications.-->
<category name="EMAIL" additivity="false">
<priority value="ERROR"/>
<appender-ref ref="EMAIL"/>
</category>

<appender class="org.apache.log4j.net.SMTPAppender" name="EMAIL">
<param value="1" name="BufferSize"/>
<param value="10.1.20.148" name="SMTPHost"/>
<!--Comment out the SMTPUsername and SMTPPassword parameters to skip
authentication.-->
<!--
<param value="" name="SMTPUsername"/>
<param value="" name="SMTPPassword"/>
-->
<param value="support@acme.com" name="From"/>
<!--Use a comma delimited string of email addresses for To, cc and
bcc.-->
<param value="user@acme.com,user@acme.com" name="To"/>
<param value="user@acme.com,user@acme.com" name="cc"/>
<param value="user@acme.com,user@acme.com" name="bcc"/>
<param value="Error Message" name="Subject"/>
<param value="ERROR" name="threshold"/>
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern" value="%d-[%t]-%m\n"/>
</layout>
</appender>
```

- This category is used to log informational output by Watchdog:

```
<!--Used to log Watchdog informational output.-->
<category name="Watchdog.output" additivity="false">
<priority value="INFO"/>
<appender-ref ref="watchdog-stdout"/>
<appender-ref ref="watchdog-allroll"/>
</category>
```

- These categories and appenders are used to log debug and error messages by Watchdog. Change the Priority value to 'DEBUG' to log debug messages.

```
<!--Used to log Watchdog debug and error messages.-->
<category name="com.docucorp.watchdog.Watchdog" additivity="false">
```

```
 <priority value="ERROR"/>
 <appender-ref ref="watchdog-stdout"/>
 <appender-ref ref="watchdog-allroll"/>
 </category>

 <!--Logs Watchdog messages to stdout.-->
 <appender name="watchdog-stdout"
class="com.docucorp.util.logging.IDSConsoleAppender">
 <layout class="org.apache.log4j.PatternLayout">
 <param name="ConversionPattern" value="%d-[%t]-%m\n"/>
 </layout>
 </appender>

 <!--Watchdog Appender.-->
 <appender name="watchdog-allroll"
class="com.docucorp.util.logging.IDSFileAppender">
 <param name="Append" value="true"/>
 <param name="File" value="watchdog.log"/>
 <param name="Encoding" value="ISO-8859-1"/>
 <layout class="org.apache.log4j.PatternLayout">
 <param name="ConversionPattern" value="%d-[%t]-%m\n"/>
 </layout>
 </appender>
```

- These categories and appenders are used by each Watchdog instance monitor thread to log information for each instance monitored separately. Change the Priority value to 'DEBUG' to log debug messages.

```
  <!--Used to log each thread's Instance Monitor debug and error
messages.-->
 <category name="com.docucorp.watchdog.monitor.InstanceMonitor"
additivity="false">
 <priority value="ERROR"/>
 <appender-ref ref="instance-stdout"/>
 <appender-ref ref="instance-allroll"/>
 </category>

 <!--Logs Instance Monitor thread messages to stdout.-->
 <appender name="instance-stdout"
class="com.docucorp.util.logging.IDSConsoleAppender">
 <layout class="org.apache.log4j.PatternLayout">
 <param name="ConversionPattern" value="%d-[%t]-%m\n"/>
 </layout>
 </appender>

 <!--Logs Instance Monitor thread messages to separate file(s).-->
 <appender name="instance-allroll"
class="com.docucorp.watchdog.util.WatchdogFileAppender">
 <param name="Append" value="true"/>
 <param name="File" value="~THREADID.log"/>
 <param name="Encoding" value="ISO-8859-1"/>
 <layout class="org.apache.log4j.PatternLayout">
 <param name="ConversionPattern" value="%d-[%t]-%m\n"/>
 </layout>
 </appender>
```

- These categories and appenders are used to log debug and error information for IPC (Inter-Process Communication) messages between Watchdog and the instances. Change the Priority value to 'DEBUG' to log debug messages.

```
 <!--Used to log IPCConnector debug and error messages.-->
 <category name="com.docucorp.watchdog.ipc.IPCConnector"
additivity="false">
 <priority value="ERROR"/>
 <appender-ref ref="connector-stdout"/>
 <appender-ref ref="connector-allroll"/>
 </category>

 <!--Logs IPCConnector debug and error messages to stdout.-->
 <appender name="connector-stdout"
class="com.docucorp.util.logging.IDSConsoleAppender">
 <layout class="org.apache.log4j.PatternLayout">
 <param name="ConversionPattern" value="%d-[%t]-%m\n"/>
 </layout>
 </appender>

 <!--Logs IPCConnector debug and error messages to a file.-->
 <appender name="connector-allroll"
class="com.docucorp.watchdog.util.WatchdogFileAppender">
 <param name="Append" value="true"/>
 <param name="File" value="~THREADID.log"/>
 <param name="Encoding" value="ISO-8859-1"/>
 <layout class="org.apache.log4j.PatternLayout">
 <param name="ConversionPattern" value="%d-[%t]-%m\n"/>
 </layout>
 </appender>
```

A watchdog configuration file can contain multiple sections, each with its own set of options. Here are some examples:

Watchdog section

Here is an example of the Watchdog section:

```
<configuration>
    <section name="Watchdog">
        <entry name="UseJMX">No</entry>
        <entry name="JMXCheckIntervalSeconds">60</entry>
        <entry name="JMXMemoryChecks">3</entry>
        <entry name="JMXVerboseMemory">Yes</entry>
        <entry name="JMXVerboseClassLoader">Yes</entry>
    </section>
    <section version="2.2" name="DocumentServer">
        <entry name="StartCommand">java</entry>
        <entry name="StartArguments">-Djava.endorsed.dirs=lib/
endorsed -Xmx256m -Ddsimessage.debug=N -Dmarshaller.output=N -cp
.;lib/DocucorpStartup.jar -Dids.configuration=docserv.xml -
Dlogging.configuration=logconf.xml com.docucorp.startup.Startup
com.docucorp.ids.DocumentServer</entry>
        <entry name="StartDirectory">.</entry>
        <entry name="Instances">2</entry>
        <entry name="UseLoadBalancing">Yes</entry>
        <entry name="MaxInstances">10</entry>
        <entry name="IncrementCount">2</entry>
        <entry name="IdleTimeCheckIntervalSeconds">60</entry>
        <entry name="IdleTimeCheckDelaySeconds">120</entry>
```

```
            <entry name="IdleTimeChecks">2</entry>
            <entry name="MinIdleTimeSeconds">5</entry>
            <entry name="MaxIdleTimeSeconds">120</entry>
            <entry name="MaxTransactions">10000</entry>
            <entry name="MaxReportIntervalSeconds">60</entry>
            <entry name="MaxUptimeSeconds">28800</entry>
            <entry name="MaxRestarts">5</entry>
            <entry name="RestartIntervalSeconds">60</entry>
            <entry name="MaxMemoryUsagePercent">95</entry>
            <entry name="MemoryChecks">3</entry>
            <entry name="CheckIntervalSeconds">1</entry>
            <entry name="UseJMX">No</entry>
            <entry name="JMXPort">49163</entry>
            <entry name="JMXCheckIntervalSeconds">60</entry>
            <entry name="JMXMemoryChecks">3</entry>
            <entry name="JMXVerboseMemory">Yes</entry>
            <entry name="JMXVerboseClassLoader">Yes</entry>
            <entry name="WaitForShutdownSeconds">20</entry>
            <entry name="OrderedRestartIntervalSeconds">60</entry>
        </section>
    </configuration>
```

These JVM options are supported:

Option	Description
-Dwatchdog.configuration	(Optional) The name of the XML configuration file for watchdog. The default is docserv.xml.
-Dlog4j.configuration	(Optional) The name of the XML configuration file for LOG4J. The default is logconf.xml.
-Dwatchdog.prefix	(Optional) A unique string that should be used as the prefix for all Watchdog files/locks generated on disk. Use this option when running more than one Watchdog instance from the same directory.

Here are some examples:

Scenario 1 A platform contains a single CPU and the default values are used for the Instances option and for load balancing.

In this case, the default value of Instances will be two (2) and the default value of MaxInstances will also be two (2) so no load balancing will occur.

Scenario 2 A platform contains four CPUs and the default values are used for the Instances option and for load balancing.

In this case the default value of Instances is two (2) and the default value of MaxInstances is 8. The default increment count will be two (2), the default minimum idle time will be 5 seconds, and the default maximum idle time will be 120 seconds.

Load balancing will occur and Watchdog will check the idle time of any running instances every 60 seconds. If each of the instances running has an idle time that is less than 5 seconds, Watchdog deems them all busy and starts two additional instances. Watchdog then continues on to the next check interval.

These steps are repeated during each check interval until the total number of instances running reaches eight. If any of the running instances were started for the purpose of load balancing and reach an idle time greater than 120 seconds, they are shut down by Watchdog.

Scenario 3

A platform contains four CPUs and the value for the Instances option is set to 20 and the default values are used for load balancing.

In this case the value for MaxInstances will be eight but the value for the instances will be greater than the value for the maximum instances that can be reached during load balancing so no load balancing will occur.

Sending Results and Receiving Requests in Multiple Formats

The HTTP-based and queue-based messaging systems in IDS can accept messages in multiple formats and will return results in the same format as the request. This is done so IDS can communicate with third-party products that would find it difficult to produce messages in current IDS-compatible formats.

The translation is done by *marshaller* code that translates a foreign message format into the message format used internally by IDS (*com.docucorp.messaging.data.DSIMessage*).

Marshallers are Java objects that implement the interface *com.docucorp.messaging.data.marshaller.DSIMessageMarshaller*, which is documented in the SDK Reference.

Objects of the DSIMessage class hold the message variables and attachments passed in as input to IDS and the output message variables and attachments produced by processing requests in IDS.

The most important functions for marshallers are shown here:

Marshaller	Takes
marshall	A DSIMessage as an input and produces an object that is suitable for sending to a client application via messaging.
unmarshall	An object from a client application and a DSIMessage, and uses the Object to fill in data in the DSIMessage.
isType	An object from a client application and determines if it is in the format of the marshaller.

The messaging systems, both HTTP-based and queue-based, keep a list of marshallers for formats that they recognize. When a message comes in from a client application, a messaging system compares the message's format against the formats will recognizes using the isType function for each marshaller.

If there is a match the incoming data is translated into a DSIMessage with the marshallers unmarshall function and the DSIMessage, holding the request to be processed, is used as input into the main request processing in IDS. This produces an output DSIMessage holding message variables and attachments. The output DSIMessage is translated into the same format as the input message and sent back to the client application.

As a default, the HTTP-based messaging system understands the SOAP with MIME Attachments format. (See Using HTTP on page 125 for more information.) The default marshallers for the queue-based messaging system understand the SOAP with MIME Attachments format and the binary format used by IDS version 1.6 and earlier.

The queue-based messaging system currently works with WebSphere MQ, formerly known as MQSeries, and Java Message Service based queues. Both queuing systems can deliver messages as text (a Java string) and binary (a Java array of bytes), so marshallers can work with either format. The HTTP-based messaging system only recognizes text with HTTP headers, such as Content-length, so marshallers written to work with HTTP must work within these limitations.

Configuring and Deploying Marshallers

To configure IDS to recognize custom marshallers, you can add a marshallers section to either of these sections:

- 'BusinessLogicProcessor', subsection 'messaging', subsection 'queue'
- 'BusinessLogicProcessor', subsection 'messaging', subsection 'http'

For example, in the docserv.xml configuration file, in section 'BusinessLogicProcessor', subsection 'messaging', subsection 'queue' create a 'marshallers' section and add these entries:

```
<section name="marshallers">
        <entry
name="marshaller.class">com.docucorp.messaging.data.marshaller.Lega
cyByteArrayDSIMessageMarshaller</entry>
        <entry
name="marshaller.class">com.docucorp.messaging.data.marshaller.SOAP
MIMEDSIMessageMarshaller</entry>
</marshallers>
```

This sets up the queuing system to use these two formats for receiving and sending requests. If no entries are specified then IDS defaults to SOAP with attachments and legacy IDS byte format.

NOTE: If a custom marshaller list is added to the configuration file, the default marshallers are not automatically added to the list. This allows greater customization. If you want to also use the default marshallers in messaging, you must add them to the marshallers list in addition to the custom marshallers.

To deploy your marshaller code, package the Java class files in a JAR file and put it in the /lib subdirectory under where IDS was installed. The next time IDS starts your custom marshallers will be available.

Using the DSIMessage marshaller class

IDS version 2.1 added the DSIMessage marshaller class called *XSLTTemplateDSIMessageMarshaller*.

For marshalling, the marshaller starts with the usual SOAP/XML format of a DSIMessage, then uses that as the source of a XSLT transformation to convert the SOAP message to a third-party XML format.

For unmarshalling, the marshaller starts with a third-party XML format and the XSLT transformation converts it to the Docupresentment SOAP/XML format, which is then unmarshalled into a DSIMessage object.

In addition to the usual marshalling methods, the marshaller adds methods to pass in the files holding the XSLT templates, or the text of the XSLT template directly.

NOTE: This marshaller is mainly used in IDS rules, so there is no configuration-based setup of XSLT templates for marshalling and unmarshalling. You set up the XSLT with Java methods in the XSLTTemplateDSIMessageMarshaller class.

These methods set up XSLT templates for the marshaller:

- public void setMarshallerText(String text)

  This method sets the XSLT text for marshalling messages (converting to a foreign format).

- public void setMarshallerFilename(String filename)

  This method sets the file that holds the XSLT for marshalling messages (converting to a foreign format). The file is loaded into the marshaller filter.

- public void setUnmarshallerText(String text)

  This method sets the XSLT text for unmarshalling messages (converting from a foreign format).

- public void setUnmarshallerFilename(String filename)

  This method sets the file that holds the XSLT for unmarshalling messages (converting from a foreign format). The file is loaded into the unmarshaller filter.

Logging and Tracing

There are several ways you can configure IDS to log messages. For instance, you can configure IDS to:

- Log only events based on their severity, from debug messages to fatal errors.
- Control where logging messages are sent, whether they go to files, the Windows event logger, into emails, and so on.
- Include and format the relevant information, such as time of day, elapsed time since IDS was started, where the message came from, which thread the code is currently running in, and so on.

Logging in IDS is based on the Log4j logging library. A complete description of Log4j's capabilities is available at

http://jakarta.apache.org/log4j

IDS logging is configured by the file specified in the Java system property 'logging.configuration'. If this property is not set, the default is to look for the logconf.xml file in the current directory. This file is checked periodically for changes by IDS when it is running, so it is possible to change logging options while IDS is running. IDS does not need to restart when a logging change is made.

Severity levels

Logging messages have a severity level assigned to them. This is used to determine if and when a logging message is written. From least to most severe, the severity levels are:

- DEBUG
- INFO
- WARN
- ERROR
- FATAL

When setting up logging you can decide what kind of messages to receive by picking a severity level. Any messages at that severity level and those more severe are output. For example, if you choose a severity level of WARN, only WARN, ERROR, or FATAL messages are sent, while DEBUG and INFO messages are suppressed. For diagnosing problems, the severity level would be set to DEBUG to produce more messages.

Logging categories

What messages to log and where to log them is determined by logging categories or *loggers*. Categories are based on a hierarchy of names or words separated by periods. As an example, *DocumentServer* would be a parent category to *DocumentServer.output* and *DocumentServer.error*. When a parent category is assigned a severity level, all children categories inherit it as their default. Children categories can override these defaults.

For example, if the *DocumentServer* category's severity level is set to WARN and the *DocumentServer.output* category's severity is set to INFO, then only *DocumentServer* and *DocumentServer.error* will use WARN.

The *DocumentServer.output* category is where normal runtime messages for IDS are sent, such as the startup message and how many transactions were completed at shutdown or restart time. Common errors encountered during configuration and rule setup are sent to the *DocumentServer.error* category. Since these messages are in logging categories, they can be sent to multiple places in addition to being printed on the console. There are other categories that have debugging messages and you may be asked to activate these by Support.

NOTE: For additional information, see Using Logging Categories to Access the Internal Format of Requests on page 94.

Logging appenders

The destinations for logging messages are known as *appenders*, since new logging messages are appended to the end of the file, event log, and so on. The most common appenders are for the console and for files.

A category can send messages to multiple appenders. Like severity levels, appenders inherit from parent categories, but unlike severity levels, appenders are added to the parent appenders and do not override the parent category's settings. This inheritance can be turned off, as shown in the logging example.

Logging formats

In addition to the logging message itself, other information can be configured to be output with the logging message, such as the date and time the logging occurred, the severity level of the message, and so on. In Log4j the formatting strings are called *conversion patterns*, which you will see in the logging example. You can arrange the formatting commands in any order. Text not part of a formatting command is sent verbatim, so you can use commas, dashes, and other characters to separate the formatted text.

You may want to set up conversion patterns for different appenders. For example, you may want to include the date and time on a message going to a file but excluding it from a message going to the Windows Event Log, since the message's date and time are logged by the Event Logger.

Here are some of the common formatting commands:

Parameter	Description
%m	The actual message being logged.
%n	A system-dependent newline character.
%c	The category of the message.
%d	Date and time, down to the millisecond, when the message was generated.
%r	Elapsed time, in milliseconds, since the application was started.
%p	Severity level (priority) of the message generated.
%t	The name of the Java thread that generated the message.

Logging example

This is an example logging configuration file. This table shows how the various categories output messages:

Category	Outputs messages
DocumentServer	With a WARN security level or higher
DocumentServer.output	With an INFO security level or higher
DocumentServer.output	That go to the console
DocumentServer.error	That go to the console, a logging file, and to the Windows Event Logger.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration debug="false">

    <!-- An appender that writes messages to a file. -->
    <appender name="rollfile"
class="org.apache.log4j.RollingFileAppender">
        <param name="Append" value="true" />
        <param name="File"   value="docserver.log" />
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern"
                   value="%-5p [%t] %r: %c{1} - %m\n"/>
        </layout>
    </appender>

    <!-- An appender that writes messages to the console. -->
    <appender name="stdout"
class="org.apache.log4j.ConsoleAppender">
            <layout class="org.apache.log4j.PatternLayout">
          <param name="ConversionPattern" value=" %d %-5p [%t]: %c{1}
%m\n"/>
            </layout>
    </appender>

   <!-- An appender that writes messages to the Windows Event Log. -->
    <appender name="ntevent"
class="org.apache.log4j.nt.NTEventLogAppender">
            <layout class="org.apache.log4j.PatternLayout">
        <param name="ConversionPattern" value=" Docupresentment
Server: %m\n"/>
            </layout>
    </appender>

    <!-- The parent category for outputs and errors. -->
    <category name="DocumentServer">
      <priority value="WARN" />
      <appender-ref ref="stdout" />

      <appender-ref ref="rollfile" />
      <appender-ref ref="ntevent" />


    </category>
```

```
    <!-- The "standard output" for DocumentServer - where regular
messages go. -->
    <category name="DocumentServer.output" additivity="false">
      <priority value="INFO" />
      <appender-ref ref="stdout" />
    </category>

    <!-- A debugging category, will be turned on and off. -->
    <category name="com.docucorp">
      <priority value="WARN" />
      <appender-ref ref="stdout" />

    </category>

    <root>
    </root>
</log4j:configuration>
```

NAMING LOGGING MESSAGES

IDS includes a Log4j Appender class lets you control the naming of the files logging messages are written to. The full name of the class is:

```
com.docucorp.ids.serverutils.IDSFileAppender
```

When this class is used as part of the logging setup, you can use IDS-specific variables. The IDS-specific variables are:

Variable	Description
~INSTANCE	The *instance number* of the server being run. A primary server has an instance number of zero (0) and secondary instances are numbered starting at one (1). For example, running three instances of IDS, one primary and two secondary, the instances are numbered 0, 1, and 2.
~SERVERGUID	A IDS GUID identifier unique to each IDS instance but not recycled upon restart. When IDS is started, the primary instance and any secondary instances are assigned a unique identifying string. These strings are assigned to the same instance of IDS if it is shut down and restarted. The GUID strings remain the same until the number of instances is changed in the docserv.xml configuration file.
~THREADID	Each thread in IDS is given a name. You can use this name as part of the file name. Since the same code may run in different threads during the lifetime of an IDS session, and since the thread ID can be printed during a logging message, you would seldom need to use this option. An exception is for debugging the HTTP subsystem of IDS, where each HTTP message is run in a pool of threads.
~UPTIME	The date and time of when the instance of IDS was started.
~LASTRESTART	The date and time of the most recent restart of this instance of IDS.
~CURRENTTIME	The current date and time. The time can be measured down to the millisecond so this option is handy for debugging time-critical and performance issues.

The ~UPTIME, ~LASTRESTART, and ~CURRENTTIME options are followed by a formatting parameter which ends with a semicolon (;). The formatting options are based on Java's SimpleDateFormat class. You can find full documentation for these formatting options at:

http://java.sun.com/j2se/1.3/docs/api/java/text/SimpleDateFormat.html

The formatting characters are:

Symbol	Meaning	Presentation	Example
G	era designator	Text	AD
y	year	Number	1996
M	month in year	Text & Number	July & 07
d	day in month	Number	10
h	hour in am/pm (1~12)	Number	12
H	hour in day (0~23)	Number	0
m	minute in hour	Number	30
s	second in minute	Number	55
S	millisecond	Number	978
E	day in week	Text	Tuesday
D	day in year	Number	189
F	day of week in month	Number	2 (2nd Wed in July)
w	week in year	Number	27
W	week in month	Number	2
a	am/pm marker	Text	PM
k	hour in day (1~24)	Number)	24
K	hour in am/pm (0~11)	Number	0
z	time zone	Text	Pacific Standard Time
'	escape for text	Delimiter	
''	single quote	Literal	'

Here are some sample formats:

Sample	Explanation
MM-dd-yyyy	Two digits for the month, two digits for the day, and four digits for the year.
MMMM-dd-yyyy	The month spelled out, two digits for the day, and four digits for the year.
yyyyMMddHHmmssSSSS	A 4-digit year, 2-digit month, 2-digit day, 2-digit hours in day, 2-digit minutes in day, 2-digits seconds in a minute, and 4-digit milliseconds in a second. In this format, sorting alphabetically is the same as sorting by date.

As a Log4j Appender, the IDSFileAppender has several parameters that are set in the logging configuration file, usually named *logconf.xml*. The parameters are:

Parameter	Description
File	The name of the file to write. This can be a static file name or a dynamic file name created using the above options.
Append	If True, the next logging message is appended to the end of the file. If false, the file is first erased. The default is True.
Close	If True, the file is closed after the message is written. This can be useful if you want to edit, move, or delete the file while IDS is still running. The default is False.

If the name of a file changes between two logging calls, for example if you are using ~LASTRESTART and IDS is restarted, or if you are using ~CURRENTTIME and the time changes by a sufficient amount, the old file is automatically closed.

Using combinations of Append and Close can produce different effects. With Append as False and Close as True, only one logging message will be in the file at any time. This can be handy when debugging messages passed in and out of IDS.

If Append is True and Close is False, the file acts like a regular FileAppender. In fact, if you are running only one instance of IDS and you do not want to change the file name, use the regular FileAppender since it is slightly faster than the IDSFileAppender. IDSFileAppender has to re-evaluate the file name for each logging message. Setting Append to True and Close to True gives you an ever-growing file that you can move, delete, edit, and so on.

Here are some examples:

The IDSFileAppender is used in the *appender* elements in the logging configuration.

```
<appender name="multiinstance"
class="com.docucorp.ids.serverutils.IDSFileAppender">
<param name="Append" value="true" />
<param name="File" value="server-~INSTANCE .log" />
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern"
value="%-5p [%t] %r: %c{1} - %m\n"/>
</layout>
</appender>
```

This produces a separate log file for each instance of IDS. If there are a total of three instances running, a primary and two secondaries, these log files are created:

- server-0.log
- server-1.log
- server-2.log

```
<appender name="everyhour"
class="com.docucorp.ids.serverutils.IDSFileAppender">
<param name="Append" value="true" />
<param name="File" value="server-~CURRENTTIME MMddHH;.log" />
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern"
value="%-5p [%t] %r: %c{1} - %m\n"/>
</layout>
</appender>
```

This produces a log file and which is written to until the hour changes. The system then starts writing to a new file. The date/time formatting is for the month, day, and hour, so on October 15, from noon until 1 pm, logging entries are written to the *server-101512.log* file. From 1 pm to 2 pm, logging entries are written to the *server-101513.log* file, and so on.

USING LOGGING CATEGORIES

To make it easier to debug problems in IDS, you can use logging categories to sort messages IDS receives from client programs and messages it sends to client programs. You can have the system treat all messages the same or distinguish between messages from message queues (WebSphere MQ or JMS) and messages from HTTP.

Combining the use of the message categories with options in the IDSFileAppender provides the same functionality as the *send.msg* and *receive.msg* message debugging of IDS version 1.8, but also allows other options.

The categories are as follows:

- SendMessage.queue
- ReceiveMessage.queue
- SendMessage.http
- ReceiveMessage.http

As with other Log4j categories, the categories are hierarchical, so using category names *SendMessage* and *ReceiveMessage* will use the same category for both queue and HTTP-based messages.

Since the messages are handled as Log4j categories, they can have all the destinations of other categories, such as files, the NT Event logger, email, and so on.

Here are some examples. When looking at the examples, remember that for each request, a message is first received by IDS then a message is sent.

This combination of categories and appenders gives the same behavior as in IDS version 1.8. When the categories are set to *DEBUG*, any received messages are placed in the receive.msg file. Any sent messages are placed in the send.msg file. When new messages are processed, either by queues or HTTP, they are placed in the receive.msg and send.msg files.

```
<appender name="receivemessage"
class="com.docucorp.ids.serverutils.IDSFileAppender">
<param name="Append" value="false" />
<param name="File" value="receive.msg" />
<param name="Close" value="true" />
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern" value="%m"/>
</layout>
</appender>

<appender name="sendmessage"
class="com.docucorp.ids.serverutils.IDSFileAppender">
<param name="Append" value="false" />
<param name="File" value="send.msg" />
<param name="Close" value="true" />
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern" value="%m"/>
</layout>
</appender>

<category name="ReceiveMessage">
<priority value="DEBUG" />
<appender-ref ref="receivemessage" />

</category>

<category name="SendMessage">
<priority value="DEBUG" />
<appender-ref ref="sendmessage" />

</category>
```

This set of categories and appenders puts the received messages and sent messages in the same file, with one file for each instance of IDS that is running. Since Append is False for receiving and Append is True for sending, this file is overwritten for each receive/send pair. The header *Received*: is added in front of the received message and *Sent*: is placed in front of the sent message.

```
<appender name="receivecombined"
class="com.docucorp.ids.serverutils.IDSFileAppender">
<param name="Append" value="false" />
<param name="File" value="combined-~INSTANCE .msg" />
<param name="Close" value="true" />
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern" value="Received:%n%m%n"/>
</layout>
</appender>

<appender name="sendcombined"
class="com.docucorp.ids.serverutils.IDSFileAppender">
<param name="Append" value="true" />
```

```
<param name="File" value="combined-~INSTANCE.msg" />
<param name="Close" value="true" />
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern" value="Sent:%n%m"/>
</layout>
</appender>

<category name="ReceiveMessage">
<priority value="DEBUG" />
<appender-ref ref="receivecombined" />

</category>

<category name="SendMessage">
<priority value="DEBUG" />
<appender-ref ref="sendcombined" />

</category>
```

In this example you distinguish between messages handled by the queues and messages handled by HTTP. The queue messages are placed in receive.msg and send.msg, but since the HTTP messages are handled simultaneously by multiple threads, the HTTP messages include the thread ID in the file names.

The system also notes the categories and sub-categories. In the categories with HTTP, *additivity* is set to False, meaning the HTTP categories should not use appenders from the SendMessage and ReceiveMessage categories.

```
<appender name="receivemessage"
class="com.docucorp.ids.serverutils.IDSFileAppender">
<param name="Append" value="false" />
<param name="File" value="receive.msg" />
<param name="Close" value="true" />
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern" value="%m"/>
</layout>
</appender>

<appender name="sendmessage"
class="com.docucorp.ids.serverutils.IDSFileAppender">
<param name="Append" value="false" />
<param name="File" value="send.msg" />
<param name="Close" value="true" />
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern" value="%m"/>
</layout>
</appender>

<appender name="receivehttp"
class="com.docucorp.ids.serverutils.IDSFileAppender">
<param name="Append" value="false" />
<param name="File" value="receive-~THREADID .msg" />
<param name="Close" value="true" />
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern" value="%m"/>
</layout>
</appender>
```

```
<appender name="sendhttp"
class="com.docucorp.ids.serverutils.IDSFileAppender">
<param name="Append" value="false" />
<param name="File" value="send-~THREADID .msg" />
<param name="Close" value="true" />
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern" value="%m"/>
</layout>
</appender>


<category name="ReceiveMessage">
<priority value="DEBUG" />
<appender-ref ref="receivemessage" />

</category>

<category name="SendMessage">
<priority value="DEBUG" />
<appender-ref ref="sendmessage" />

</category>

<category name="ReceiveMessage.http" additivity="false">
<priority value="DEBUG" />
<appender-ref ref="receivehttp" />

</category>

<category name="SendMessage.http" additivity="false">
<priority value="DEBUG" />
<appender-ref ref="sendhttp" />

</category>
```

Logging Information about Requests

IDS lets you store information about a request in a database for later viewing or processing. IDS logs information about a request (transaction) in any Java Database Connectivity (JDBC) compliant database. You can find additional information about JDBC at

http://java.sun.com/products/jdbc/

On Windows computers, Open Database Connectivity (ODBC) is available, and Java has built-in drivers to ODBC connections.

Request logging automatically rolls over and starts new database tables on a daily or weekly basis and you can configure IDS to automatically delete old database tables. The configuration file can specify how many days or weeks of log tables to keep when purging old tables.

NOTE: Set up a separate database for IDS transaction logs. When purging old tables, *all* tables that do not qualify as the most recent transaction logs are deleted from the database.

IDS lets you use a browser to display request logs and sort and filter the requests. You can display logs from just a single database table or from multiple tables.

When IDS sends result messages back to client programs, it includes these message variables:

Variable	Description
IDSHOSTNAME	Contains the host name of the machine running IDS.
IDSGUID	Cntains the unique ID for the running instance of IDS.

With this information you can track a message back to a particular instance of IDS running on a particular machine, even if all messages are logged to a common place.

Request logging configuration

To add this capability, go to the DOCSERV.XML configuration file and find the BusinessLogicProcessor section. Then create a TransactionLogDatabase subsection, as shown here:

```
<section name="TransactionLogDatabase">
    <entry name="class">sun.jdbc.odbc.JdbcOdbcDriver</entry>
    <entry name="URL">jdbc:odbc:TRAN_LOG</entry>
    <entry name="userid">sa</entry>
    <entry name="password"></entry>
    <entry name="time.type">weekly</entry>
    <entry name="time.count">2</entry>
    <entry name="time.startofweek">monday</entry>
    <entry name="close.database">N</entry>
    <section name="columns">
        <entry name="Reqtype">REQTYPE</entry>
        <entry name="UserID">USERID</entry>
        <entry name="Password">PASSWORD</entry>
    </section>
</section>
```

Parameter	Description
class	The JDBC Java class that connects to the database. In this example, we are using Java's built in ODBC connectivity in Windows.
URL	The JDBC-based name of the database that tables will be written into. This will vary for different JDBC drivers, but somewhere will include the database name, in this case TRAN_LOG. Consult your JDBC driver documentation on setting up this URL.
userid	The user name for logging in to the database.
password	The password for the specified user name.

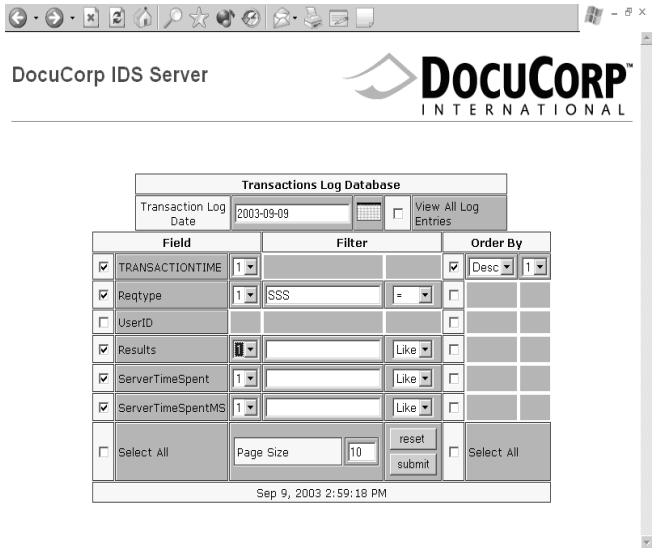
Parameter	Description
time.type	Either *daily* or *weekly*, specifying how often to roll over into a new database table.
time.count	How many of the most recent tables in the database to keep when deleting old tables. This will be either the most recent days or weeks worth of information, depending on the time.type setting.
time.startofweek	When time.type is weekly, this specifies what day of the week to start a new database table.
close.database	Whether or not to close the database after writing a transaction to the transaction database. Setting this to Y or N will depend on how many database connections are available to the database and how often transactions are logged.
columns	This subsection holds any number of entries specifying the column names in the database. The entry name is the name of the column that will be used in the database. The entry value is the name of the message variable in the output that will be written in the column.

Accessing the transaction database through IDS

IDS includes requests and HTML pages to let a user view the transaction log database from a browser. To begin, start a web browser and go to this URL

```
http://localhost:49152/request?REQTYPE=LOGMETADATA
```

localhost:49152 refers to the IP address and port that IDS is set up for HTTP messaging. You will see a screen similar to this:



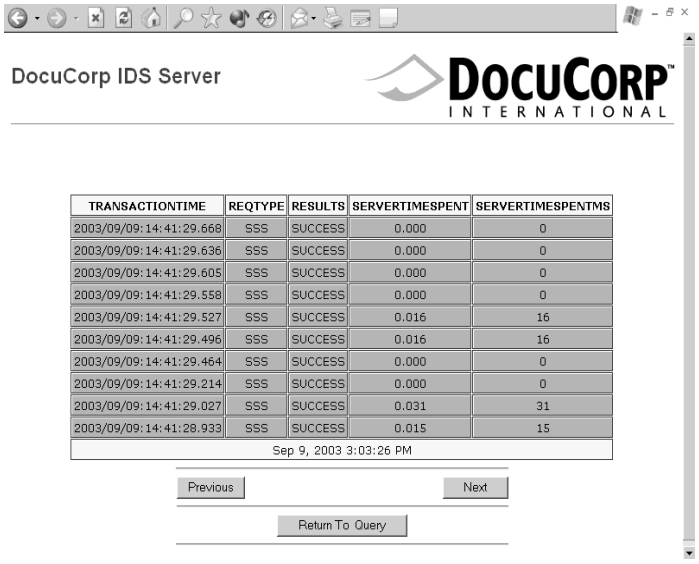
The Transaction Log Date field specifies which day's worth of records that results will be pulled from. Use the calendar button next to this field to pick another date. If data is being collected in weekly mode, you can check the View All Log Entries field to select data from all the records in a week's database table, not just the data for one day.

The Field column lists all the data fields you can display. You can choose individual fields or select them all. The numeric drop-down box next to each field name can be used to select the display priority. The first fields to display will have a display priority of 1, then all fields with a display priority of 2, and so on.

The Filter field lets you filter each field by a value. Enter a value for filtering, then choose an operation to perform on the filter.

Use the Order By column to sort a column's information, ascending or descending. The numeric drop-down box picks the ordering priority, with 1 having the highest priority.

After you pick the settings and click Submit, a screen similar to this one appears:



All the previously selected columns appear. Use the Previous and Next buttons to move through the selected records. Click Return to Query to return to the selection page.

Accessing the transaction database directly

Since the transaction database is a regular JDBC compliant database it can be accessed through any database program that uses JDBC or the database's native format. This section explains the naming and data format conventions used in the transaction database.

The database name is just the name of the database set up by you or your database administrator.

The names of tables in the database will all start with the string *TRANLOG* followed by the date that data is first written into the table. The format is shown here:

```
TRANLOGyyyymmdd
```

Parameter	Description
TRANLOG	All tables start with this string.
yyyy	Year
mm	Month (01 - 12)

Parameter	Description
dd	Day of the month (01 - 31)

For example, a table starting on September 1, 2003 would be named:

```
TRANLOG20030901
```

Each row in the database table begins with a column named *TRANSACTIONTIME*, a 23-character string that is the key for the row. In SQL, this is known as a *CHAR(23)*. The string is the time of the transaction, to millisecond precision, formatted in a way that sorting the table on the column sorts the logs by date and time recorded. The format is:

```
yyyy/mm/dd:hh:nn:ss.xxx
```

Parameter	Description
yyyy	Year
mm	Month (01 - 12)
dd	Day of the month (01 - 31)
hh	Hour in the day (00 - 23)
nn	Minute in hour (00 - 59)
ss	Second in minute (00 - 59)
xxx	Millisecond in second (000 - 999)

The names of the other columns in the table row are generated from the IDS configuration, and each column type is a variable-length string, known in SQL as a *VARCHAR(255)*.

QUERYING TRANSACTION INFORMATION

You can use the getMetaData and the QueryTranLogs Java rules to query transaction information. These rules let you monitor information such as the amount of server time spent for each request, the requests that failed, user IDs, and passwords.

These rules use the TransactionLogDatabase section in the docserv.xml file to build a connection to the log database.

NOTE: See Logging Information about Requests on page 86 for information on how to set up the transaction log database.

You can use the logmetadata and logrecords XSL templates with the version 2.***x*** HTTP server to query transaction information from the log database. Information can be found by matching a table name based on the date specified in the web user interface, the time type (daily or weekly) for the table, and the starting day for the table.

Here is an example of a URL that uses the web interface:

```
http://localhost:49152/request?reqtype=logmetadata
```

getMetaData

This rule displays meta-data about the log database. The rule returns the field count, which is the number of fields available in each table. This information is set up in the TransactionLogDatabase section of the docserv.xml file. The rule also returns each of the field names that correspond to the field count and the TransactionLogDatabase section.

Field names are returned as attachment variables field*1* through field*n*, where *n* is the field count. The getMetaData rule also returns the table time type being used for the transaction log database which can be daily or weekly.

This information is set up in the TransactionLogDatabase section of the XML file. The rule also returns a table count which corresponds to the number of tables currently present in the log database. Furthermore, the rule also returns each table name as attachment variables table*1* through table*n*, where *n* corresponds to the table count value.

Input attachments

Variable	Description
REQTYPE	LOGMETADATA

Output attachments

Variable	Description
FIELDCOUNT	The number of fields in each table within the transaction log database.
FIELD1...FIELDn	The name of each field in the transaction log database tables, where *n* corresponds to FIELDCOUNT.
TABLETIMETYPE	The table time type, a setting in the TransactionLogDatabase section of the docserv.xml file.
TABLECOUNT	The number of tables present in the transaction log database.
TABLE1...TABLEn	The name of each table present in the log database, where *n* corresponds to TABLECOUNT.

NOTE: Use this rule with the logmetadata xslt template and the HTTP server that comes with version 2.***x***. Keep in mind you must first set up logging (see Logging Information about Requests on page 86) before you can use this rule.

Here is a sample URL:

```
http://localhost:49152/request?reqtype=logmetadata
```

Here is a sample request type:

```
<section name="ReqType:LOGMETADATA">
    <entry name="function">atcw32->;ATCLogTransaction</entry>
    <entry name="function">java;com.docucorp.ids.rules.
IDSTransactionRule;;static;reportTimes;INCLUDEMS</entry>
    <entry name="function">atcw32->;ATCLoadAttachment</entry>
```

```
    <entry name="function">irlw32->;IRLCopyAttachment</entry>
    <entry name="function">java;com.docucorp.ids.rules.
TransactionLogRSRule;;transaction;getMetaData;</entry>
</section>
```

QueryTranLogs

Use this rule to query the transaction log database and return a recordset of transactions.

NOTE: See Logging and Tracing on page 77 for information on how to set up the transaction log database.

Input attachments

Variable	Description
REQTYPE	The request type that contains the QueryTranLogs rule. This should be LOGRECORDS when you are using the logrecords.xsl template.
SQLCMD	(Optional) An SQL select statement for the query.
SQLTABLENAME	(Optional) The table name of the database log that is to be queried, such as TRANLOG20030121.
SQLFIELDS	(Optional) A comma delimited list of selection fields to use for to the query. Here is an example: `reqtype,results,userid,password`
SQLWHERE	(Optional) The filters specified for the query. Here is an example: `reqtype like 'sss%' and userid = 'FORMAKER' and password <> 'I'`
SQLORDERBY	(Optional) The sort order criteria to use in the query, such as: `reqtype asc, password desc, userid asc`
SQLTIMEOUT	(Optional) The SQL connection time-out, specified in seconds. The default is 300 seconds.
SQLABSOLUTEPAGE	(Optional) The current page to display for the query. This is used in recordset paging. The default is one (1).
SQLPAGESIZE	(Optional) The number of records to return for a query. The default is 10.
SQLEXACTMATCH	(Optional) Enter Yes if the system should only display the records for the date specified. Enter No if you want it to display all the records in the current table.

Output attachments

Variable	Description
SQLCMD	The generated SQL query string.

Variable	Description
SQLPAGESIZE	The number of records returned for the query.
FIRSTPAGE	Returns True if the records returned for the query are the first page. Otherwise, the system returns False.
LASTPAGE	Returns True if the records returned for the query are the last page. Otherwise, the system returns False.
SQLSELECTION FIELDCOUNT	The number of fields used in the query.
SELECTIONFIELDS	A rowset containing the names of each of the fields returned by the query.
RECORD1....RECORDn	Rowsets for each record returned in the query, where n is the last record returned in the query. (this value should be equal to that of SQLPAGESIZE if LASTPAGE is False)

NOTE: Use this rule with the logrecords xslt template and the HTTP server that comes with version 2.0 or higher.

By default the rule tries to use the SQLCMD input attachment. If it is not found or its value is omitted, the rule then tries to build a select statement using the SQLTABLENAME, SQLFIELDS, SQLWHERE, and SQLORDERBY input attachment variables.

Here is a sample request type:

```
<section name="ReqType:LOGRECORDS">
    <entry name="function">atcw32->;ATCLogTransaction</entry>
    <entry name="function">java;com.docucorp.ids.rules.
IDSTransactionRule;;static;reportTimes;INCLUDEMS</entry>
    <entry name="function">atcw32->;ATCLoadAttachment</entry>
    <entry name="function">irlw32->;IRLCopyAttachment</entry>
    <entry name="function">java;com.docucorp.ids.rules.
TransactionLogRSRule;;transaction;QueryTranLogs;</entry>
</section>
```

MONITORING PERFORMANCE STATISTICS

IDS lets you use Java Management Extensions (JMX) to monitor performance data. This lets external JMX monitoring applications track performance data and is similar to SNMP support.

JMX support is built-in to Java versions 1.5 and higher, but IDS includes JMX libraries that let you monitor IDS with Java 1.4 versions. For Java 1.4 versions, IDS uses the JMX Message Protocol (JMXMP), based on Sun's Reference Implementation of the JMX Remote API.

In the docserv.xml configuration file, in the DocumentServer section, create a JMX subsection, as shown here:

```
<section name="DocumentServer">
    <section name="JMX">
        <entry name="Enabled">yes</entry>
        <entry name="JMXMPPort">9875</entry>
    </section>
</section>
```

The *Enabled* entry determines whether JMX support is enabled. The default is No.

The *JMXMP* entry is the TCP/IP port where the JMX Messaging Protocol (JMXMP) will be supported. Each instance of IDS will have its own JMXMP port starting with this number. If you omit this entry, JMXMP support is not enabled.

In addition to compiling application-wide statistics automatically, IDS can also track performance times for a specific request or for a specific function in a request. You can set this up in the configuration. In the DocumentServer section, create a StatisticsMonitors subsection, as shown here:

```
<section name="StatisticsMonitors">
    <entry name="Monitor">SSS</entry>
    <entry name="Monitor">SSS/5</entry>
</section>
```

To monitor an entire request, enter the request's name, such as *SSS*. To monitor a specific function within a request, specify the request's name, a slash (/), and the line number of the function inside the request, such as *SSS/5*.

For each monitor you list, IDS tracks the most recent timings for the Initialize, Run Forward, Run Reverse, and Terminate messages, and the total time for all messages run.

For general information about JMX, see:

http://java.sun.com/products/JavaManagement/

Generating a Logging Configuration File

IDS includes a LogConfConvert.xsl template which you can use to generate a logconf.xml file from the docserv.xml file.

The template takes into account the XMLMessage, XMLMessageAppend, TransactionTime, and RuleTime options in the Debug section of the docserv.xml file during the generation of the logconf.xml file.

A logconfcovert script file is also included in the docserv directory which you can use to run the command necessary for the conversion process, as shown here:

Windows

```
java -cp .;.\lib\DocucorpUtil.jar com.docucorp.util.XslTransformer
source=docserv.xml template=LogConfConvert.xsl output=logconf.xml
```

UNIX

```
java -cp .:./lib/DocucorpUtil.jar com.docucorp.util.XslTransformer
source=docserv.xml template=LogConfConvert.xsl output=logconf.xml
```

Using Logging Categories to Access the Internal

FORMAT OF REQUESTS

To make it easier to debug problems which can occur when translating requests and results, the system includes logging categories you can use to see the internal data format of received requests and sent results. The categories are as follows:

- DSIMessage.ReceiveMessage.queue
- DSIMessage.SendMessage.queue
- DSIMessage.ReceiveMessage.http
- DSIMessage.SendMessage.http

As with other Log4j categories, the categories are hierarchical so using category names DSIMessage.SendMessage and DSIMessage.ReceiveMessage will use the same category for both queue and HTTP-based messages.

Since the messages are handled as Log4j categories, they can have all the destinations of other categories, such as files, the NT Event logger, email, and so on.

A combination of categories and appenders will add the internal data format of messages to the end of receive.msg and send.msg files, adding useful information about how the messages are translated.

Use the following Log4j appenders to add the internal data information to the receive.msg and send.msg files:

```
<appender class="com.docucorp.ids.serverutils.IDSFileAppender"
name="dsireceivemessage">
<param value="true" name="Append"/>
<param value="receive.msg" name="File"/>
<param value="true" name="Close"/>
<layout class="org.apache.log4j.PatternLayout">
<param value="%m" name="ConversionPattern"/>
</layout>
</appender>
<appender class="com.docucorp.ids.serverutils.IDSFileAppender"
name="dsisendmessage">
<param value="true" name="Append"/>
<param value="send.msg" name="File"/>
<param value="true" name="Close"/>
<layout class="org.apache.log4j.PatternLayout">
<param value="%m" name="ConversionPattern"/>
</layout>
</appender>
```

Use these appenders with the following Log4j categories:

```
<category name="DSIMessage.ReceiveMessage.queue">
<priority value="DEBUG"/>
<appender-ref ref="dsireceivemessage"/>
</category>
<category name="DSIMessage.SendMessage.queue">
<priority value="DEBUG"/>
<appender-ref ref="dsisendmessage"/>
</category>
```

CONFIGURING IDS

The IDSConfig program lets you perform the following tasks to make it easier to configure IDS:

- Running IDSConfig on page 97
- Creating New Files on page 97
- Adding Nodes on page 97
- Adding Nodes with Text on page 97
- Editing Nodes on page 98
- Copying Nodes on page 98
- Moving Nodes on page 98
- Adding Attributes on page 98
- Adding Comments on page 99
- Adding Text on page 99
- Adding a Request or Function on page 99
- Adding an IDS Function on page 99
- Converting DOCSERV.INI or DOCCLIENT.INI Files into XML Format on page 100
- Adding a Section or Entry on page 100
- Locating Text on page 100
- Importing Configuration Information on page 100
- Configuring MQSeries Buffer Sizes on page 101
- Testing File Transmission on page 102

Running IDSConfig

To run the IDSConfig program, use the syntax shown below:

```
java -classpath <path to xerces.jar>;<path to xalan.jar>;<path to
DocuCorpUtil.jar>;<path to idsconfig.jar>
com.docucorp.idsconfig.idsconfig
```

Here is an example:

```
java.exe -classpath
d:\jars\xerces.jar;d:\jars\xalan.jar;d:\jars\DocuCorpUtil.jar;
d:\jars\idsconfig.jar; com.docucorp.idsconfig.idsconfig
```

NOTE: To run IDSConfig, you must have idsconfig.xml and idsrules.xml stored in the working directory.

Creating New Files

To create a new file, follow these steps:

1. Select the File, New option.
2. Select the appropriate type of configuration file: client or server.

Depending on the type of configuration you chose, the system creates a new XML file base or template file named either docservtp.xml or docclienttp.xml.

Adding Nodes

Follow these steps to add a node:

1. Choose one of these options:
   - Right click the parent node to which you want to add a child node.
   - Select the parent node, then choose the Edit, Add Node option.
   - Select the parent node, then click the Add Node button.
   - Select the parent node, then press INSERT.
2. Enter the name you want to assign to the node and click Ok.
3. Enter *Text* if you want to create a Text node. Otherwise, leave it blank.

Adding Nodes with Text

Follow these steps to add a node with text:

1. Choose one of these options:
   - Right click the parent node to which you want to add a child node.
   - Select the parent node, then choose the Edit, Add Node option.
   - Select the parent node, then click the Add Node button.

- Select the parent node, then press INSERT.

2 Enter the name you want to assign to the node and click Ok.

3 Enter *Text* and click Ok.

Editing Nodes

Follow these steps to edit a node:

1 Click once to select the node. The system highlights the node.

2 Press F2 or click again to edit the node.

3 Press ENTER when finished or press ESC at any time to cancel editing.

Copying Nodes

Follow these steps to copy a node:

1 Click to select the node you want to copy.

2 Press and hold the CTRL key while dragging the source node onto the destination node. Drop the source node by releasing the mouse button.

The system copies the source node and adds it as a child of the destination node.

Moving Nodes

There are two ways to move nodes:

Move as a child node

1 Select the node you want to move.

2 Drag and drop the source node onto the destination node.

The system adds the source node as a child of destination node.

Move as previous node

1 Select the node you want to move.

2 Press and hold the SHIFT key, then drag and drop the source node onto the destination node.

The system inserts the source node before the destination node.

Adding Attributes

Follow these steps to add attributes:

1 Right click the attribute header.

2 Select the Add Attribute option.

3 Enter the value of the attribute and press ENTER.

Adding Comments

Follow these steps to add comments:

1. Choose one of these options:
   - Right click the node to which you want to add a comment and choose Add comment from the popup menu
   - Select the node, then click the Add comment button.
   - Select the node, then press CTRL+M.
2. Enter your comment and click Ok when you are finished.

Adding Text

Follow these steps to add text:

1. Right click the node to which you want to add text.
2. Select the Add Text option.
3. Enter the text. Press ENTER when finished or ESC to cancel.

Adding a Request or Function

The IDSConfig program provides a quick way to add request types and functions. Follow these steps:

1. Right click on the Configuration and select the Add Request option.
2. Enter the name of the request type and click Ok.
3. Right click on the section name you just added and select the Add Function option.
4. Type in the name of the function you want to add and press ENTER.

The new function is added to the request type node.

Adding an IDS Function

The IDSConfig program provides a quick way to add an IDS function.

NOTE: Be sure to store the functions you add in the idsrules.xml. file.

1. Right click on the node to which you want to add functions and select Add IDSFunction. The Add IDS Function window appears.
2. Use the SHIFT and CTRL keys to select multiple functions in the Basic Functions area. Once you have selected the functions you want to add, drag them into the Destination area or click the >> button.

NOTE: You can rearrange the order of the functions in Destination area using your mouse.

3 Click Ok when you are finished.

Converting DOCSERV.INI or DOCCLIENT.INI Files into XML Format

You can use IDSConfig to convert a DOCSERV.INI or DOCCLIENT.INI file into an XML format file.

1 Select the File, Convert INI file option.

2 Enter the following information:

- The name of the INI file you want to convert
- The name you want assigned to the XML output file
- The name of the XSL file (DocumentClientConvert.xsl or DocumentServerConvert.xsl) and the MQ Series Server if it exists

3 Click Start to start the conversion. Click Close when it finishes.

Adding a Section or Entry

Follow these steps to add a section or entry

1 Right click the node to which you want to add a section or entry.

2 Select the Add Section or Add Entry option.

- If you are adding a section, enter the section name
- If you are adding an entry, enter the entry name and text

3 Click Ok when you are finished.

Locating Text

You can use Find and Find Next to locate text.

1 Click to select the beginning searching node.

2 Press CTRL+F, then enter the text and click Find to start searching. The system locates the matching text.

You can continue searching the same text by pressing F3.

Importing Configuration Information

You can import configuration information into your main configuration file. IDS lets you import configuration information into either the older IDS version 1.x INI configuration files or the newer IDS version 2.x XML configuration files.

Most control groups (INI) or sections (XML) of the imported configurations are added to the end of the main configuration file. This means that if there are control groups or sections with the same name in both the main configuration and in the imported configurations, the entries in the control group or section in the main configuration takes precedence. These control groups and sections, however, are exceptions to this rule:

- ReqType:INI
- ReqType:THREADINI
- ReqType:SAR

If these control groups or sections exist in the imported configurations, the entries in the imported control groups and sections are merged into the corresponding control groups or sections in the main configuration file — so in this case the entries in the imported file take precedence.

Here is an example section from an XML-based configuration file:

```
<section name="configuration-imports">
    <entry name="import">documanage_requests.ini</entry>
    <entry name="import">ipps_requests.ini</entry>
</section>
```

NOTE: In version 2.1 and later, IDS detect changes to imported configurations via the INIFiles or configuration-imports sections in the docserv.xml file and reload them into memory when there is a change.

Configuring MQSeries Buffer Sizes

You can increase the default buffer size of MQSeries messages and make the buffer size a setting you can maintain with a configuration entry.

- In server configuration files, the entry is put in the "BusinessLogicProcessor" section, "messaging" subsection, "queue" subsection.
- In client configuration files, the entry is put in the "DocumentClient" section, "messaging" subsection, "queue" subsection.

The entry is:

```
<entry name="mq.inputqueue.starting.buffer.size">131072</entry>
```

This setting indicates the initial size of the buffer allocated to hold an incoming message. If the message is larger than this size, a buffer is allocated that is large enough to hold the message and the application tries again. The default is 131072 (128K).

If you know that most of your messages will be smaller than 128k, you can decrease the buffer size for lower overhead for memory allocation. If, however, the majority of your messages will be larger, increase the buffer size to avoid situations where it takes multiple getMessage calls to get a message.

Testing File Transmission

Use the DSITEST utility to test the transfer of files to and from IDS. Here is an example of the syntax and parameters for this utility:

NOTE: By default, the DSITEST utility runs in Java mode. You can, however, run it in C mode. To switch modes, open DSITEST in a text editor and follow the instructions at the beginning of the script.

Syntax

```
dsitestw /time /waitonlast / display /nowait /reqtype /msg /
notrans /noattachs /norcvs /atcfile /rcvfile
```

Parameter	Description
Time	Displays total seconds for all operations. Do not include NoRCVs, ATCFile, or RCVFile with this parameter because those parameters contain user prompts that affect the time.
WaitOnLast	Waits on the last message before capturing the ending time.
Display	Displays the resulting DSI Soap XML message that contains the name/value pairs for each transaction.
NoWait	Do not wait for the server before adding next message to queue.
ReqType	The IDS request type. The default is SSS.
MSG	The name of the file that contains the request name/value pairs.
NoTrans	The total number of transactions to process.
NoAttchs	The total number of file attachments to send per transaction using the DSISendFile API. If you include this parameter, the program expects an input file named SENDFILES.MSG that contains the information for each attachment to send.
NoRCVs	The total number of file attachments to receive per transaction via the DSIReceiveFile API. If you include this parameter, the program expects an input file named RECEIVEFILES.MSG that contains the information for each attachment to receive.
ATCFile	A single file attachment to send via the DSISendFile API. The program prompts the user for the attachment ID, file name, and encoding type.
RCVFile	A single file attachment to receive via the DSIReceiveFile API. The program prompts the user for the attachment ID and file name.

Neither the case nor the order of the parameters is important.

You can include these parameters on the command line or place them in an input file named PARAMS.MSG. On the command line, separate parameters with slashes (/), dashes (-), or spaces:

```
DSITESTW /time=yes
DSITESTW -time=yes
```

```
DSITESTW time=yes
```

If you include the parameters in the PARAMS.MSG file, format them as shown in this example of the PARAMS.MSG file:

```
time=yes
waitonlast=no
display=yes
nowait=no
reqtype=LGN
notrans=50
msg=prt.msg
noattchs=0
norcvs=0
atcfile=yes
rcvfile=yes
```

Here is an example of how you could execute this program from the command line:

```
dsitesw time=yes display=yes notrans=2 reqtype=prt msg=c:\prt.msg
```

Here is an example of the PRT.MSG file:

```
USERID=FORMAKER
Arckey=00345A0D5600000008
reqtype=PRT
config=UTILITY
company=1199999
lob=Lee
policynum=Roswell, Ga 30015
rundate=021705
printpath=\10.8.10.137\Websrvr_client\html
```

If the NoAttchs parameter is greater than zero, the program expects an input file named SENDFILES.MSG which contains a list of the attachments to send. Use either NoAttachs or ATCFile, but not both.

Use the ATCFile parameter when you only want to send one file attachment. The ATCFile parameter uses command line parameters for the attachment ID, file name, and encoding type. Here is an example of the ATTACHMENTS.MSG file:

```
name=UTILITYINI
file=X:\IDS\AddlSrvrs\utility.ini
type=TEXT
name=TESTPDF
file=X:\websrvr_client\html\test.pdf
type=BINARY
```

If the NoCRVs parameter is greater than zero, the program expects an input file named RECEIVEFILES.MSG, which contains a list of attachments to receive. Include either NoRCVs or RCVFile, but not both.

Use the RCVFile parameter when you only want to receive one attachment. The RCVFile parameter uses command line parameters for the attachment ID and file name. Here is an example of the RECEIVEFILES.MSG file:

```
name=PDFFILE1
file=X:\\IDS\\AddlSrvrs\\Output\\file1.pdf
name=PDFFILE2
file=X:\\IDS\\AddlSrvrs\\Output\\file2.pdf
```

If you omit the request type from the command line or the PARAMS.MSG file, the program uses SSS as the default request type.

Referencing Attachment Variables

IDS lets you reference the attachment variable from an INI file. You can use this technique with the DAP.INI, CONFIG.INI and DOCSERV.XML files.

NOTE: This capability was previously added for the ATCSendFile and ATCReceiveFile rules. With version 2.***x***, this capability should work for all requests and rules in DOCSERV.XML, as well as the other sections imported from a DOCSERV.INI file.

Here is an example of how you reference an attachment variable via an INI option:

```
< Group >
    Option = ~GetAttach VARNAME,QUEUE
```

To reference a message variable in a configuration XML file use the following syntax:

```
<section name="Group">
    <entry name="Option">~GetAttach VARNAME,QUEUE</entry>
</section>
```

The VARNAME is the name of the variable. QUEUE specifies which queue to search for this value. For example, assume the attachment variable PRINTERTYPE specifies the printer type to use for output. IDS rules use this configuration XML option to determine the printer type (<Print>, PrtType =). In this case, you can modify the XML file as shown here:

```
<section name="Print">
    <entry name="PrtType">~GetAttach PRINTERTYPE,INPUT</entry>
</section>
```

So when the rule gets a configuration option, the value will equal the value of the input queue variable PRINTERTYPE. When the rule gets a configuration XML option, the value equals the value of attachment variable PRINTERTYPE.

NOTE: If ~GetAttach does not find the attachment variable it returns an empty string.

You can also use this to dynamically specify the file extension for the file created by ATCReceiveFile rule when you want to import that file into Documanage. You can do this as shown here in the DOCSERV.XML file:

```
<entry name="function">atcw32->ATCReceiveFile,IMPORTFILE,V2IMP,*.
~GetAttach FILETYPE,INPUT,KEEP</entry>
```

The ATCReceiveFile rule finds the attachment variable FILETYPE and uses its value as the file extension of the generated file name. Note that there are no spaces between the asterisk and period (*.) and the tilde (~) prefacing *GetAttach*. If you include a space there, it will also be in the file extension.

NOTE: The IDS attachment variable contains the printer value for each recipient. Here is an example:

```
AGENT_OUTPUT=PRINTER1
```

The client code should be able to find URLPRINTER1 to determine the output file name.

USING UNICODE IN ATTACHMENT VARIABLES

IDS supports Unicode, via UTF-8 encoding, in the setting and retrieving of values from attachment variables. The support is implemented via functions and defined constants in the DSILIB library. These functions are:

```
DSIAddAttachVarEx
DSIAddToAttachRecEx
DSILocateAttachVarEx
DSIAttachVarLengthEx
DSIAttachCursorFirstEx
DSIAttachCursorNextEx
DSIAttachCursorPrevEx
DSIAttachCursorLastEx
DSIAttachCursorValueEx
DSIAttachCursorValueLengthEx
DSIEncryptValueEx
```

These functions are similar to the *base* versions of the functions, but have an extra encoding parameter that you can set to either DSIENCODING_SINGLE_BYTE or DSIENCODING_UTF_8.

For example, when adding an attachment variable you can use...

```
DSIAddAttachVar(hdsi, DSI_OUTPUTQUEUE, "FIELD", szValue);
```

or

```
DSIAddAttachVarEx(hdsi, DSI_OUTPUTQUEUE, "FIELD", szValue,
DSIENCODING_SINGLE_BYTE);
```

or

```
DSIAddAttachVarEx(hdsi, DSI_OUTPUTQUEUE, "FIELD", szValue,
DSIENCODING_UTF_8);
```

When using the base versions of these functions, the default encoding is DSIENCODING_SINGLE_BYTE, so the first two function calls would do the same thing.

DSIENCODING_SINGLE_BYTE uses Codepage 1252 encoding, which has a one-to-one mapping between bytes and Unicode characters between 32 and 255, *except* from 128 to 159, which maps some Unicode characters down into this range. For example, the Unicode character for the Euro symbol (hex 20ac) is converted to a 128 (hex 80) and vice versa. This makes IDS compatible with how Documaker handles the Euro symbol.

DSIENCODING_UTF_8 uses UTF-8 encoding, which is a way to translate Unicode multi-byte characters into a format compatible with null-terminated C language strings while retaining all the character information.

Using the Message Queues

Docupresentment lets you use a queueing system that is based on an in-memory operation. This eliminates the need to use MQSeries in low volume production systems.

Choosing the Right Queuing Options

Choosing the right queueing options for your company can be a difficult process. There are several scenarios, as this table shows:

Installation	Function
Single PC with a single server	Used for demos and development. Not recommended for production use, even with low volumes.
Single PC with multiple servers	Here, multiple Docupresentment servers are set up on a single PC. This is recommended for low volume production systems. IDS cannot be a single point of failure because if one instance stops responding, the other instances pick up the work.
Multiple PCs with multiple servers	Here, multiple Docupresentment servers are installed on multiple PCs. This is recommended for high volume production systems.

To evaluate your system, answer these questions:

1. Is it a production system?
   - If yes, go to step 5.
   - If no, go to step 2.
2. Do you intend to use this system for performance testing?
   - If yes, go to step 5.
   - If no, go to step 3.
3. How many users will your system be servicing?
   - If more than one, go to step 5.
   - If only one, go to step 4.
4. Your system is demo or development system and you can use a single PC with a single server. You can use HTTP. You can also use JMS or MQSeries, as they are a step up from HTTP queues.
5. Your system requires multiple servers. Go to step 6.
6. How many simultaneous users does your system need to handle? The common way to determine this is calculate 5% of the maximum number of users.
   - If your system must handle more than five simultaneous users, go to step 8.
   - If it only needs to handle less than five simultaneous users, go to step 7.
7. Your system is a low volume production system and can be set up as a single PC with multiple servers. For this scenario use JMS or MQSeries.

8 Your system is a high volume production system and should be set up as multiple PCs with multiple servers. Use JMS or MQSeries.

This table summarizes the recommended queuing options:

PCs	Servers	Use this queue
Single	Single	HTTP. You can also use JMS or MQSeries, as they are a step up from HTTP queues.
Single	Multiple	JMS or MQSeries. HTTP can be used.
Multiple	Multiple	JMS or MQSeries. The use of HTTP queues is not recommended.

NOTE: The default message queue handler for IDS 2.***x*** is differs from that used for prior versions. In prior versions, IDS used xBase queues which placed messages in a physical file on disk. In version 2.0 or later, IDS uses a messaging system based on HTTP.

You do not have to do anything for the message queue system to work. All queue setup options have default values. Furthermore, any attempt to communicate with IDS will cause IDS to start if it has not already been started. It is, however, possible default port values may conflict with an application already in use, so you may have to make modifications in some cases.

Understanding the Router Process

You can use the HTTP router application to enable the client to communicate with multiple IDS servers. This application controls IDS processes and starts them as needed. The client can start this process if it fails to connect to IDS. The router is a Java application that can be started manually with the file idsrouter.bat (idsrouter.sh on UNIX platforms).

Middleware queuing systems usually have the option to save messages that are not successfully delivered to their destination. The IDS router application also has the option to do this. Any messages it receives can be stored in a JDBC compliant database and they are automatically removed when they are sent to an instance of IDS. If the IDS router is stopped, any undelivered messages that are stored will be sent to IDS when the IDS router is started up again. The HTTP connections to client programs would be severed at this point, but the messages would still be processed by IDS (for example, to ensure archiving operations are done).

The default database settings for the IDS router uses a file-based database, so no database administration or startup of external processes are required. If you are going to use a different database, you will need to obtain the JDBC drivers for the database from your database administrator (usually in the form of JAR or ZIP files) and add these files in the *lib* subdirectory under your IDS installation directory.

Under Windows platforms, the router process will have its own DOS window. If you press CTRL+C inside this DOS window, the router terminates and shuts down any instances of IDS it has started.

Under Unix platforms, the router process will write its process ID to a file named *router-pid*. Sending a *SIGINT* (signal 2) to this process ID will cause the router to terminate and shut down any instances of IDS it has started. This is usually done with the 'kill' command in a separate terminal.

How HTTP Queues are Handled

Here is an overview of how the queues work by default:

1. When client tries to send a message it attempts to get a TCP/IP connection. If the connection fails, the client tries to start the router.
2. The router starts IDS instances if necessary. Two IDS instances are started by default.
3. The client program sends the message to the router process.
4. The router process gets the message and sends it to one of the instances of IDS.
5. IDS returns the response back to router.
6. The router process returns the message back to the client that initiated the transaction.

Here is an overview of how the system typically gets an IP address and port number. Keep in mind the HTTP queue system, by default, needs ports from 49152 to 49154 and all default IP addresses are localhost.

1. The client tries to talk to the router process on port 49152.
2. When the client starts the router process, it passes port 49152 to it.
3. The router process listens on port 49152.
4. When router starts the first instance of IDS, it passes port 49153 to it.
5. The primary IDS listens on port 49153 as it was passed as a parameter from the router.
6. The primary IDS starts the next IDS and passes port 49154 (incrementing its own port address by one).
7. The secondary IDS listens on port 49154.

Using the Router Section

In the DOCSERV.XML file, the router program uses the same settings in DOCSERV.XML as IDS, as much as possible, to simplify setup. The settings used by both IDS and the router include the http port to listen to, the arguments for starting instances of IDS, and the number of instances to start.

There is an additional Router section in the configuration file for enabling and setting up the JDBC database that stores undelivered messages. This example shows the defaults:

```
<section name="Router">
    <section name="database">
        <entry name="enabled">no</entry>
```

```
        <entry name="class">org.apache.derby.jdbc.EmbeddedDriver</
entry>
        <entry name="URL">jdbc:derby:global/router-db;create=true</
entry>
        <entry name="table">DOCUCORPROUTER</entry>
        <entry name="userid"></entry>
        <entry name="password"></entry>
    </section>
</section>
```

Option	Definition
enabled	Determines whether the undelivered messages will be stored in a database for later delivery.
class	The JDBC Database driver class for the database being used. If the default database is not used you will need to contact your database administrator to obtain driver files.
URL	The JDBC locator for the database being used. If the default database is not used you will need to contact your database administrator for the proper locator string for the database.
table	The name of the table where the undelivered messages will be stored.
userid	The user ID used to gain access to the database.
password	The password for the user ID used to gain access to the database.

Using Multiple Queuing Systems

Prior to version 2.1, IDS would process requests that came in from HTTP and from one queuing system, such as MQSeries or JMS. With version 2.1 or later, IDS lets you use multiple queuing systems for processing messages, for example MQSeries and JMS queues can both be used in one instance of IDS.

To enable this, you must add queue sections to the "BusinessLogicProcessor" and "messaging" sections in the docserv.xml configuration file. Each queue section is processed and used to open a new set of input and output queues for requests and results.

Here is an excerpt from a configuration with multiple queuing systems enabled:

```
<section name="BusinessLogicProcessor">
    <section name="messaging">
        <section name="queue">
            <section name="marshallers">
                <entry
name="marshaller.class">com.docucorp.messaging.data.marshaller.SOAP
MIMEDSIMessageMarshaller</entry>
            </section>
            <entry
name="queuefactory.class">com.docucorp.messaging.mqseries.DSIMQMess
ageQueueFactory</entry>
            <entry name="ReceiveRequestIntervalMillis">1000</entry>
            <entry name="mq.queue.manager">queue1.manager</entry>
            <entry name="mq.inputqueue.name">req</entry>
            <entry name="mq.inputqueue.maxwaitseconds">5</entry>
```

```
            <entry name="mq.outputqueue.name">res</entry>
            <entry name="mq.outputqueue.expiry">600</entry>
            <entry name="mq.tcpip.host">10.1.10.123</entry>
            <entry name="mq.queue.channel">SCC_queue1.atl3nt03</
entry>
            <entry name="mq.tcpip.port">1415</entry>
            <entry name="mqseries.tracing">0</entry>
        </section>
        <section name="queue">
            <section name="marshallers">
                <entry
name="marshaller.class">com.docucorp.messaging.data.marshaller.Seri
alizationDSIMessageMarshaller</entry>
            </section>
            <entry name="mq.queue.manager">queue2.manager</entry>
            <entry name="mq.inputqueue.name">requestq</entry>
            <entry name="mq.inputqueue.maxwaitseconds">5</entry>
            <entry name="mq.outputqueue.name">resultq</entry>
            <entry name="mq.outputqueue.expiry">60</entry>
            <entry name="mq.tcpip.host">10.1.10.234</entry>
            <entry name="mq.queue.channel">SYSTEM.DEF.SVRCONN</
entry>
            <entry name="mq.tcpip.port">1414</entry>
            <entry name="mqseries.tracing">0</entry>
</section>
```

Using the Java Message Service (JMS)

The Java Message Service API is a standard programming interface for sending and receiving messages across applications. JMS itself is not a product — it is a standard that implementers (businesses or open-source groups) develop their products for. Since it is a standard, any JMS implementation can be used by IDS merely by changing configuration information. No coding changes are required. You can find general information about JMS at:

http://java.sun.com/products/jms/

JMS is part of the J2EE standard. This means that any J2EE-compliant application server, such as WebSphere or WebLogic, has an implementation of JMS as part of the application server. Other companies provide stand-alone implementations. You can find a partial list of vendors at:

http://java.sun.com/products/jms/licensees.html

Use an enterprise queuing system, such as a JMS implementation or WebSphere MQ, if your implementation has a high volume of use.

Setting up JMS

The JMS resources needed for IDS to communicate with client programs are called JMS *administered objects*. The necessary administered objects are a queue ConnectionFactory and two queues, one for requests (messages from clients to IDS) and one for results (messages from IDS to clients.

Different vendors implement the JMS in different ways so you will have to refer to the specific vendor's documentation for setting up the queues and factory.

Since different vendors implement the JMS in different ways, there has to be a standard way to access the location of the queues and factory. This is done by another Java standard, the Java Naming and Directory Interface (JNDI). JNDI support is built into IDS and clients but there are some names of resources that the JMS system administrator will have to provide to you, such as...

- Initial Context Factory – This is a name of vendor-specific Java programming code used to find the JMS administered objects.
- Provider URL – The location of the JMS administered objects.
- Security Principal (Optional) – The user ID, if required, needed to access the JMS administered objects.
- Security Credentials (Optional) – The password, if required, needed to access the JMS administered objects.
- Queue ConnectionFactory Name – The name of the queue ConnectionFactory created during setup. The JMS implementation may have only one Queue ConnectionFactory that it set up itself.
- Request and Result Queue Name – The names of the queues created for IDS communication.

For IDS, you must add these values to the docserv.xml file, in the BusinessLogicProcessor section, messaging subsection, queue subsection. Here is an example:

```
<section name="BusinessLogicProcessor">
  . . .
  <section name="messaging">
    <section name="queue">
      <entry
name="queuefactory.class">com.docucorp.messaging.jms.DSIJMSJNDIMess
ageQueueFactory</entry>
      <entry
name="jms.initial.context.factory">com.sun.jndi.fscontext.RefFSCont
extFactory</entry>
      <entry name="jms.provider.URL">file:///C:/docserv/jndi/jms/
IDS2</entry>
      <entry name="jms.security.principal">userid</entry>
      <entry name="jms.security.credentials">password</entry>
      <entry name="jms.qcf.name">IDS2QCF</entry>
     <entry name="jms.inputqueue.connectstring">jmsrequestq</entry>
     <entry name="jms.outputqueue.connectstring">jmsresultq</entry>
```

The entry *queuefactory.class* tells IDS you'll be using a JMS queuing system. The next four entries are for the standard JNDI entries. *jms.qcf.name* is for the name of the JMS Queue ConnectionFactory. *jms.inputqueue.connectstring* and *jms.outputqueue.connectstring* are for the names of the queues for communication. Since this is the server, it receives requests as input and sends results as output.

Client programs use the file docclient.xml to store configuration information. Queue information would go in the DocumentClient section, messaging subsection, queue subsection. A client program that would use the same queues as this server would have this setup:

```
<section name="DocumentClient">
  <section name="messaging">
    <section name="queue">
      <entry name="queuefactory.class">com.docucorp.messaging.jms.
        DSIJMSJNDIMessageQueueFactory</entry>
       <!-- Settings for JNDI JMS connection -->
       <entry name="jms.initial.context.factory">com.sun.jndi.
         fscontext.RefFSContextFactory</entry>
       <entry name="jms.provider.URL">file:///C:/docserv/jndi/jms/
         IDS2</entry>
       <entry name="jms.security.principal">userid</entry>
       <entry name="jms.security.credentials">password</entry>
       <entry name="jms.qcf.name">IDS2QCF</entry>
      <entry name="jms.inputqueue.connectstring">jmsresultq</entry>
        <entry name="jms.outputqueue.connectstring">jmsrequestq</
entry>
    </section>
    <!-- queue section -->
  </section>
  <!-- messaging section -->
</section>
<!-- DocumentClient -->
```

For example, consider SwiftMQ, a JMS system that uses JNDI to let users find queues. To add this capability go to the DOCSERV.XML configuration file and find the BusinessLogicProcessor section. In the Messaging subsection, under the Queue subsection, you would have these entries:

```
<section name="queue">
<entry
name="queuefactory.class">com.docucorp.messaging.jms.DSIJMSJNDIMess
ageQueueFactory</entry>
<!-- SwiftMQ -->
<entry
name="jms.initial.context.factory">com.swiftmq.jndi.InitialContextF
actoryImpl</entry>
<entry name="jms.provider.URL">smqp://docserv:docserv@server:4001</
entry>
<entry name="jms.security.principal">userid</entry>
<entry name="jms.security.credentials">password</entry>
<entry name="jms.qcf.name">plainsocket@docservrouter</entry>
<entry name="jms.inputqueue.connectstring">requestq@docservrouter</
entry>
<entry name="jms.outputqueue.connectstring">resultq@docservrouter</
entry>
</section> <!-- queue section -->
```

Entry	Description
queuefactory.class	Specifies the IDS class that sets up JMS using JNDI.
jms.initial.context.factory	Indicates the name of the Java class that interfaces a particular vendor's JMS implementation.
jms.provider.URL	Specifies how to connect to the vendor's JMS implementation.
jms.security.principal	(Optional) Indicates the JMS term for a user ID.
jms.security.principal	(Optional) Indicates the JMS term for a password.
jms.qcf.name	Indicates the name of the vendor's *queue connection factory*, a JMS term for how to find JMS queues.
jms.inputqueue.connectstring	Indicates the name of the JMS input queue that will hold requests.
jms.outputqueue.connectstring	Indicates the name of the JMS output queue that will hold results.

USING WEBSPHERE MQ

WebSphere MQ, formerly known as MQSeries, is an IBM product you can use to send and receive messages across applications. You can find general information about WebSphere MQ at:

http://www.ibm.com/software/integration/wmq/

Use an enterprise queuing system, such as a JMS implementation or WebSphere MQ, if your implementation has a high volume of use.

For Windows and UNIX platforms, you can use the RUNMQSC tool to create queues and queue managers. You can use the WebSphere MQ Explorer GUI tool on Windows to create queues and queue managers on the local Windows and remote UNIX, Linux, or Windows WebSphere MQ servers when configured for remote administration. When WebSphere MQ installs on Windows, a queue manager is created if you select the option to install default configuration.

Java support for WebSphere MQ is included in version 5.3 and later. The Java libraries for WebSphere MQ can be used in either client mode (via TCP/IP) or in bindings mode (where the Java application runs on the same machine as the MQSeries server), with the exception of OS/390 which must be run in bindings mode.

NOTE: All queues are under a queue manager and there will be a queue manager for each machine that uses WebSphere MQ Server. The queue managers communicate with each another and pass the messages to the appropriate queue underneath.

When adding machines to the cluster, you can use a wizard through the WebSphere MQ Explorer to set up the cluster. Even though the WebSphere MQ installation program creates a default cluster that uses the network domain name, it does not create the cluster sender channels needed to communicate between the repository machines.

SETTING UP WEBSPHERE MQ

The WebSphere MQ resources needed for IDS to communicate with client programs are two WebSphere MQ queues, one for requests (messages from clients to IDS) and one for results (messages from IDS to clients).

For IDS, add these values to the docserv.xml file, in the BusinessLogicProcessor section, messaging subsection, queue subsection. Here is an example:

```
<section name="BusinessLogicProcessor">
  . . .
  <section name="messaging">
    <section name="queue">
      <entry name="queuefactory.class">
com.docucorp.messaging.mqseries.DSIMQMessageQueueFactory</entry>
      <entry name="mq.queue.manager">queue.manager</entry>
      <entry name="mq.inputqueue.name">requestq</entry>
      <entry name="mq.inputqueue.maxwaitseconds">5</entry>
      <entry name="mq.outputqueue.name">resultq</entry>
      <entry name="mq.tcpip.host">10.1.10.1</entry>
      <entry name="mq.queue.channel">SCC_channel</entry>
      <entry name="mq.tcpip.port">1414</entry>
```

The entry *queuefactory.class* tells IDS you'll be using a WebSphere MQ queuing system. The following are names of WebSphere MQ objects required for communication:

- mq.queue.manager
- mq.inputqueue.name

Client programs use the docclient.xml file to store configuration information. Queue information would go in the DocumentClient section, messaging subsection, queue subsection. A client program that would use the same queues as this server would have this setup:

```
<section name="DocumentClient">
    <section name="messaging">
        <section name="queue">
            <entry
name="queuefactory.class">com.docucorp.messaging.mqseries.DSIMQMess
ageQueueFactory</entry>
             <entry name="mq.queue.manager">queue.manager</entry>
             <entry name="mq.inputqueue.name">resultq</entry>
             <entry name="mq.inputqueue.maxwaitseconds">5</entry>
             <entry name="mq.outputqueue.name">requestq</entry>
             <entry name="mq.tcpip.host">10.1.10.1</entry>
             <entry name="mq.queue.channel">SCC_channel</entry>
             <entry name="mq.tcpip.port">1414</entry>
        </section>
        <!-- queue section -->
    </section>
    <!-- messaging section -->
</section>
<!-- DocumentClient -->
```

Using MSMQ

Use the DSIMSMQMessageQueue and DSIMSMQMessageQueueFactory classes to communicate via asynchronous messages through MSMQ on Windows platforms. Support is provided for path names and direct format names. These platforms are supported:

- Windows 2000
- Windows XP
- Windows 2003 and later Windows operating systems

To enable MSMQ messaging, change the queue factory class in the configuration file to:

```
com.docucorp.messaging.msmq.DSIMSMQMessageQueueFactory
```

Please refer to the HTML documentation shipped with the Java SDK for a description of the properties supported for the MSMQ message bus. In particular, see a description of the setProperties method in the com/docucorp/messaging/msmq/DSIMSMQMessageQueueFactory class.

Property settings

Here is an example of the docserv.xml file:

```
<section name="queue">
    <entry name="queuefactory.class">com.docucorp.messaging.msmq.
        DSIMSMQMessageQueueFactory</entry>
    <entry name="ReceiveRequestIntervalMillis">1000</entry>
    <!-- Settings for MSMQ connection -->
    <entry name="msmq.server.name">jr</entry>
    <entry name="msmq.inputqueue.name">DIRECT=OS:jr\private$
        \requestq</entry>
    <entry name="msmq.outputqueue.name">DIRECT=OS:jr\private$
        \resultq</entry>
    <entry name="msmq.timeout">10000</entry>
</section>
```

Here is an example of the docclient.xml file:

```
<section name="queue">
    <entry name="queuefactory.class">com.docucorp.messaging.msmq.
        DSIMSMQMessageQueueFactory</entry>
    <!-- Settings for MSMQ connection -->
    <entry name="msmq.server.name">jr</entry>
    <entry name="msmq.inputqueue.name">DIRECT=OS:jr\private$
        \resultq</entry>
    <entry name="msmq.outputqueue.name">DIRECT=OS:jr\private$
        \requestq</entry>
    <entry name="msmq.timeout">10000</entry>
</section>
```

Here is an example of the dsimessage.properties file:

```
queuefactory.class=com.docucorp.messaging.msmq.DSIMSMQMessageQueueF
actory
    # MSMQ for windows
    msmq.server.name=jr
    msmq.inputqueue.name=private$\resultq
    msmq.outputqueue.name=private$\requestq
    msmq.timeout=10000
```

These components are required:

- DocucorpMsg.jar
- msmqlib.dll

Be sure to include the MSMQLIB.DLL in the system path so it is found at run time.

NOTE: Testing has shown that when using direct format names to private queues, the queue used to read messages should be a local queue. This improves performance significantly (about 10 times) and avoids the generation of unnecessary TCP socket connections that is incurred when reading messages from a remote private queue.

If the IDS client and IDS server components reside on separate boxes, configure the client to read messages from a local private result queue. Configure the server to read messages from a local private request queue.

Using correlation IDs

IDS supports WebSphere MQ/JMS client applications that specify a correlation ID or a message ID in a request.

Client applications can specify a correlation ID in a request and retrieve a response with the same correlation ID. Client applications can also use the conventional method of correlating a response with a request by specifying a message ID in a request and retrieving a response with a correlation ID matching the message ID of the request.

Using the ReceiveByCorrelationID API

IDS supports MSMQ 3.0 client applications that want to retrieve messages via the ReceiveByCorrelationID API using the message ID property of the request message as the parameter.

On the server side, IDS sets the Correlation ID property of a response message equal to the value of the Message ID property of the request message when the UNIQUE_ID value of the request message is blank. For this reason, client side applications that want to retrieve a message by correlation ID, should make sure the UNIQUE_ID value of the request message is blank.

Generating the message ID

The message buses for IDS can generate the message ID for client applications during a request. A message bus generates the message ID when the putMessage method of the output queue provides a value of null or a blank value for the messageID argument.

The message ID generated by the message bus can then be retrieved through the getMessageID method of the output queue and used as the ID argument in the getMessage method of the input queue to retrieve a matching reply.

If a messageID value is provided to the putMessage method, then that value is used instead, with one exception — the MSMQ message bus only supports message bus generated message IDs.

Here is a Java example that lets the message bus generate the messageID for a client request:

```
outputQueue.putMessage(null, reqObj);
String unique = outputQueue.getMessageID();
Object resObj = inputQueue.getMessage(unique, timeOutMillis, 3);
```

Here is a Java example that lets a client application define the message ID for a request:

```
UniqueStringGenerator usg = new UniqueStringGenerator();
String messageID = usg.generateUniqueString(32);
outputQueue.putMessage(messageID, reqObj);
Object resObj = inputQueue.getMessage(messageID, timeOutMillis, 3);
```

These message buses support this functionality:

- Java: HTTP, JMS, MSMQ, MQSeries, and Mail

  The IDSJSP package (Java) also supports this feature when used with the DSIJavaMsg and DocucorpMsg packages. Use the setUniqueId method of a dsi or dsimsg bean to set the unique ID to null or a blank value to indicate you want the message bus to generate the message ID during a client request.

- Csharp: HTTP, MSMQ, and MQSeries

  The DSIInterface class in the DocucorpDSI assembly (Csharp) also supports this feature when used with the DocucorpMsg assembly. Use the setUniqueId method of the DSIInterface class to set the unique ID to null or a blank value to indicate you want the message bus to generate the message ID during a client request.

Using MSMQ direct format queue names

You can set up the MSMQ_FileConvert control group to recognize path names, format names, and direct format names.

NOTE: Before version 2.0, only path names were supported in the INI file and were converted into format names by the MSMQ API MQPathNameToFormatName().

The system recognizes when you specify a format name in the INI file and skips the API call to MQPathNameToFormatName. For example, private queues cannot be accessed unless the direct name is used. So, if you are using a private queue but the IDS client and server are on different PCs, you must use direct format names.

Here is an example of the INI options you would set:

```
< DBTable:RequestQ >
    DBHandler = MSMQ
< DBTable:ResultQ >
    DBHandler = MSMQ
< MSMQ_FileConvert >
    ;requestq = FSINTSRV08\JRREQQ
    ;resultq = FSINTSRV08\JRRESQ
    requestq = DIRECT=TCP:10.8.10.137\PRIVATE$\JRREQQ
    resultq = DIRECT=TCP:10.8.10.137\PRIVATE$\JRRESQ
    ;requestq = DIRECT=OS:JDOE\PRIVATE$\JRREQQ
    ;resultq = DIRECT=OS:JDOE\PRIVATE$\JRRESQ
    ;requestq = PUBLIC=dc7b9469-dbae-11d6-ae6c-00104bd359c1
    ;resultq = PUBLIC=dc7b946c-dbae-11d6-ae6c-00104bd359c1
    ;requestq = .\private$\JRREQQ
    ;resultq = .\private$\JRRESQ
    ;requestq = PRIVATE=cdb19274-6146-4ab9-8679-
6e998943a938\00000016
    ;resultq = PRIVATE=cdb19274-6146-4ab9-8679-6e998943a938\00000017
```

```
< RequestQ >
    Name = requestq
< RESULTQ >
    Name = resultq
```

Keep in mind these definitions:

FORMAT NAMES. A *format name* is a unique name generated by MSMQ. The MQPathNameToFormatName API normally converts a path name specified in the INI file into a format name by looking up the format name in the MSMQ MQIS. The format name is then used by the MQOpenQueue API to open a queue.

To avoid conversion of the path name into a format name, you can specify a format name in the INI file. Here are some examples:

```
PUBLIC=dc7b9469-dbae-11d6-ae6c-00104bd359c1
PRIVATE=cdb19274-6146-4ab9-8679-6e998943a938\00000016
```

DIRECT FORMAT NAMES. You can also specify a *direct format name* to avoid the path name to format name conversion and to skip the connection to the MQIS altogether. This, in essence, generates a One-HOP direct connection from one MSMQ box to another, which can be useful when you are connecting to a remote box that hosts private queues in an MSMQ workgroup configuration. Here are some examples:

```
DIRECT=TCP:10.8.10.137\PRIVATE$\JRREQQ
DIRECT=OS:JDOE\PRIVATE$\JRREQQ
```

In the first example, the protocol specified is TCP and the IP address of the box hosting the private queues is specified.

In the second example, *OS* indicates that the native protocol of the operating system where the private queues reside should be used and the NetBIOS name is specified instead of the IP address of the box hosting the queues. All connection information is contained within the direct format names to avoid a connection to the MQIS.

NOTE: Direct format names are only supported in Windows NT 4.0 Service Pack 6a or later.

QUEUE PATH NAMES. MQLIB also supports specifying normal queue path names in the INI file by calling the MQPathNameToFormatName API to convert them into the proper format name before calling the MQOpenQueue API. Here are some examples:

```
FSINTSRV08\JRREQQ
.\private$\JRREQQ
```

Queue pooling

When setting up the messaging parameters in a Java application to talk to IDS, you can specify that the message queues should be kept in a *pool* of queues and reused throughout the life of the application. If a pooled connection is not used after 30 seconds, it may be removed from the pool and the queue is closed automatically. You can also close pooled connections manually.

Message pooling has these advantages:

- Reusing an existing, previously opened queue can be faster than opening a new queue connection for every communication with IDS.
- In an application there can be more threads of execution than there are available queue connections on the queuing server. Threads are automatically blocked until a connection to a queue is available. This keeps you from having to limit the number of threads to the number of available connections or creating some kind of multi-threaded blocking.

This functionality is most useful in application servers, such as WebSphere, that host long-running, multi-threaded Java clients such as JSPs or servlets. No extra configuration is necessary on the web application, such as WebSphere.

To use queue connection pooling, you must make the following changes in your the client configuration file. This can be a properties file such as dsimsgclient.properties, if you are upgrading from IDS 1.x or a XML-based configuration file such as docclient.xml.

NOTE: IDS version 2.***x*** detects which kind of configuration you have and loads it automatically.

In both the DSIJavaMsg and DocucorpMsg libraries, the settings for queue connections are passed to the libraries by Java properties objects. To enable and set up pooling, use these options:

Option	Description
pooling.enabled	Set to Y to enable pooling. The default is N.
pooling.input.pool.size	Number of input queue connections to pool. The default is 10.
pooling.output.pool.size	Number of output queue connections to pool. The default is 10.

Here is an example of a client properties file that would enable an input and output pool of 20 MQSeries connections. This file would normally be named dsimsgclient.properties.

```
pooling.enabled=Y
pooling.input.pool.size=20
pooling.output.pool.size=20
queuefactory.class=com.docucorp.messaging.mqseries.DSIMQMessageQueu
eFactory
mq.queue.manager=venus.queue.manager
mq.inputqueue.name=RESULTQ
mq.outputqueue.name=REQUESTQ
mq.outputqueue.expiry=120
mq.tcpip.host=10.1.10.1
mq.queue.channel=SYSTEM.DEF.SVRCONN
```

```
mq.tcpip.port=1414
```

You must also make the following API changes if your implementation talks directly to the DocucorpMsg library, whether it's in Java or JSP, to use queue connection pooling.

If you are using the DocucorpMsg library, pooled connections can only be obtained through the static method DocucorpMsgUtil.getQueueFactory(Properties props). This method returns a DSIMessageQueueFactory that may be pooled, but is used in the same manner as the previously unpooled DSIMessageQueueFactory objects.

If you are creating a multi-threaded command line program, any outstanding pooled connection can be closed with the static method DocucorpMsgUtil.closePooledConnections(Properties props).

NOTE: No code changes are needed for users of the DSIJavaMsg library.

USING SECURITY EXITS

You can attach custom security exits to WebSphere MQ queues. Security exits are external libraries of code that can be installed and run in WebSphere MQ queues. For IDS, security exits consist of a Java class in a .jar file, with an optional native component.

To have a security exit installed and run, you need to know the name of the Java class for the security exit and the name of the .jar file that has the security exit.

In the docserv.xml configuration file, set up a queue section for WebSphere MQ queues. In that section, add an entry similar to the one shown here:

```
<entry name="mq.customsecurityexit.classname">com.customer.
    securityClassName<entry>
```

Substitute the name of the your security exit Java class name for:

```
mq.customsecurityexit.classname
```

You must load the .jar file that has the custom security exit code. For application servers running Docupresentment client code, refer to the application server's documentation for information on modifying the classpath for the web application or for including a .jar file in a particular directory.

For Docupresentment server, you can either...

- Put the .jar file in the server's lib directory, or
- Modify how Docupresentment server is run by adding the System property

  ```
  com.skywiresoftware.extraClasspath
  ```

  with a reference to any .jar files needed to run the security exit.

For example, for the docserver.bat file, you could add an entry like the one shown here:

```
-Dcom.skywiresoftware.extraClasspath=/path/to/security.exit.jar
```

Using Client Connection Definition Tables

The WebSphere message bus can read connection information from Client Connection Definition Table (CCDT) files. The code can then use any queue manager listed in the CCDT file to establish a connection.

NOTE: Support for CCDT files in Java requires WebSphere MQ, version 6.0 or later. Refer to the WebSphere MQ documentation for information about Client Connection Definition Tables.

For additional information, see the description of the mq.ccdt.url property in the HTML documentation for the com.docucorp.messaging.mqseries.DSIMQMessageQueueFactory class. This information is included with the Java SDK.

Using SSL Connections

To use SSL connections the queue manager and server connection channel must be configured to use SSL. SSL connections are only supported in MQ client mode (version 5.3 or later).

See the security guide for your version of WebSphere MQ for information on how to configure a queue manager and server connection channel for SSL communication. A Java key store must be available to IDS and it must contain a personal certificate issued by a trusted Certificate Authority (CA). This trusted CA must be present in the key repository used by the queue manager. The CA certificate must be part of the trusted certificates in the key repository used by the queue manager.

Also, the same CA certificate must be present in the Java keystore. If the personal certificate used by the queue manager was issued by another CA, then the CA certificate for it must also be present in the Java key store so the SSL handshake can be negotiated between the MQ Client (IDS) and the MQ Server (the queue manager).

If the server connection channel is configured to authenticate client certificates or to verify the Distinguished Name of client certificates via the SSLPeer property, the personal certificate in the Java key store must also contain the private key. The entry type in the Java key store for the personal certificate should be '*keyEntry*' instead of '*trustedCertEntry*' in this case.

See the HTML documentation for the DSIMQSeriesMessageQueueFactory class, which is shipped with the IDS DSI Java SDK, for additional information on the MQ SSL properties. In particular, see the description of the setProperties method.

Here is an example of these configuration properties:

```
<entry name="mq.ssl.cipherspec">RC4_MD5_US</entry>
<entry name="mq.ssl.keyrepository">c:\ibm\mq\ssl\key</entry>
<entry name="mq.ssl.peername">CN=ssl_qmgr, C=US, S=GA, L=Atlanta,
O=Docucorp International, OU=PD</entry>
<entry name="mq.ssl.socketFactory.class">
com.docucorp.messaging.mqseries.DSIMQSSLSocketFactory</entry>
<entry name="mq.ssl.protocol">SSLv3</entry>
```

```
<entry name="mq.ssl.keystore">c:/docserv/keystore/
java_certificate_store</entry>
<entry name="mq.ssl.keystore.type">JKS</entry>
<entry name="mq.ssl.keystore.manager.type">SunX509</entry>
<entry name="mq.ssl.keystore.pwd">changeit</entry>
<entry name="mq.ssl.truststore">c:/docserv/keystore/
java_certificate_store</entry>
<entry name="mq.ssl.truststore.type">JKS</entry>
<entry name="mq.ssl.truststore.manager.type">SunX509</entry>
<entry name="mq.ssl.truststore.pwd">changeit</entry>
<entry name="mq.ssl.debug">true</entry>
```

USING THE REPLYTOQUEUENAME AND REPLYTOQUEUEMANAGERNAME PROPERTIES

IDS checks for these MQ message properties during MQGET calls issued by IDS:

- replyToQueueManagerName
- replyToQueueName

If the properties are defined in the message, they are used to reply to a specific queue manager and queue — MQPUT1 calls are used instead of MQPUT calls in this case. In addition, if you are using the SOAP marshaller, reply messages will contain the REPLYTOQUEUEMANAGER and REPLYTOQUEUE elements in the control block.

If the replyToQueueName property is not defined in the message received by IDS, the system uses the value defined in the mq.outputqueue.name property in the docserv.xml configuration file.

To use this functionality, client applications submitting requests to IDS should make sure the replyToQueueManagerName and replyToQueueName message properties are set.

USING HTTP

In addition to processing requests received from enterprise queuing systems (WebSphere MQ and JMS), IDS can receive requests through HTTP. You can configure IDS to use HTTP-based messaging, queue-based messaging, or both.

HTTP messaging replaces the file-based xBase queues from earlier versions of IDS (prior to 2.0) as the low-volume messaging system. Like the xBase queues from earlier versions, there is no setup of an extra program to run the messaging system, plus HTTP messaging does not have the 64K limit of xBase queues.

Although you can use HTTP-based messaging for lower-volume installations, an enterprise-ready queuing system is recommended for higher-volume installations and in situations where requests should not be lost if IDS or IDS clients are halted.

The default format of messages sent to HTTP-based messaging is the SOAP with Attachments format. The request type and message variables are in an XML message embedded into a SOAP envelope, and attachments are added in MIME format to the message. For more information, see Using the XML Messaging System on page 152. You can find more information about SOAP and attachments at:

http://www.w3.org/TR/SOAP/

http://www.w3.org/TR/SOAP-attachments

The combination of HTTP messaging and the SOAP format means that IDS can act as a Web Services server for IDS requests. Web Services clients can send messages to IDS via HTTP using the SOAP and SOAP With Attachments format. IDS processes the request and sends the result back to the Web Services client in SOAP format. A Java sample program, JAXMClient, is provided for testing and as a source code template so you can create your own Web Services clients using Sun's Java API for XML Messaging. You can find more information about the Java API for XML Message at:

http://java.sun.com/xml/jaxm/index.html

NOTE: No particular version of the SOAP protocol is required. In IDS version 1.8 and version 2.***x*** the system does manual XML parsing to extract information, so versions are essentially ignored. No WSDL file is needed for a .NET client.

Setting up HTTP

For IDS, add these values to the docserv.xml file, in the BusinessLogicProcessor section, messaging subsection, http subsection:

```
<section name="BusinessLogicProcessor">
  ...
    <section name="messaging">
        <section name="http">
            <entry name="port">49152</entry>
            <entry name="WaitForResultMillis">30000</entry>
            <entry name="HttpProcessors">15</entry>
            <entry name="RequestPath">xslPath</entry>
            <entry name="HtmlPath">htmlPath</entry>
        </section>
```

Entry	Description
port	Indicates the http port that the first instance of IDS will be accessible from. If more than one instance is running they will use subsequent ports, starting with this one.
WaitForResultMillis	Indicates how long, in milliseconds, to wait for the request to be processed before timing out.
HttpProcessors	Indicates how many extra threads will be set up to accept http requests from clients.
RequestPath	Indicates where IDS will find customization XSL style sheets.
HtmlPath	Indicates where IDS will find extra HTML-based information.

Client programs use the file docclient.xml to store configuration information. Queue information would go in the DocumentClient section, messaging subsection, queue subsection. A client program that would use the same queues as this server would have this setup:

```
<section name="DocumentClient">
  <section name="messaging">
    <section name="queue">
      <entry name="queuefactory.class">com.docucorp.messaging.http.
        DSIHTTPMessageQueueFactory</entry>
      <entry name="http.url">http://localhost:49152</entry>
    </section>
    <!-- queue section -->
  </section>
  <!-- messaging section -->
</section>
<!-- DocumentClient -->
```

The entry *queuefactory.class* tells the client program to use HTTP to communicate. *http.url* gives the name of the server machine and TCP/IP port to use.

NOTE: The http subsection is also used by IDS when it processes ordinary requests via HTTP messaging. See Using IDS to respond to requests via a browser on page 127 for more information.

Responding to URL requests

IDS can respond to requests formatted as a URL from a browser and display results in HTML. You can customize the display for a particular request or use a default display which shows message variables, rowsets, and any errors encountered. An example URL is

```
http://localhost:49152/
request?REQTYPE=SSS&USERID=USERID&PASSWORD=PASSWORD
```

Where *localhost* is the IP address to contact, and *49152* is the port number at the IP address that IDS is using.

You can add any number of message variables to the URL but attachments are not supported.

To use this capability you must make changes in the DOCSERV.XML configuration file. First, find the BusinessLogicProcessor section, then locate the Messaging subsection. In this subsection, create an HTTP subsection, as shown here:

```
<section name="http">
<entry name="port">49152</entry>
<entry name="WaitForResultMillis">30000</entry>
<entry name="HttpProcessors">15</entry>
<entry name="RequestPath">xslPath</entry>
<entry name="HtmlPath">htmlPath</entry>
</section>
```

Entry	Description
port	Indicates the http port that the first instance of IDS will be accessible from. If more than one instance is running they will use subsequent ports, starting with this one.
WaitForResultMillis	Indicates how long, in milliseconds, to wait for the request to be processed before timing out.
HttpProcessors	Indicates how many extra threads will be set up to accept http requests from clients.
RequestPath	Indicates where IDS will find customization XSL style sheets.
HtmlPath	Indicates where IDS will find extra HTML-based information.

Using IDS to respond to requests via a browser

IDS can respond to requests directly from a web browser and display the results formatted as HTML. A web server is not needed for the display. This is useful for debugging purposes. You can customize the formatting for each request type, otherwise a default page appears showing any message variables and error messages for the request.

After a request is processed, the results are formatted in the SOAP with MIME Attachments format. See Setting up HTTP on page 125 for more information. IDS looks for an XSL style sheet named:

```
REQTYPE.xsl
```

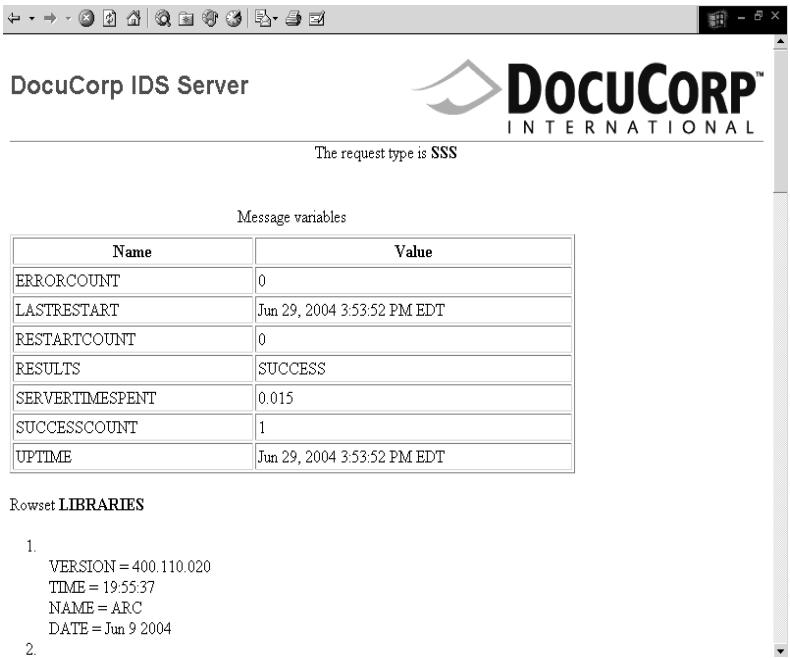
where *REQTYPE* corresponds to the request type of the request. If this file is found, it is used to transform the SOAP XML. If there is no XSL style sheet for that request, a default style sheet is used. The default style sheet displays the message variables name/value pairs in a table then lists any rowsets.

To have a request displayed in your browser, build a URL similar to:

```
http://localhost:49152/
request?REQTYPE=SSS&USERID=USERID&PASSWORD=PASSWORD
```

localhost and *49152* are the TCP/IP address and port number where IDS is running. Message variables are entered as NAME=VALUE pairs, separated by an ampersand (&). The only required variable is REQTYPE. Any number of message variables can be added to the URL but attachments are not supported.

Here is an example:



DocuCorp IDS Server

DOCUCORP INTERNATIONAL

The request type is **SSS**

Message variables

Name	Value
ERRORCOUNT	0
LASTRESTART	Jun 29, 2004 3:53:52 PM EDT
RESTARTCOUNT	0
RESULTS	SUCCESS
SERVERTIMESPENT	0.015
SUCCESSCOUNT	1
UPTIME	Jun 29, 2004 3:53:52 PM EDT

Rowset **LIBRARIES**

1. VERSION = 400.110.020
   TIME = 19:55:37
   NAME = ARC
   DATE = Jun 9 2004
2.

Configuring IDS to handle HTTP requests

In the docserv.xml configuration file, in section 'BusinessLogicProcessor', subsection 'messaging' create an http subsection:

```
<section name="http">
  <entry name="port">49152</entry>
  <entry name="WaitForResultMillis">30000</entry>
  <entry name="HttpProcessors">15</entry>
  <entry name="RequestPath">xslPath</entry>
  <entry name="HtmlPath">htmlPath</entry>
</section>
```

Parameter	Description
port	The HTTP port you access IDS from.
WaitForResultMillis	How long, in milliseconds, to wait for the request to be processed before timing out
HttpProcessors	The number of extra threads to set up to process HTTP requests from clients.
RequestPath	Where IDS can find customization XSL style sheets. If you use a relative path, keep in mind the current directory is where IDS was installed.
HtmlPath	Where IDS can find extra HTML-based information. If you use a relative path, keep in mind the current directory is where IDS was installed.

The http subsection is also used by IDS when it processes ordinary requests via HTTP messaging. See Using HTTP on page 125 for more information.

Using Multiple Bridges

You can set up multiple bridges to use a single server. These bridges include Documaker Bridge, the Documanage Bridge, the Printstream Bridge, and the Docuflex Bridge. There are two ways to do this:

- The simplest way is to combine all of the required INI options into the DAP.INI file, which is loaded by the DPRInit rule.
- The more advanced and recommended way is to use the DPRSetConfig rule and multiple INI files. This lets you set INI options specifically for each bridge (different values for the same INI option for different bridges).

NOTE: The system expects to globally apply the values it finds in an INI file. You handle this by switching the context based on the attachment variable CONFIG, using the DPRSetConfig rule.

To accomplish this you need to use the CONFIG attachment variable. This attachment variable can be passed from an HTML form (or link) from the browser to the CGI, or added to the attachment list using a custom rule.

In the DAP.INI file, you specify which INI files should be used for each of the CONFIG values, for example:

```
< Config:TIFF >
    INIFile = tiff.ini
< Config:DAPARC >
    INIFile = daparc.ini
```

The system supports multiple INI files, such as:

```
< Config:TIFF >
    INIFile = tiff.ini
    INIFile = myownini.ini
```

In the case of the usage of the DPRSetConfig rule and the appropriate INI values, for example if the DAP.INI file includes this setting:

```
< Config:TIFF >
    INIFile = tiff.ini
```

The actual INI context used in the rules and bridges, will include the INI values from the TIFF.INI and DAP.INI files. When using multiple INI files with the same INI options, but different values, the first INI value — the value from the first INI file — is returned when the rules ask for the INI value.

The only thing you have to watch out for is when the system expects multiple INI values. For example, Documaker Bridge rules look for multiple INI values for the AppIdx options in the ArcRet control group. In this case, the system gets all of the matching values from the first INI file (TIFF.INI in this example) and then gets all of the matching INI values from the DAP.INI file.

Here's another example:

Assume IDS is installed with the Documanage and Printstream bridges. You create two CONFIG values: TIFF and META. The DAP.INI file will include these options:

```
< Config:TIFF >
    INIFile = tiff2pdf.ini
< Config:META >
    INIFile = meta2pdf.ini
```

The Documaker-related INI options for each bridge should go into each of these INI files. You can optionally place common INI options in the DAP.INI file. You should use the DPRSetConfig rule in any rule list which includes request types and that intend to use DPR, TPD, or MTC rules.

The DPRSetConfig rule is located in DPRW32.DLL and runs on MSG_RUNF. Here is the example (from the DOCSERV.INI file of the rule list with this rule. This rule must run before any other rules which use Documaker code and expect Documaker-related INI options.

```
< ReqType:MTC >
    function = atcw32->ATCLogTransaction
    function = atcw32->ATCLoadAttachment
    function = dprw32->DPRSetConfig
    function = atcw32->ATCUnloadAttachment
    function = mtcw32->MTCLoadFormset
    function = dprw32->DPRRotateFormsetPages
    function = mtcw32->MTCPrintFormset
```

NOTE: The TPDInitRule rule should be in the list after DPRInit rule.

Request types and multiple bridges

When you install one bridge over another or develop a new bridge, keep in mind that some of the request types listed in the DOCSERV.INI or DOCCLNT.INI files may already be in use by another application or bridge.

If this happens, change your bridge or application to use an unused request type. The length of the request type string is not limited to three characters. You can enter up to 19 characters. Do not include any special characters, instead limit it to alphanumeric characters and underscores.

For example, suppose you are trying to use request type LGN, but it is already taken and the list of rules in the DOCSERV.INI file for this request type is not the same as you need.

In this situation, you could define your own request type, such as MYLGN and add the rules you need to DOCSERV.INI file under the ReqType:MYLGN control group. Be sure to change your application to use the MYLGN request type instead of LGN.

Submitting Batch Requests

You can use the FILE2IDS utility to read a text file which contains a series of requests and submit those requests to an IDS server. Each line of the text file equals one request. You specify the request type on the command line and the attachment variables are created from each line in the input file in this manner:

- Each line is broken into 1000 byte chunks (1000 is the default size, you can set the size using the /L parameter)
- Each chunk is added as attachment variable RECORDLINE*XX*, where *XX* is the sequence number of the chunk.

The attachment variable RECORDPARTS specifies how many chunks are added.

NOTE: The FILE2IDS is a Visual Basic program. You must have a VB runtime installed to run this program. You can click the Help button when you run FILE2IDS to see a summary of the various parameters.

To run the FILE2IDS utility, enter this command:

```
file2ids /C /D /I /L /K /R /T /Name /W
```

All parameters are passed in with the equals (=) sign, for example:

```
/I=file.txt
```

Parameter	Description
/C	Specifies the value of the CONFIG attachment variable
/D	Set to *On* to turns on the debugging window and wait for the result from IDS.
/I	The name of the file
/K	Enter *Y* or *N* to have the system wait or not wait for a key to be pressed in the debug window.
/L	Specifies the line length. The default is 1000.
/R	The request type. The default is ***Email***.
/T	The length of time in milliseconds before IDS times out. The default is 15000.
/Name	Lets you pass a name and value to the IDS rule in this format: /Name=Value. For instance, /ABC=BCD adds the attachment variable ABC with the value BCD.
/W	The length of time in milliseconds IDS will wait before retrying. The default is 1000.

If you run the utility with no parameters, it displays a window which lets you then enter the parameters shown above.

The utility returns one of the following values:

Value	Description
0	The utility completed its task and no errors occurred.
1	No input file information was found.
2	The utility was canceled before it ran. Typically, this indicates you clicked Exit on the window which asks for the parameters.
3	Other error.

Printing in Duplex Mode to PCL Printers

Windows does not let you print files that are a mixture of simplex and duplex pages from Acrobat. The whole document has to be printed the same way. IDS, however, lets you print a file to a local PCL printer which preserves the file's duplex information. There are two ways to do this:

- By inserting blank pages into a PDF file for the pages in simplex mode. This requires the system to create two PDF files, one for printing and one for viewing.
- By creating the PCL file, compressing the file, downloading the file, and then decompressing it for printing.

IDS can create compressed PCL files several ways:

- Using the IDS print rules. If you use the print rules, use the Compression attachment variable to create a compressed file. When used by Print Preview, you must also pass the Compression attachment variable to the DPRPrint rule. See the SDK Reference for more information about these rules.
- Using Documaker. If you create the file via Documaker, you set INI options to create the file. The PRTZCompressOutPutFunc function is called to compress the output files. To use this function to compress an output file such as a PCL print batch file, add these INI options:

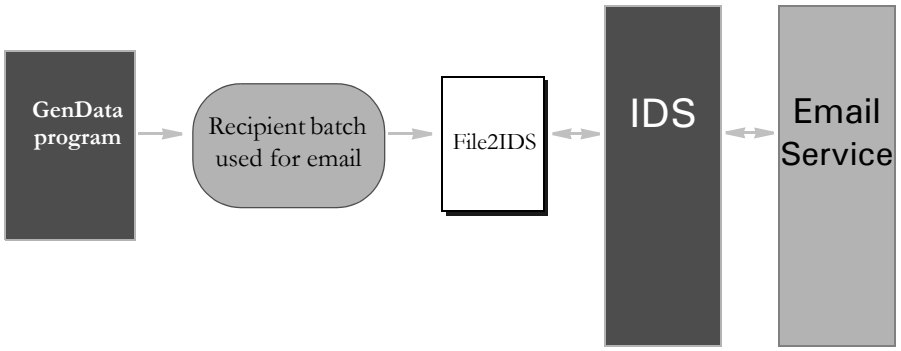
```
< PrtType:PCL >
    OutputMod = PRTW32
    OutputFunc = PRTZCompressOutputFunc
```

- Using a Java application which can decompress the file and send it to a local printer. The application is provided in the WindowsRawPrinter.jar file and it requires that you install the DSIJWP.DLL file.

NOTE: The output is compressed, regardless of the file's extension. You must decompress the file before you can print it.

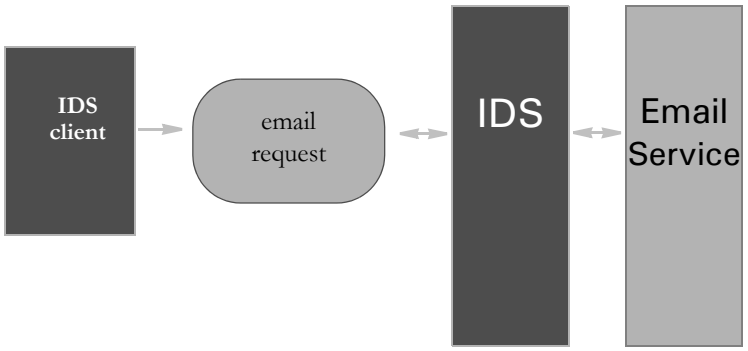
Using IDS to Distribute Email

You can use the Internet Document Server to distribute email. The following illustration shows how it works with the GenData program, which is part of Documaker:



NOTE: Because the File2IDS utility is a Visual Basic program, the above scenario is only available for Windows environments.

The following scenario, which is available for both Windows and UNIX environments, shows how it works if you are using Documaker Bridge, but not the GenData program:



For either approach, to use the Internet Document Server to distribute email, you must modify these files:

- DOCSERV.XML
- DAP.INI

Modifying the docserv.xml Configuration File

This configuration file must contain the following section. These rules are used to take messages sent to IDS and send formatted email messages to an email server such SMTP. The system supports text-based templates or HTML templates that can be sent as an attachment or used as the message body.

```
<section name="ReqType:EMAIL">
  <entry name="function">atcw32->;ATCLogTransaction</entry>
  <entry name="function">atcw32->;ATCLoadAttachment</entry>
  <entry name="function">dprw32->;DPRParseRecord</entry>
```

```
  <entry name="function">dprw32->;DPRFindTemplate</entry>
  <entry name="function">dprw32->;DPRAdd2Attachment</entry>
  <entry name="function">dprw32->;DPRCreateEMailAttachment</entry>
  <entry name="function">dprw32->;DPRMail</entry>
  <entry name="function">atcw32->;ATCUnloadAttachment</entry>
</section>
```

Modifying the DAP.INI File

This INI file must contain the following control groups and options.

EmailDFD control group

The DFD file specified by this option is used by the DPRParseRecord rule. The system expects to receive the email data from the client in fixed-length records defined by this DFD. Using a DFD to define the record layout increases flexibility. This DFD can be identical to a batch recipient DFD, where the system is taking an output from the GenData program and using it as input to the email server.

```
< EmailDFD >
    File = .\dfd\attchdfd.dfd
```

Email2IDS control group

This group is also used by the DPRParseRecord rule. The option (on the left) must match a field name in the DFD defined under the EmailDFD group. The DPRParseRecord rule copies the data to attachment variables used by other email rules (see the section on attachment variables). This lets you take fields defined in the DFD whatever they are named, and transfer the data to attachment variables that are used by other email rules.

```
< Email2IDS >
    EmailAdd = ADDRESS
    PullCode = REQTYPE
```

XML2Body control group

This control group is used by the DPRFindTemplate rule. Templates are used to predefine text for the body of an email, while variable data is inserted at indicated places within the body of text. The XML2Body control group defines which template files are used for the message body.

The options in the example below, such as e301, e302, and so on, are the values of the ReqType used by the IDS. This value can come directly from the message the IDS receives or from the DPRParseRecord rule. The system must have the ReqType either in the ATTCHDFD.DFD file or set up in the Email2IDS control group.

```
< XML2Body >
    e301 = .\tmpl\e301.txt
    e302 = .\tmpl\e302.txt
    e303 = .\tmpl\e303.txt
    e304 = .\tmpl\e304.txt
    e305 = .\tmpl\e305.txt
    e306 = .\tmpl\e306.txt
    e307 = .\tmpl\e307.txt
    e308 = .\tmpl\e308.txt
    e309 = .\tmpl\e309.txt
    e310 = .\tmpl\e310.txt
    e311 = .\tmpl\e311.txt
    e312 = .\tmpl\e312.txt
```

XML2Attach control group

This control group is used when you need to use template processing to produce an attachment. This group functions just like the XML2Body control group except the output of the template processing is sent as an attachment.

```
< XML2Attach >
    e301 = .\tmpl\e301.txt
    e302 = .\tmpl\e302.txt
    e303 = .\tmpl\e303.txt
    e304 = .\tmpl\e304.txt
    e305 = .\tmpl\e305.txt
    e306 = .\tmpl\e306.txt
    e307 = .\tmpl\e307.txt
    e308 = .\tmpl\e308.txt
    e309 = .\tmpl\e309.txt
    e310 = .\tmpl\e310.txt
    e311 = .\tmpl\e311.txt
    e312 = .\tmpl\e312.txt
```

EmailAdd2Attachment control group

The control group is used by the DPRAdd2Attachment rule to take values from the INI file for the email rules (see section on predefined attachment variable names).

```
< EmailAdd2Attachment >
    default = SUBJECT, Important notice
```

Mail and MailType control groups

These control groups are used by the DPRMail rule to define the email protocol. See the Documaker Workstation Supervisor Guide for information on setting up email support.

These options are identical to the ones set up in the FSIUSER.INI or FSISYS.INI files for a Documaker Workstation.

```
< Mail >
    MailType = SMTP
< MailType:SMTP >
    Name = Send Mail
    Module = SMMW32.DLL
    MailFunc = SMMSendSMTP
    ReplyTo = someone@docucorp.com
    From = JoeJones@docucorp.com
    AltFrom = Joe Jones
    Port = 25
    Server = 10.8.10.216
    Debug = Yes
```

Option	Description
Name	Name of the system (identifies the system on internal dialogs).
Module	Name of the Documaker DSO that supports the email system.
MailFunc	Exported DSO function name of the email handler.

Option	Description

ReplyTo	For SMTP, this option lets you specify a reply-to address.
	The system also lets you specify a different reply-to email address for each IDS transaction. Use this built-in function to specify an attachment variable which contains the value for the ReplyTo option:
	`< MailType:SMTP >`
	`ReplyTo = ~GetAttach REPLYTO,INPUT`
	The value for the ReplyTo option is replaced by the value of the attachment variable in the input attachment with the name REPLYTO.
	The first parameter is the name of the attachment variable, the second is the INPUT or OUTPUT, specifying which attachment is used.
	You can use the built-in INI function in the DAP.INI file or in a particular configuration INI file. You cannot use it in the DOCSERV.INI or DSI.INI files.

From	For SMTP, this option lets you specify who the email was from.
AltFrom	For SMTP, this option lets you specify an alternative from address, indicating where the email was from.
Port	Enter the port.
Server	Enter the address of the server.
Debug	Enter Yes to turn on debugging.

Attachment Variables Used by Email Rules

Messages sent to the IDS are contain attachment variables. Attachment variables can also be used to send information from one rule to another. Some INI options refer to attachment variables. The attachment variables listed below are used by the email rules. For more information on attachments and the email rules, see the SDK Reference.

Variable	Description
XMLTEMPLATTACH	The DPRCreateEMailAttachment rule uses this variable to know which file to open as the template for the attachment. This variable is usually created by the DPRFindTemplate rule.
XMLTEMPLBODY	The DPRCreateEMailAttachment rule uses this variable to know which file to open as a template for the message body. This variable is usually created by the DPRFindTemplate rule.
HTMLATTACHFILE	The file name that contains the output from the template processing used for the attachment. This variable is created by the DPRCreateEMailAttachment rule and contains a path to a file name—not the actual data.
HTMLBODYFILE	The file name that contains the output from the template processing used for the message body. This variable is created by the DPRCreateEMailAttachment rule and contains a path to a file name—not the actually data.

Variable	Description
RECORDLINE(##)	The variable that contains raw data sent to IDS from the client. The format of the data is defined by the DFD specified by the Path option in the EmailDFD control group. There can be any number of these variables but no variable can contain more than 1024 bytes of data. The variables must be defined as follows: `(RECORDLINE00 RECORDLINE01…RECORDLINE99)` These variables must be created by the client program. These variables are processed by the DPRParseRecord rule.
RECORDPARTS	The number of RECORDLINE variables. Processed by the DPRParseRecord rule.
REQTYPE	The transaction identifier code. Used by the DPRFindTemplate and DPRAdd2Attachment rules to determine which template to use for a particular message.
ADDRESS	The email address. Serves as input for the DPRMail rule.
MSGBODY	The body of email. Serves as input for the DPRMail rule.
SUBJECT	The subject of the email.
ATTACHMENT	A file attached to the email—no template processing.

USING EMAIL RULES

You can use the following rules when working with email. For more information, see the SDK Reference.

DPRParseRecord

Use this rule to assemble an attachment into a record and then convert to an XML tree. This rule expects the RECORDLINE## and RECORDPARTS attachment variables in the message it receives. The rule performs the following operations.

1. Takes the data from each of the RECORDLINE## variables and appends them together into one record.
2. The data from this record is converted into an XML tree with variable names in the ATTCHDFD.DFD file used as the variable names for the XML tree.
3. The data from the DFD variables can be transferred to attachment variables. Therefore you can use any variable defined in the DFD to set any of the specific attachment variables listed in the previous section. The Email2IDS control group maps the DFD variable names to the attachment variable names. So you have two groups of variables; the attachment variables and the XML tree variables.

```
< Email2IDS >
    DFD VARIABLE = ATTACHMENT VARIABLE
```

DPRFindTemplate

Use this rule to specify the template file. It expects a REQTYPE variable name to in the attachment. So the DPRParseRecord rule must be executed first to set the REQTYPE variable. It uses the XML2Body control group to define the templates to create the message body and the XML2Attach control group to set up email attachments.

DPRAdd2Attachment

Use this rule to set attachment variables from the INI file. It uses the EmailAdd2Attachment control group in the INI file to set the variable names. It can use a default setting in the INI file or it can use a specific request type, or multiple request types.

```
< EmailAdd2Attachment >
    e301 = SUBJECT, Warning Notice
    e301 = ATTACHMENT, d:\ticket.doc
    default = SUBJECT, Important notice
```

DPRCreateEMailAttachment

Use this rule to perform template processing. This rule can be used for the message body as well an attachment. Template processing uses a text or HTML file to define constant data. Variable data is then inserted at indicated places within the text.

Here is an example of a template text file. You must define the CurrentDate and AcctName fields in the ATTCHDFD.DFD file.

```
<%descendant::COLUMN[ATTRIBUTE::NAME="CurrentDate"],%>

Dear <%descendant::COLUMN[ATTRIBUTE::NAME="AcctName"],%>,

Thank you for opening a certificate of deposit with DeepGreen Bank.
You will receive documents pertaining to your account in the mail
shortly. If you have any questions, please email us at
accountinquiry@deepgreenbank.com or contact our Customer Care
Center at 1-888-888-8888.

Sincerely,
DeepGreen Bank

E301
```

Whether the template is used for the message body or an attachment depends on whether the request type was listed under the XML2Body or the XML2Attach control group. This is determined when the DPRFindTemplate rule is executed.

DPRMail

Use this rule to transfer the data from the attachment variables to the email server. IDS acts as a client to an email server such as Microsoft Exchange. The following attachment variables should be considered as input to this rule.

- ADDRESS
- MSGBODY
- SUBJECT
- ATTACHMENT
- HTMLATTACHFILE
- HTMLBODYFILE

DPRLog

Use this rule to confirm whether an email was sent by IDS. This rule stores information in a log file from either the attachment variables or the XML document created by the DPRParseRecord rule. The DPRMail rule puts the RESULTS attachment variable into the output queue. If no RESULTS variable exists, then the DPRMail rule was not executed and no mail was sent.

USING THE EMAIL BUS

IDS includes an email message bus you can use to receive request messages with or without file attachments from an email inbox. Replies can be emailed asynchronously with or without file attachments to the originators of the request messages.

You can also configure a reply email box as the default reply queue in the server or client configuration files, in which case test utilities or client applications using DSILIB or DocucorpMsg Java package can also communicate with IDS via the email message bus.

The main body part of a request should be formatted in plain text and should contain the XML that should be used as the main body part of the MIME message. Here is an example:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
envelope/">
    <SOAP-ENV:Body>
        <DSIMSG VERSION="100.020.0">
            <CTLBLOCK>
                <REQTYPE>SSS</REQTYPE>
             <UNIQUE_ID>f9db68c1b1c67998662b6cee85a5bdd2</UNIQUE_ID>
            </CTLBLOCK>
            <MSGVARS>
                <VAR NAME="REQTYPE">SSS</VAR>
              <VAR NAME="MAIL.MARSHALLER.XSL.TEMPLATE">sss.xsl</VAR>
            </MSGVARS>
        </DSIMSG>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The content of the main body part of a reply message can be formatted via an XSL template that is accessible to IDS and that is supplied via input message variable MAIL.MARSHALLER.XSL.TEMPLATE. If an XSL template is not provided, IDS returns XML of a format similar to that in the example shown above for a request message.

Malformed email requests are logged in a bad-soap.log file along with a reference ID and IDS will change the request type to EML and send a response along with the same reference ID back to the end user detailing the nature of the error.

For information on these properties, please see the HTML documentation shipped with IDS. You will find this documentation in the following directory:

```
dsi_sdk\java\docs\com\docucorp\messaging\mail\DSIMailMessageQueueFa
ctory.html
```

Be sure to specify the MailDSIMessageMarshaller and the DSIMailMessageQueueFactory classes for the marshaller and queue.factory properties.

Here is an example of a queue configuration section for IDS (docserv.xml):

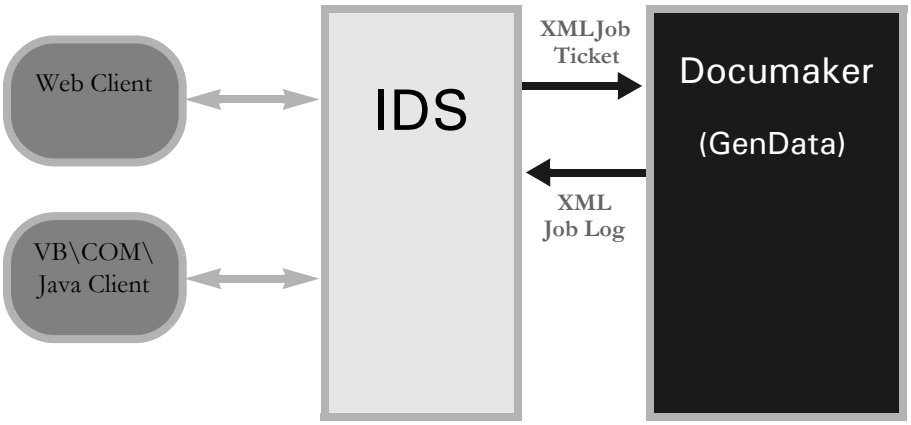
```
<section name="queue">
    <section name="marshallers">
        <entry
name="marshaller.class">com.docucorp.messaging.data.marshaller.Mail
DSIMessageMarshaller</entry>
    </section>
    <!-- input options -->
    <entry
name="queuefactory.class">com.docucorp.messaging.mail.DSIMailMessag
eQueueFactory</entry>
    <entry name="mail.input.server">pop.gmail.com</entry>
    <entry name="mail.input.port">995</entry>
    <entry name="mail.input.user">requestq</entry>
    <entry name="mail.input.password">pdtest123</entry>
    <entry name="mail.input.protocol">pop3</entry>
    <entry name="mail.input.queue">requestq@gmail.com</entry>
    <entry name="mail.input.use.authentication">no</entry>

    <entry name="mail.input.use.ssl">yes</entry>
    <entry
name="mail.input.ssl.socketFactory.class">com.docucorp.messaging.ma
il.ssl.input.DSIMailSSLSocketFactory</entry>
    <entry name="mail.input.ssl.socketFactory.fallback">false</
entry>
    <entry name="mail.input.ssl.protocol">SSLv3</entry>
    <entry name="mail.input.ssl.keystore">c:/docserv/keystore/
cacerts</entry>
    <entry name="mail.input.ssl.keystore.type">JKS</entry>
    <entry name="mail.input.ssl.keystore.manager.type">SunX509</
entry>
    <entry name="mail.input.ssl.keystore.pwd">changeit</entry>
    <entry name="mail.input.ssl.truststore">c:/docserv/keystore/
cacerts</entry>
    <entry name="mail.input.ssl.truststore.type">JKS</entry>
    <entry name="mail.input.ssl.truststore.manager.type">SunX509</
entry>
    <entry name="mail.input.ssl.truststore.pwd">changeit</entry>

    <!-- output options -->
    <entry name="mail.output.server">smtp.gmail.com</entry>
    <entry name="mail.output.port">465</entry>
    <entry name="mail.output.user">resultq@gmail.com</entry>
    <entry name="mail.output.password">pdtest123</entry>
    <entry name="mail.output.protocol">smtp</entry>
    <entry name="mail.output.queue">resultq@gmail.com</entry>
    <entry name="mail.output.use.authentication">yes</entry>

    <entry name="mail.output.use.ssl">yes</entry>
    <entry
name="mail.output.ssl.socketFactory.class">com.docucorp.messaging.m
ail.ssl.output.DSIMailSSLSocketFactory</entry>
    <entry name="mail.output.ssl.socketFactory.fallback">false</
entry>
    <entry name="mail.output.ssl.protocol">SSLv3</entry>
    <entry name="mail.output.ssl.keystore">c:/docserv/keystore/
cacerts</entry>
    <entry name="mail.output.ssl.keystore.type">JKS</entry>
```

```
    <entry name="mail.output.ssl.keystore.manager.type">SunX509</
entry>
    <entry name="mail.output.ssl.keystore.pwd">changeit</entry>
    <entry name="mail.output.ssl.truststore">c:/docserv/keystore/
cacerts</entry>
    <entry name="mail.output.ssl.truststore.type">JKS</entry>
    <entry name="mail.output.ssl.truststore.manager.type">SunX509</
entry>
    <entry name="mail.output.ssl.truststore.pwd">changeit</entry>

    <!-- common mail options -->
    <entry name="mail.debug">yes</entry>
</section>
```

Using IDS to Run Documaker

You can set up IDS to run Documaker as a subordinate process, as shown below. Web clients communicate with IDS using queues. IDS communicates with Documaker via XML files called *job tickets* and *job logs*.

This diagram illustrates the process:



IDS can start or stop Documaker as needed, without user interaction. One IDS session controls one Documaker process. You can, however, implement multiple IDS sessions and have multiple Documaker processes as well.

Keep in mind these limitations:

- You can only run Documaker in single step mode. Consult the Documaker Server System Reference for more information on single step processing.
- Different resource setups for Documaker are supported, but Documaker processing restarts if resources are changed, eliminating the performance benefits. This should not be a problem because it is unlikely multiple Documaker setups will be used with a single IDS implementation. You can, however, experience problems testing a system with multiple setups.
- During processing, some INI options can be changed by the client. Since some Documaker rules use static variables and store INI values in memory, it is possible that a client will be unable to change an INI option if those Documaker rules are used. To handle these situations, you must restart Documaker.
- IDS and Documaker must exist on the same node/server machine.

SETTING UP IDS

To set up IDS to run Documaker, make these changes in the following configuration files:

docserv.xml file

Make these changes in the docserv.xml file, or the configuration file the IDS is configured to use. Here is an example of how to add a request type for Documaker:

```
<section name="ReqType:RPD">
<entry name="function">atcw32->;ATCLogTransaction</entry>
<entry name="function">atcw32->;ATCLoadAttachment</entry>
<entry name="function">atcw32->;ATCUnloadAttachment</entry>
<entry name="function">dprw32->;DPRSetConfig</entry>
<entry name="function">rpdw32->;RPDCheckRPRun</entry>
<entry name="function">rpdw32->;RPDCreateJob</entry>
<entry name="function">rpdw32->;RPDProcessJob</entry>
</section>
```

If necessary, add two more request types, one to check if Documaker is running and one to stop Documaker. Here is an example:

```
<section name="ReqType:CHECK">
<entry name="function">atcw32->;ATCLogTransaction</entry>
<entry name="function">atcw32->;ATCLoadAttachment</entry>
<entry name="function">atcw32->;ATCUnloadAttachment</entry>
<entry name="function">dprw32->;DPRSetConfig</entry>
<entry name="function">rpdw32->;RPDCheckRPRun</entry>
</section>

<section name="ReqType:STOP">
<entry name="function">atcw32->;ATCLogTransaction</entry>
<entry name="function">atcw32->;ATCLoadAttachment</entry>
<entry name="function">atcw32->;ATCUnloadAttachment</entry>
<entry name="function">dprw32->;DPRSetConfig</entry>
<entry name="function">rpdw32->;RPDStopRPRun</entry>
</section>
```

Also add the following IDS rule to the ReqType:INI section:

```
<entry name="function">rpdw32->;RPDStopRPRun</entry>
```

DAP.INI file

Add a configuration option for a the master resource library you will use. Here is an example which is based on the RPEX1 master resource library:

```
< Configurations >
    CONFIG    = RPEX1
< Config:RPEX1 >
    INIFile   = RPEX1.INI
```

RPEX1.INI file

Make these changes in the RPEX1.INI file (or the INI file you are using for your configuration):

```
< IDSServer >
    ExtrPath     = e:\fap\mstrres\rpex1\extract\
    PrintPath    = e:\fap\mstrres\rpex1\data\
    WaitForStart = 60
    SleepingTime = 500
    MaxWaitTime  = 120
    GENSemaphoreName = gendata
```

```
    RPDSemaphoreName = rpdrunrp
    PrintFileCacheTime = 7200
    TextFileCacheTime = 7200
< RPDRunRP >
    Executable    = e:\rel101\shipw32\gendaw32.exe
    Directory     = e:\fap\mstrres\rpex1\
    UserINI      = e:\fap\mstrres\rpex1\fsiuser.ini
    BaseLocation = http://10.8.10.69/fap/mstrres/rpex1/data/
< Printer >
    PrtType      = PDF
< Debug >
    RPDProcessJob = Yes
```

Setting up Multiple Internet Document Servers

The semaphores used by IDS and Documaker are global for a computer, so if you need multiple IDS processes on the same computer, each IDS process and subordinate Documaker process should use different semaphore names.

Semaphore names are generated automatically by IDS for each instance. These names are passed to Documaker as command line parameters. No user intervention is needed.

To specify naming conventions for these semaphores, change these INI options:

```
< IDSServer >
    GENSemaphoreName =
    RPDSemaphoreName =
```

Keep in mind the names must be unique for a computer, so two IDS servers will have to use two different INI files specifying semaphore names.

Controlling Documaker

To control Documaker via IDS, use these IDS rules:

- RPDCheckRPRun - Makes sure Documaker is running. If Documaker is not running, this rule starts it.
- RPDCreateJob - Finds the attachment variables for each of the values in the job ticket and adds them to the XML tree. The XML tree is added to the RPDJobTicket variable so the next rule can use it.
- RPDProcessJob - Gets the XML tree from the RPDJobTicket variable and writes it to a file. This file is used as the job ticket which triggers the Documaker process.
- RPDStopRPRun - Receives the current process ID from the RPDRunProcess variable and then terminates Documaker.

For more information about these rules, see the SDK Reference.

If a critical error is encountered

IDS restarts Documaker Server (GenData) if it encounters a critical error and resubmits the transaction being processed when the error occurred. This helps you handle situations where you have sporadic memory access problems in custom or 3rd-party code.

NOTE: Before the release of version 11.2 shared objects, when the GenData program started, the RPDProcessJob rule communicated with GenData via TCP/IP, sending the job ticket message to GenData and receiving a job log response. If the TCP/IP communication failed, the RPDProcessJob rule forced GenData to stop. This would prepare IDS for the next request.

Version 11.2 shared objects added the ability to automatically restart GenData after the process described above. After it confirms that GenData has been stopped, the RPDProcessJob rule calls the RPDCheckRPRun rule to restart GenData and then calls itself to communicate with GenData and send the same job ticket.

To keep a copy of each transaction, an eight-digit index number is added to the job ticket and job log file names when they are downloaded for information in debug modes.

Also, the system includes these error messages which can appear if there is a TCP/IP failure:

Message	Description
RPD0011	Unexpected program termination of GenData.
RPD0012	Socket connection failure.
RPD0013	Can not unload job ticket to the msg buffer.
RPD0014	Can not load the msg buffer to the job log.
RPD0016	Socket time-out.
RPD0017	Time exceeded the MaxWaitForStart specification.
RPD0018	GenData failure.

The system includes additional information in the log trace file in case of failure. This includes the job ticket, the input attachment variables, and error messages. On the IDS side, this file is named *dprtrc.log*. On the GenData side, this file is the trace file.

Error file processing

You can use INI options to turn on or off the recording of error information when using IDS to run Documaker. This can help in debugging.

To create an error file and write errors into the error file, include these options in the Debug control group:

```
< Debug >
    RPDCheckRPRun  = Yes
    RPDCreateJob   = Yes
    RPDProcessJob  = Yes
    RPDErrFile     = rpderr.dat
```

Option	Description
RPDCheckRPRun	Enter Yes if you want to record any errors encountered when running this rule. If you enter No, errors produced while this rule is run are not recorded.
RPDCreateJob	Enter Yes if you want to record any errors encountered when running this rule. If you enter No, errors produced while this rule is run are not recorded. Be sure to set this option to Yes to record GenData errors.
RPDProcessJob	Enter Yes if you want to record any errors encountered when running this rule. If you enter No, errors produced while this rule is run are not recorded.
RPDErrFile	Enter the name of the error file. The system does not create an error file if you do not enter a name in this field.

Returning record IDs

When you use IDS to run Documaker with WIP and archive rules, the WIP and archived record IDs are written to the print log (PrtLog) file. Furthermore, the first WIP record ID and the first archived record ID are sent to a job log (JobLog) file and are also output as the following attachment variables.

Variable	Description
WIPRECORDID	The first WIP record ID.
ARCRECORDID	The first archived record ID.

These XML elements are added to both the PrtLog and JobLog files:

```
<WIPRECORDID>12345</WIPRECORDID>
<ARCRECORDID>12345</ARCRECORDID>
```

SETTING UP DOCUMAKER

The first step is to set up Documaker to run in a single step mode. See the Documaker Server System Reference for more information.

Keep in mind these considerations...

- If the Documaker programs and DSOs are located on the network, the start time for Documaker can be significant. Keep in mind, however, that the start time only affects the first transaction. Subsequent transactions will process much more quickly. If the start time exceeds 10 seconds, consider changing the WaitForStart option to a higher value.
- All of the standard Documaker performance-related INI options are available even when IDS runs Documaker as a subordinate process. For best results, optimize Documaker's performance before using it with IDS.
- Documaker will run fastest if the resource files for Documaker, as well as input and output files, are physically located on the computer where IDS and Documaker are running.
- Documaker (GenData) automatically creates the XML export file from the transaction and returns the name as XMLOUTPUT and URLXMLOUTPUT.

In addition, you will need to make changes to your FSISYS.INI or FSIUSER.INI files and to your AFGJOB.JDT file.

FSISYS.INI or FSIUSER.INI file

Be sure to turn off all Documaker stop options, as shown here:

```
< GenDataStopOn >
    BaseErrors          = No
    TransactionErrors   = No
    ImageErrors         = No
    FieldErrors         = No
< GenData >
    ClearMsgFile        = Yes
< PrintFormSet >
    MultiFilePrint      = Yes
    LogFileType         = XML
    LogFile             = .\data\printlog.xml
```

Option	Description
GenDataStopOn control group	
BaseErrors	Enter No to prevent the system from stopping on base-level errors.
TransactionErrors	Enter No to prevent the system from stopping on transaction-level errors.
ImageErrors	Enter No to prevent the system from stopping on image-level errors.
FieldErrors	Enter No to prevent the system from stopping on field-level errors.
GenData control group	

Option	Description
ClearMsgFile	Enter Yes to clear the message file (MsgFile) before a job process starts. This prevents the previous job's information from being reused and is necessary when running in single-step mode. The default is No.
PrintFormSet control group	
MultiFilePrint	Enter Yes to generate multiple print files which have a 46-byte unique name. To identify which recipients are in which print batch, enter No or omit this option. This causes the PrintFormSet rule to save the printer for the print batch along with its recipient information. The MultiFilePrint option should only be used with the PDF, RTF, HTML, and XML print drivers.
LogFileType	Specify the type of the log file, such as XML or TEXT.
LogFile	Specify the name and path of the log file, such as `\data\printlog.xml` If you omit the extension, the system uses the LogFileType option to determine the extension.

These INI options are optional:

```
< IDSServer >
    SleepingTime       = 500
    GENSemaphoreName   = gendata
    RPDSemaphoreName   = rpdrunrp
< Debug >
    RULServerJobProc   = Yes
```

Option	Description
IDSServer control group	
SleepingTime	Enter the amount of time in milliseconds you want the system to wait before it checks for a job ticket. The default is 1000 (1 second).
GENSemaphoreName	Semaphore names are generated by IDS for each instance and are passed to Documaker as command line parameters. Use this option to specify naming conventions for semaphore names. The default is *gendata*. Keep in mind semaphore names must be unique for a computer, so two IDS servers will have to use two different INI files specifying semaphore names.
RPDSemaphoreName	Semaphore names are generated by IDS for each instance and are passed to Documaker as command line parameters. Use this option to specify naming conventions for semaphore names. The default is *rpdrunrp*. Keep in mind semaphore names must be unique for a computer, so two IDS servers will have to use two different INI files specifying semaphore names.
Debug control group	

Option	Description
RULServerJobProc	Enter Yes to get a copy of the job ticket file before the system removes it.

AFGJOB.JDT file

Prior to the release of version 11.1, you had to change the base rule from RULStandardBaseProc, as shown here:

```
<Base Rules>
;ServerBaseProc;1;;
...
```

The ServerBaseProc rule replaced the RULStandardJobProc rule and let IDS run Documaker as a separate, *stay alive* process. This meant Documaker only had to start once and IDS could continue even if Documaker failed. For more information on the ServerBaseProc rule, see the Rules Reference.

NOTE: If you are running Documaker version 11.1 or higher, you do not have to substitute ServerBaseProc for RULStandardBaseProc.

Naming Conventions for Output Files

The output files from Documaker use the names generated by the IDS rules and submitted to Documaker in the job ticket file. If you need different names, provide them in the IDS request. In this case, you must make sure the names are unique or else they will be overwritten. The names generated by IDS can consist of up to 45 characters and are similar to the names generated by the DPRPrint rule in IDS.

The directory where the output files are created is determined in this manner:

- If the file name and path was provided, the system uses that information.
- If the file name was provided, but the path was omitted, the system looks for the path in the PRINTPATH attachment variable.
- If the path is not in the PRINTPATH attachment variable, the system looks for the PrintPath option in the DSIServer control group.
- If no path was specified in the PrintPath option, the system places the output file in the current directory.

The extension of the output files is determined in this manner:

- If the name and extension was provided in the attachment, the system uses that information.
- If the name and extension were omitted, the system generates a name and uses the printer type as the extension for the print output files. For other files, the system looks for the FileExt option in the IDSServer control group to find the extension. The default is *DAT*.

CREATING DPW FILES

You can generate a DPW file from GenData or from an import file using IDS. The code is structured as a print driver so setting it up is virtually the same as with print drivers.

The DPW library supports the INI2XML, WIP2DPW, and File2DPW control groups used by the WIP Edit plug-in. To generate a DPW file from Documaker, include these INI options:

```
< Printers >
    PrtType     = DPW
< Printer >
    PrtType     = DPW
< PrtType:DPW >
    PrintFunc   = DPWPrint
    Module      = DPWW32
    Debug       = No
```

Set the Debug option to Yes to capture additional debug information to the trace log.

The PrtType:DPW group can also contain variable names that correspond to index element names in the DPW file. The values specified can be one of these formats:

```
name = val
```

The value will be used as provided, where *val* is the actual value provided.

```
name = ~GVM gvmname
```

If the DPW library is run under the GenData program, the value of the GVM variable matching the name provided is used, where *gvmname* stands for the name provided.

```
name = ~GetAttach attachname
```

If the DPW library is run under IDS, the value of the attachment variable matching the name provided will be used, where *attachname* stands for the name provided.

The system checks the index of the DPW file for the presence of any variables specified in the PrtType:DPW control group and, if present, it updates their values with those specified in the INI file.

Accessing IDS Attachment Variables in GenData

There are times when the GenData program needs access to data passed from IDS which is not in the extract file. To meet this need, the GenData program can access IDS attachment variables as GVM variables. If a GVM variable with the same name already exists, its value does not change.

Here is how it works:

On the IDS side

The RPDCreateJob rule adds any input attachment variables to the XML tree (job ticket) besides the existing variables, such as MsgFile, ErrFile, ExtrFile, LogFile, DbLogFile, NaFile, PolFile, NewTrn, PrtLog, PrtType, ExtrPath, PrintPath, PrintBatches, BatchFiles, IniOptions, EWPSRequest, EWPSResults, ShowErrors, WIPRECORDID, XMLOUTPUT, and so on.

For example, if an attachment variable called RPDTEST is located and it has a value of *This is a test*, it is added to the XML tree as shown here:

```
<DOCUMENT>
<JOBTICKET>
. . .
<RPDTEST>This is a test</RPDTEST>
</JOBTICKET>
</DOCUMENT>
```

On the Documaker side

After the ServerJobProc rule receives the XML tree (job ticket), its child elements are used to update INI values or create GVM variables or both.

Using TCP/IP Communications

IDS and GenData use TCP/IP (socket) to replace the job ticket/job log file I/O communication.

In IDS

The RPDCheckRPRun rule sets up the host name and the port number for a GenData configuration by checking the HOST name and PORT number from these INI options:

```
< IDSServer >
    MaxConfigAllowed = 10
    Host =  localhost
    Port = 49300
```

Option	Description
MaxConfigAllowed	Enter the maximum number of configurations you want to allow. The default is 10.
Host	Enter the host name. The default is localhost.
Port	Enter the base IP port number. The default is 49300.

If you have multiple GenData configurations running over multiple IDS instances, the port number is generated based on the IDS instance number and the base IP port number, so you must decide the base IP port number and the range of IP port numbers. Note that the total number of IP port numbers is decided by:

instances (IDS instance number) x MaxConfigAllowed (allowed to open GenDatas)

If the current configuration differs from the previous configuration, the current configuration starts a GenData process with the assigned IP port number. Both the host name and port number are saved in the configuration structure and are appended to the configuration list. If the configuration exists, the configuration element is extracted and uses the saved host and port number as current.

NOTE: TCP/IP communication is for shared objects 11.1 and above and can not be used with early versions.This rule checks the version to decide whether TCP/IP or file I/O communication should be used before it starts a GenData process.

The RPDProcessJob rule uses the host name and port number to establish communication with GenData and sends a message that contains the XML document (job ticket) to the server (GenData) when it detects that GenData has started. You can set the maximum time to wait for GenData to start using this INI option:

```
< IDSServer >
    WaitForStart = 30
```

Option	Description
WaitForStart	Enter the maximum time to wait for GenData to start in seconds. The default is 30 seconds.

After IDS detects that GenData has started, it sends a message in XML format (the job ticket) and waits for GenData to finish and send back a response in XML format (the job log file). You can set the maximum time to wait for GenData to respond using this INI option:

```
< IDSServer >
    MaxWaitTime = 30
```

Option	Description
WaitForStart	Enter the maximum time to wait for GenData to respond in seconds. The default is 30 seconds.

If GenData does not start before the waiting time elapses, IDS displays this error message on the IDS side:

```
Unexpected Program Termination of GenData
```

In GenData

As soon as GenData starts, it initiates the communication between IDS and GenData using the IP port number retrieved from the command line argument. It receives the job ticket document sent by IDS and continues the GenData process. You can set the maximum time to wait to receive a job ticket using this INI option:

```
< IDSServer >
    MaxWaitTime = 30
```

Option	Description
MaxWaitTime	Enter the maximum time to wait to receive a job ticket in seconds. The default is 30 seconds.

After GenData finishes, the JOBLOG tree is unloaded into a message buffer and is sent to client side as a response message.

You can use options in the Debug control group to determine whether to unload the job ticket and job log files for reference purposes. On the IDS side, set this option to Yes to keep a copy of the job ticket:

```
< Debug >
    RPDProcessJob = Yes
```

On GenData side, set this option to Yes to keep a copy of the job log:

```
< Debug >
    RULServerJobProc = Yes
```

CUSTOMIZING THE EXECUTION OF DOCUMAKER

When IDS runs Documaker in multi-step mode, you can introduce special steps which occur between the GenTrn, GenData, and GenPrint steps. You can use these steps, for instance, to

- Sort the TRNFILE or recipient batches
- Copy files to different locations
- Send files to the printer
- Notify an operator that steps were completed

The RPDRunRP rule can run a custom executable after each step in the process. Use these INI options to define the custom executable name and path:

```
< RPRun >
    PostGenTrnExecutable   =
    PostGenDataExecutable  =
    PostGenPrintExecutable =
```

Option	Description
PostGenTrnExecutable	Enter the name and path of the custom executable, such as a batch file or shell script, you want the system to run after it completes the GenTrn step.
PostGenDataExecutable	Enter the name and path of the custom executable, such as a batch file or shell script, you want the system to run after it completes the GenData step.
PostGenPrintExecutable	Enter the name and path of the custom executable, such as a batch file or shell script, you want the system to run after it completes the GenPrint step.

By default, the following information is passed as parameters to each executable:

```
GenTrn – INI file, trnfile
GenData – INIfile, recipient batches
GenPrint – INI file, print file
```

These INI options are read from the FSIUSER.INI file to create the parameter list. The FSIUSER.INI file is created by the RPDRunRP rule. This INI file is passed to Documaker for each step. Internally, the RPDRunRP rule loads the FSIUSER.INI file and gets parameter information from it.

GenTrn parameters

```
< Data >
    TrnFile = (parameter trnfile)
```

GenData parameters

The system reads all of the options under the Print_Batches control group to determine the recipient batches:

```
< Print_Batches >
    Batch# = (paramter receipient batches )
```

GenPrint parameters

The system reads all of the print batches to determine the printer used and passes the port value for the printer as the parameter.

```
< Print_Batches >
    Batch1 = (value ignored)
< Batch1 >
    Printer = Printer1
< Printer1 >
    Port = (parameter print file)
```

If any of the INI options are missing, the system logs an error. It will, however, try to run the post process without the missing parameter. Memory and list allocation errors result in failure and the system will not attempt to execute the outside process. Here is a list of the potential errors:

Error	Severity
Could not get INI option <Data> TrnFile GenTrn step	non-fatal
Could not create VMM list GenTrn step	fatal
Could not load INI file GenTrn step	fatal
Could not get INI context GenTrn step	fatal
Could not create VMM list GenData step	fatal
Could not load INI file GenData step	fatal
Could not get INI context GenData step	fatal
Could not create VMM list GenPrint step	fatal
Could not load INI file GenPrint step	fatal

Error	Severity
Could not get INI context GenPrint step	fatal
Could not get INI option GenData step <group> <option>	non-fatal
Could not start process: [executable name and command line]	fatal
Memory re-allocation failed	fatal
Memory allocation failed	fatal
PROCStartProcess failed: [command line]	
PROCWaitProcess failed: [executable name and command line]	non fatal
PROCExitCodeProcess failed: [executable name and command line]	non fatal

Using the XML Messaging System

The XML messaging system is an open and documented queue control message format based on XML and the evolving SOAP standard. The XML message format is supported by the JMS, WebSphere MQ, and HTTP messaging systems.

You can find more information on the XML and SOAP on the W3C WEB site:

http://www.w3.org/

You can also find information about SOAP messages with attachments at:

http://www.w3.org/tr/soap-attachments

For information on using SOAP without a messaging system, see Using XML SOAP Outside of Messaging Systems on page 158.

NOTE: Oracle Insurance will follow the evolving standards of SOAP and UDDI and move toward universal messaging. The first version of the DSI message format is based on XML and complies with many of the initial standards for SOAP message envelopes. Later versions will move transactions and servers toward fuller SOAP and UDDI compliance.

Oracle Insurance has used message queuing as a means of serializing requests and responses between loosely coupled clients and servers without requiring one-to-one connections.

Docupresentment includes the client and server sides of the DSI (document server interface) system and of the Oracle Insurance Messaging Library system. These interface layers help manage connections between multiple simultaneous clients and multiple simultaneous servers.

The Oracle Insurance Messaging Library provides a logical abstract layer over the physical process of accessing the queue, so one implementation can support and switch between multiple queueing systems.

The DSI system provides a logical abstract layer over the physical process of assembling, delivering, and parsing of a message, so the initiator of the message does not have to know the physical format of the message, and is insulated from internal software changes to the message format between product versions.

For instance, you can use the DSI messaging client with Documaker Workstation so Documaker Workstation can...

- Interface with external systems via messaging middleware.
- Interface with IDS as a bridge to a legacy system to retrieve data for import.

The first ability means second is optional. You can also use your own internal programs and interface using messaging middleware.

The advantage of having a logical abstract layer is that it lets you deploy applications for different message queuing systems without requiring program changes. Only minimal setup changes are required to test or deploy the same application with a different queuing system.

By abstracting the message format, applications are insulated from internal changes to the message format and can use the Oracle Insurance APIs to correctly assemble or disassemble messages.

The disadvantage of message format abstraction is that non-Oracle Insurance applications might be required to use Oracle Insurance APIs to communicate with Oracle Insurance applications. On some platforms, it may not be practical to invoke these APIs. The proprietary nature of the original message format further complicates the issue.

If you are integrating with IDS as the server, the message format documentation is not necessary. If, however, you are integrating with another application, the message format may be needed if you do not use IDS APIs.

The following topics outline the XML message file format.

The XML-based DSI message format

The DSI message format complies with the following XML-based structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
envelope">
    <SOAP-ENV:Body>
        <DSIMSG VERSION="100.017.0">
            <CTLBLOCK>
              <UNIQUE_ID> { guid hex string } </UNIQUE_ID> (required)
             <REQTYPE> { message request type } </REQTYPE> (required)
                <USERID> { user ID }</USERID> (optional)
                <RESULTQMGR> { remote queue manager } </RESULTQMGR>
(optional)
               <RESULTQ> { remote queue name } </RESULTQ> (optional)
                <ATTACHMENT TYPE="TEXT or BINARY"> (optional)
               <DELIMITER> { tag delimiter } </DELIMITER> (required
for ATTACHMENT)
                </ATTACHMENT>
            </CTLBLOCK>
            <MSGVARS> (required)
                <VAR NAME="VAR NAME 1"> { MSG VAR CONTENT 1 } </VAR>
                <VAR NAME="VAR NAME 2"> { MSG VAR CONTENT 2 } </VAR>
                <VAR NAME="VAR NAME 3"> { MSG VAR CONTENT 3 } </VAR>
            </MSGVARS>
        </DSIMSG>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>


MIME headers
 ... Optional attached text data
 ... Example: a flat text extract file
 ... Example: an XML data file
 ... Example: a base64 (MIME) text encoded binary file
MIME headers
```

Please note:

- The essential component of the message format is the DSIMSG structure, which is encoded inside SOAP-ENV envelope and body structures.
- The indentation of the elements is intended to make it easier to read. Actual messages are not indented.)
- The message can contain attached files. Attached files are encoded inside a tagged structure outside the SOAP-ENV structure. Note that once tagged outside of the primary structure in this fashion, the message file itself is no longer well-formed XML and cannot be viewed with some XML viewers. While it is not well-formed XML, it is a valid SOAP with attachments format.

  The DSI system manages the separation of the attached files from the message. Each ATTACHMENT structure describes the controlling attributes of the attached files. The TYPE attribute specifies the type and format of the attached file, either as *TEXT* (the default) or *BINARY* (MIME format). The DELIMETER element specifies a unique tag name, which is required to be inside the beginning and ending tag brackets to delimit the file data.
- Request and response messages have identical formats. The current specification does not require a distinction between *requests* sent by the client and *responses* returned by the server.
- The client initiating the request generates the UNIQUE_ID. The server echoes the same unique identifier in the response.
- The type of request is identified by the REQTYPE element. Client and server applications must understand and agree on the identifier for the request, the nature of the work to be performed as a result, and the response to be generated.
- The RESULTQMGR and RESULTQ elements are optional, but will appear based on certain types of queue configurations.
- The MSGVARS structure provides the DSI attachment variables which would previously have been encoded using the DSIAddAttachVar rule in the IDS SDK.

Client Request Messages

Here are several example client request messages in XML format:

Without attachments

Here is an example of a client request message in XML format which does not include attachments

```
Content-Type: text/xml
Content-Transfer-Encoding: 8bit

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
envelope">
<SOAP-ENV:Body>
<DSIMSG VERSION="100.017.0">
<CTLBLOCK>
<UNIQUE_ID>a9ae6c91-1d1b-11d2-b21a-00c04fa357fa</UNIQUE_ID>
<REQTYPE>CLAIMS DATA</REQTYPE>
<USERID>JOHN DOE</USERID>
```

```
</CTLBLOCK>
<MSGVARS>
<VAR NAME="CONFIG">FFIC</VAR>
<VAR NAME="KEY1">AUTO BI/UM</VAR>
<VAR NAME="KEY2">CONTACT</VAR>
<VAR NAME="KEYID">123 98 678245</VAR>
<VAR NAME="RUNDATE">20010908</VAR>
<VAR NAME="USERID">JOHN DOE</VAR>
</MSGVARS>
</DSIMSG>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

With attachments

Here is an example of a client request message in XML format which does include attachments:

```
Content-Type: multipart/related; boundary=IDSMessage

--IDSMessage
Content-Type: text/xml
Content-Transfer-Encoding: 8bit

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
envelope">
<SOAP-ENV:Body>
<DSIMSG VERSION="100.017.0">
<CTLBLOCK>
<UNIQUE_ID>a9ae6c91-1d1b-11d2-b21a-00c04fa357fa</UNIQUE_ID>
<REQTYPE>CLAIMS DATA</REQTYPE>
<USERID>JOHN DOE</USERID>
<ATTACHMENT>
<DELIMITER>CLAIMS-DATA</DELIMITER>
</ATTACHMENT>
</CTLBLOCK>
<MSGVARS>
<VAR NAME="CONFIG">FFIC</VAR>
<VAR NAME="KEY1">AUTO BI/UM</VAR>
<VAR NAME="KEY2">CONTACT</VAR>
<VAR NAME="KEYID">123 98 678245</VAR>
<VAR NAME="RUNDATE">20010908</VAR>
<VAR NAME="USERID">JOHN DOE</VAR>
</MSGVARS>
</DSIMSG>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

--IDSMessage
Content-Type: application/ids
Content-Transfer-Encoding: 7bit
Content-ID: CLAIMS-DATA

{…data in a structured COBOL record appears here …}

--IDSMessage--
```

Please note:

- The client initiates the request and generates the UNIQUE_ID. The server echoes back the same unique identifier in the response.
- The MSGVAR structure in this example provides the key fields necessary to access the claims data and create the exported data to be delivered to the client. A server application can receive more variables than are needed and should be set up to ignore those not applicable.
- The ATTACHMENT structure provides the *delimiter* element, which is used to specify the delimiting string pattern that frames a data record passed as an attached file as a part of the message.

With multiple attachments

Here is an example of a client request message in XML format which has multiple attachments:

```
Content-Type: multipart/related; boundary=IDSMessage
        (Please note that this new line must be included.)
--IDSMessage
Content-Type: text/xml
Content-Transfer-Encoding: 8bit
        (Please note that this new line must be included.)
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
envelope">
<SOAP-ENV:Body>
<DSIMSG VERSION="100.017.0">
<CTLBLOCK>
<UNIQUE_ID>a9ae6c91-1d1b-11d2-b21a-00c04fa357fa</UNIQUE_ID>
<REQTYPE>CLAIMS DATA</REQTYPE>
<USERID>JOHN DOE</USERID>
<ATTACHMENT>
<DELIMITER>CLAIMS-DATA</DELIMITER>
</ATTACHMENT>
<ATTACHMENT TYPE="BINARY">
<DELIMITER>CLAIMS-BINARY</DELIMITER>
</ATTACHMENT>
</CTLBLOCK>
<MSGVARS>
<VAR NAME="CONFIG">FFIC</VAR>
<VAR NAME="KEY1">AUTO BI/UM</VAR>
<VAR NAME="KEY2">CONTACT</VAR>
<VAR NAME="KEYID">123 98 678245</VAR>
<VAR NAME="RUNDATE">20010908</VAR>
<VAR NAME="USERID">JOHN DOE</VAR>
</MSGVARS>
</DSIMSG>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
        (Please note that this new line must be included.)
--IDSMessage
Content-Type: application/ids
Content-Transfer-Encoding: 7bit
Content-ID: CLAIMS-DATA

{…data in a structured COBOL record appears here …}
```

```
--IDSMessage

Content-Type: application/ids
Content-Transfer-Encoding: base64
Content-ID: CLAIMS-BINARY
        (Please note that this new line must be included.)
{…data in a base64 encoding form appears here …}

--IDSMessage--
```

Server XML Response Messages

Here is an example of the XML response message from the server:

```
Content-Type: multipart/related; boundary=IDSMessage
        (Please note that this new line must be included.)
--IDSMessage
Content-Type: text/xml
Content-Transfer-Encoding: 8bit
        (Please note that this new line must be included.)
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
envelope">
<SOAP-ENV:Body>
<DSIMSG>
<CTLBLOCK>
<UNIQUE_ID>a9ae6c91-1d1b-11d2-b21a-00c04fa357fa</UNIQUE_ID>
<REQTYPE>CLAIMS DATA</REQTYPE>
<USERID>JOHN DOE</USERID>
<ATTACHMENT>
<DELIMITER>DOCC-XML</DELIMITER>
</ATTACHMENT>
</CTLBLOCK>
<MSGVARS>
<VAR NAME="RESULTS">SUCCESS</VAR>
</MSGVARS>
</DSIMSG>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
        (Please note that this new line must be included.)
--IDSMessage
Content-Type: application/ids
Content-Transfer-Encoding: 7bit
Content-ID: DOCC-XML
        (Please note that this new line must be included.)
<?xml version="1.0" encoding="UTF-8"?>
<DOCUMENT TYPE="RPWIP" VERSION="10.1">
<DOCSET>
<GROUP NAME1="AUTO BI/UM" NAME2="CONTACT">
<FORM NAME="DEC PAGE">
<DESCRIPTION>Common Policy Declarations</DESCRIPTION>
<RECIPIENT NAME="AGENT" COPYCOUNT="1"/>
<RECIPIENT NAME="HOME OFFICE" COPYCOUNT="1"/>
<RECIPIENT NAME="INSURED" COPYCOUNT="1"/>
<SHEET>
```

```
<PAGE>
<SECTION NAME="CPDEC&#126;1">
<FIELD NAME="DELIVERY">DELIVERY NOTE</FIELD>
<FIELD NAME="ON_ARRIVAL">ON ARRIVAL NOTE</FIELD>
<FIELD NAME="CLAIMANT_NAME">SUSAN FRIEDEN</FIELD>
<FIELD NAME="CLAIMANT_ADDRESS">ADDRESS HERE</FIELD>
<FIELD NAME="POLICY_NUMBER">POLICY NUM 22222</FIELD>
<FIELD NAME="SALUTATION">SALUTATION FIELD</FIELD>
<FIELD NAME="INSURING_COMPANY">INSURING COMPANY N</FIELD>
<FIELD NAME="TAG_LINE">TAG LINE FOR THE IN</FIELD>
</SECTION>
</PAGE>
</SHEET>
</FORM>
</GROUP>
</DOCSET>
</DOCUMENT>
--IDSMessage--
```

Please note:

- The client generated UNIQUE_ID is echoed back in the response.
- The ATTACHMENT structure specifies the delimiter for the embedded XML export file. In this example, the file is called *DOCC-XML*.
- The MSGVAR structure specifies the returned attachment variables, which include the results from the requested operation, returned in the variable named *RESULTS*. In this example, the result is *SUCCESS*. If there is an error, the result is an error code and, typically, no XML export data is included.
- If you are transmitting messages between dissimilar platforms, say an EBCIDIC platform such as an MVS-based server application which is submitting messages to an ASCII platform such as a PC, you must set the message format attribute in the message descriptor to *string* (text). This lets the MQ Series channel sender/receiver perform the EBCDIC-to-ASCII translation. Likewise, the QSRLIB layer of the Oracle Insurance system sets the request message format to *string* so the ASCII-to-EBCDIC translation takes place. As a result, client and server applications are able to see the message data in the proper format and do not have to perform the translation themselves.

Using XML SOAP Outside of Messaging Systems

Using the DSIGetSOAPMessage and DSIGetSOAPMessageSize functions, you can code IDS client applications (such as iDocumaker) with common APIs using XML DOM of the IDS SOAP XML message. See the SDK Reference for more information on these functions.

These APIs let client applications access the DSI XML message as a buffer in memory. Access to IDS XML message is provided as a buffer in memory because of possible issues with the version of the DOM or XML parser the client application may be using.

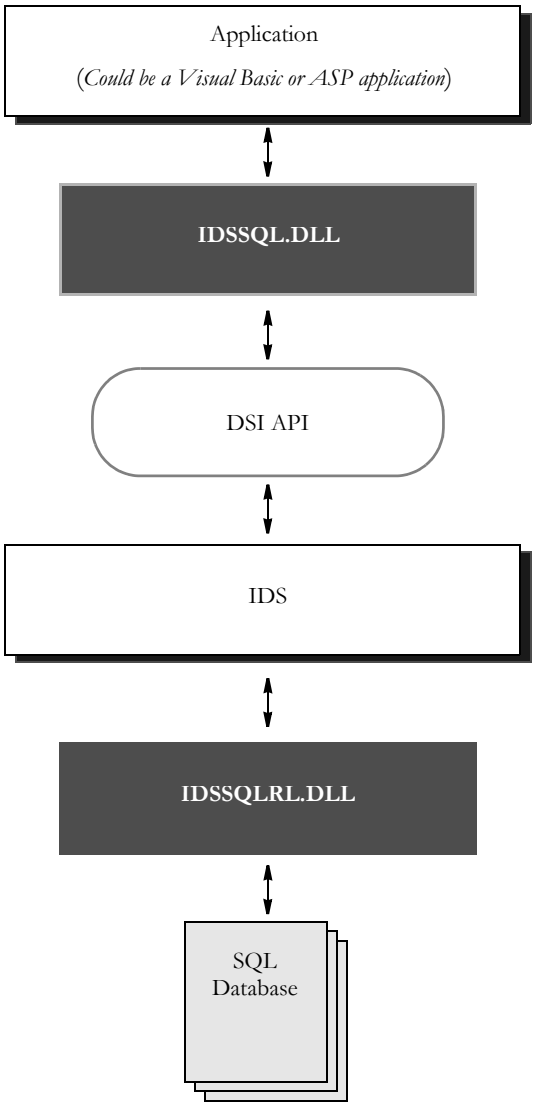
Keep in mind the XML returned is a byte array using UTF-8 encoding.

NOTE: The GetSOAPMessage is also available for the COM and Java APIs. (DSICO, IDSASP and DSIJava).

Connecting to an SQL Database

IDSSQL is a set of ActiveX® DSOs (IDSSQL.DLL and IDSSQLRL.DLL) which you can use as a Microsoft ActiveX Data Objects (ADO) programming model.

These DSOs let you send SQL commands and receive records back from an SQL database. Instead of communicating directly with the database as an object, the IDSSQL DLLs go through an IDS rule. This illustration shows how it works:



Differences between Microsoft's ADO and IDSSQL

Keep in mind these differences between ADO and IDSSQL:

- IDSSQL does not implement all features of Microsoft's ADO and record set.
- The connection between IDSSQL and the SQL database automatically opens and closes on each execute.
- The new record insert into the database is made using the SQL insert command instead of the insert and update method in the ADO record set.
- Errors are returned through IDS record sets. So it good to have a record set even if the SQL command did not return any records.

Setting up IDSSQL

Follow these steps to set up IDSSQL:

1. Add these options to your DOCSERV.INI file:

```
< ReqType:IDSSQL >
    Function = atcw32->ATCLoadAttachment
    Function = DSICoRul->Invoke,IDSSQLRL.IDS->SQL
    Function = atcw32->ATCUnloadAttachment
```

2. Set up the ODBC Data source name.

IDSSQL Classes

Here are the properties and methods for IDSSQL.ADO and IDSSQL.IDSRC:

IDSSQL.ADO

Here are the properties and methods for IDSSQL.ADO:

Properties

Property	Description
AbsolutePage	The ordinal position of the current page. The default is zero (0). If zero (0), all records queried by the SQLcommand are returned. If set to something other than zero, only those records on the page are returned. The number of records on the page are determined by the PageSize property.
CommandTimeOut	The number of seconds to wait when executing a command before terminating the attempt and returning an error. If you set this property to zero, ADO will simply wait until the execution is complete. The default is 30 seconds.
DSN	The ODBC data source name or the information used to create a connection to data source.
PageSize	The number of records on a page. The default is 10.

Property	Description
Password	The password used to connect to the database. A password is required if the DSN connection is not a trusted SQL Server connection.
SQLCommand	The SQL statement.
User	The user ID used to connect to the database. The user ID is required if the DSN connection is not a trusted SQL Server connection.

Methods

Method	Description
Execute	Process SQL command. Returns the record set requested by SQLCommand.

IDSSQL.IDSRC

Here are the properties and methods for IDSSQL.IDSRC, the IDS record set.

Properties

Property	Description
BOF	True if the current record position is before the first record.
EOF	True if the current record position is after the last record.
Errors	Errors collection.
Field	Each field of the current record.
Fields	Fields information collection.
RecordCount	The number of records currently in the record set.

Methods

Method	Description
MoveFirst	Move to the first record in the record set and make that the current record.
MoveLast	Move to the last record in the record set and make that the current record.
MoveNext	Move to the next record in the record set and make that the current record.
MovePrevious	Move to the previous record in the record set and makes that the current record.

Example Script

Here is an example in ASP:

```
<%@ Language=VBScript %>
<HTML>
```

```
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<%

CrLf = Chr(13) & Chr(10)
set sql = Server.CreateObject("idssql.ado")   'Create IDS ADO Object
set rc = Server.CreateObject("idssql.idsrc")  'Create IDS record set

ShowAllActivateAccount()

Sub ShowAllActivateAccount()
sql.DSN = "COB_TEST"'ODBC Data source name
sql.SQLCommand = "Select * from subscriberdata where ebppstatus =
'A'"
Set rc = sql.Execute()‘Execute Command

If rc.Errors.Count <> 0 Then‘Check for error
    Response.write "Error source = " & rc.Errors(1).Source & "<BR>"
    Response.write "Error Description = " & rc.Errors(1).Description
& "<BR>"
Else
%>
    <TABLE BORDER=1>
    <TR>
        <TH>RC#</TH>
        <TH>Account#</TH>
        <TH>Name</TH>
    </TR>
<%
    ‘Loop through all the return records and display the fields
    For i = 1 To rc.RecordCount
        Response.write "<TR>" & Crlf
        Response.write "<TD>" & i & "</TD>"
        Response.Write "<TD>" & rc.Field("Accountnumber") & "</TD>"
        Response.Write "<TD>" & rc.Field("SubscriberFirstName") & " "
        Response.Write rc.Field("SubscriberMiddleName") & " "
        Response.Write rc.Field("SubscriberLastName") & " "
        Response.Write "</TD>" & Crlf
        Response.write "</TR>" & Crlf
        rc.MoveNext‘Move to next record
    Next
%>
    </TABLE>
<%
End If

End Sub
%>
<P> </P>

</BODY>
</HTML>
```

Fields Here is an example of how you can access the name and data in the field of each record:

```
For i = 1 To rc.RecordCount‘Loop through all return record set
   Response.Write "========="
   Response.Write "Record " & i
   Response.Write "========="
     For j = 1 To rc.Fields.Count‘Loop through all the fields in
that record
        Response.Write rc.Fields(j) & ":" ‘Display field name
        Response.Write rc.Field(j)‘Display data in the field
     Next
Next
```

You can access the data in the field using the name or index, such as:

Rc.Field(1) or *Rc.Field("SubscriberID")*

Using the Thin Client Forms Publisher

The Thin Client Forms Publisher lets web clients enter a user ID, password, and other information at login. Depending on how you set it up, IDS then returns a list of group1/group2 combinations for the form set.

The web client can then choose a group1/group2 combination and submit it to IDS along with an effective date. The DPRSetConfig rule sets the effective date for use with Library Manager.

IDS then returns a new XML form set (through the result queue) based on the group1/group2 submittal. The Thin Client Forms Publisher loads the XML form set returned by IDS and generates an HTML tree view.

The web client can then select the forms, images, recipients, and print options and submit a request to print the form set. Once submitted, the Thin Client Forms Publisher generates a new XML form set and sends it to IDS.

IDS retrieves the new XML form set, converts it into a FAP form set and prints it. IDS then sends the final output file to the Thin Client Forms Publisher through the queues. The Thin Client Forms Publisher supports these print options: PDF, XML, PCL, AFP, MET, and HTML.

Look at these examples for more information:

Example	Uses
/formpub	dp018.dll which you can use on Windows 2000 without IDS DLL files
/formpubnt	idsasp.dll for standard messaging and dp018.dll for XML processing

These virtual directories are included in the IDS sample resources. Use *FORMAKER* for the user ID and password when viewing the examples.

PAUSING IDS

When necessary, you can pause IDS processing and then restart it. For instance, suppose you are running a system with multiple instances of IDS, each running its own Documanage Bridge, with each Documanage Bridge logging into a separate Documanage system.

In this scenario, you want IDS to become passive (stop processing requests) when the Documanage system becomes unavailable and to become active again when the Documanage system becomes available again.

This fail-over strategy avoids situations where IDS is processing requests which are failing because the bridge is having a problem with Documanage server.

To handle this scenario, you use a C or Java DSI API. This API lets a rule request that IDS go into *pause mode*. While in pause mode IDS will not receive requests from a queue and will execute only one request type PAUSE. The frequency of this request is defined using this option in the configuration file:

```
<section name="BusinessLogicProcessor">
 ....
  <section name="messaging">
    <section name="timed">
      <entry name="PauseCheckIntervalSeconds">10</entry>
 </section>
```

The entry PauseCheckIntervalSeconds is the interval that IDS will send PAUSE requests to itself when it is paused.

The Documanage Bridge calls the API and places IDS in pause mode when Documanage server becomes unavailable. The rule registered on the PAUSE request type checks to see if the Documanage server is available calls an API to resume the IDS operation.

You use these DSI APIs:

DSIQueryStatus

This API returns DSI specific status options via DSISTATUS_* flags. Use it to determine if IDS is in a paused mode. Here is a list of the available flags:

Flag	Description
DSISTATUS_PAUSE	Pause the server
DSISTATUS_STOP	Stop the server and exit the process
DSISTATUS_RESTART	Restart the server
DSISTATUS_RESUME	Resume the server after a pause

NOTE: Setting the status to DSISTATUS_STOP is non-recoverable action. Once the server exits, no other actions are possible.

Syntax

```
DSIQueryStatus()
```

Parameter	Description
hdsi	handle to instance returned by DSIInitInstance
plOptions	pointer to a long for returning the DSISTATUS_* values.

Returns

DSIERR_SUCCESS or an error code.

Errors

Message	Description
DSIERR_INVPARM	Invalid DSI instance handle or plOptions is NULL
DSIERR_INTERNAL	Internal error

Example

Here is an example:

```
long lOpt;

if ( DSIQueryStatus(hInstance,&lOpt) != DSIERR_SUCCESS ) {
... display error message
}
if ( lOpt & DSISTATUS_PAUSE )
{
printf("Server is currently paused\n");
}
if ( lOpt & DSISTATUS_STOP )
{
printf("Server is currently stopping\n");
}
```

DSISetStatus

This API sets DSI specific status options via DSISTATUS_* flags. Use it to pause IDS. Here is a list of the available flags:

Flag	Description
DSISTATUS_PAUSE	Pause the server
DSISTATUS_STOP	Stop the server and exit the process
DSISTATUS_RESTART	Restart the server
DSISTATUS_RESUME	Resume the server after a pause

NOTE: Setting the status to DSISTATUS_STOP is non-recoverable action. Once the server exits, no other actions are possible.

Syntax

```
DSISetStatus()
```

Parameter	Description
hdsi	handle to instance returned by DSIInitInstance
plOptions	pointer to a long for returning the DSISTATUS_* values.

Return values

DSIERR_SUCCESS or an error code.

Errors

Message	Description
DSIERR_INVPARM	Invalid DSI instance handle
DSIERR_INTERNAL	Internal error

Example

Here is an example:

```
if ( DSISetStatus(hInstance,DSISTATUS_PAUSE) != DSIERR_SUCCESS ) {
... display error message
}
printf("Server is currently paused\n")
```

When running a Java rule, the RequestState parameter has methods for pausing and resuming IDS as well as to check to see if it is currently paused.

Method	Description
isRequestProcessorPaused	Returns true if IDS is currently paused, false otherwise.
pauseRequestProcessor	Tells IDS to pause and stop processing requests from queues and so on.
resumeRequestProcessor	Tells IDS to resume processing requests from queues and so on.

Executing Request Types at Run Time

IDS lets you execute request types composed at run time. Client programs can specify their own XML configuration file with a set of request types to process. Multiple client programs can have request types with the same name but with a different set of rules to run. Request types no longer have to be present in the IDS configuration file.

To execute request types at run time, specify an attachment variable named DYNAMIC-CONFIGURATION-FILE with a full path and file name for a configuration file accessible to IDS.

Here is an example:

Example configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <section name="ReqType:POC-RUNRP-HTM">
        <entry name="function">atcw32->ATCLoadAttachment</entry>
        <entry name="function">atcw32->ATCUnloadAttachment</entry>
        <entry name="function">atcw32->ATCSendFile,
RPOUTPUT,INSURED,BINARY</entry>
        <entry name="function">atcw32->ATCReceiveFile,
EXTRACTFILE,EXTRFILE,c:\docserv\data\*.xml,KEEP</entry>
        <entry name="function">dprw32->DPRSetConfig</entry>
        <entry name="function">RPDW32->RPDCheckRPRun</entry>
        <entry name="function">RPDW32->RPDCreateJob</entry>
        <entry name="function">RPDW32->RPDProcessJob</entry>
  </section>
</configuration>
```

Example data file

```
<?xml version="1.0" encoding="UTF-8"?>
<message>
  <data>
    <var name="CONFIG">POC</var>
    <var name="BATCHFILES">3</var>
    <var name="INIOptions">1</var>
    <var name="INIOptions1.Group">Printer</var>
    <var name="INIOptions1.Option">PrtType</var>
    <var name="INIOptions1.Value">HTM</var>
    <var name="OUTPUTTYPE">HTM</var>
    <var name="PRINTBATCHES">3</var>
    <var name="SHOWERRORS">YES</var>
    <var name="REQTYPE">POC-RUNRP-HTM</var>
   <var name="DYNAMIC-CONFIGURATION-FILE">c:\docserv\runrp-poc-htm-
config.xml</var>
  </data>
  <attachments>
    <file name="EXTRACTFILE">C:\msgclient\extract\poc.xml</file>
  </attachments>
</message>
```

Example dynamic.htm page

```
<html>
<head>
<h2>dynamic config test</h2>
<body>
<form action="dynamic2.asp" enctype="multipart/form-data"
method="post">
<table>
  <tr>
```

```
  <td>*Enter an xml message with name/value pairs to process</td>
  <td><a href="data.xml">example</a></td><td><input name="xml-
message" type="file"/></td>
  </tr>
</table>
  <input type="submit" name="" value="submit">
</form>
```

Example dynamic.asp page

```
<%

    set parser = server.CreateObject("IDSASP.DSI")

    parser.parseData()

    message = parser.getBuffer("xml-message")

    '***process the message file

    processMessageFile message

    parser.showAtt = 1

    parser.ProcessRq




    function processMessageFile(buffer)

      set xml = Server.CreateObject("MSXML2.DOMDocument.4.0")
      xml.loadxml(buffer)


      set msgVars = xml.selectNodes("message/data/var")

      for each var in msgVars
        name = var.getAttribute("name")
        value = var.text
        parser.addReq name, value
      next

      set Attachments = xml.selectNodes("message/attachments/file")
      for each attach in Attachments
        name = attach.getAttribute("name")
        value = attach.text
        parser.SendFile name, value
      next

    end function
%>
```

Publishing Your Forms on the Web

The system provides tools you can use to create one HTML file and a number of PDF files (one per FAP file). You can then publish these HTML and PDF files on your web server.

The HTML file lists all of the company, line of business (LOB), form, and image combinations from the FORM.DAT file. This file has links to PDF files which are created for each image.

To publish your form library, you have to put these HTML and PDF files in a web server contents directory. Once you have the HTML and PDF files in a contents directory, you can use your browser to open the HTML page and view the images.

NOTE: The one PDF file per image concept does not work well in an environment which has a lot of small images. This concept works better with full page FAP files.

Use the following tools to publish your forms on the web:

- FORMPUB - This program provides a graphical interface from which you can run the FD2HTW32 and PTFMDW32 utilities.
- FD2HTW32 - This utility converts a FORM.DAT file into an HTML page. You can run this utility as a stand-alone program or from the FORMPUB tool.
- PTFMDW32 - This utility creates a PDF file for each image in the FORM.DAT file. You can run this utility as a stand-alone program or from the FORMPUB tool.
- FAP2HTML - This utility lets you convert FAP files into HTML format for use with iPPS as data entry screens.

FORMPUB

This utility provides a graphical interface from which you can run the FD2HTW32 and PTFMDW32 utilities. If you want to run these utilities as stand-alone programs, read the following descriptions of these programs.

FD2HTW32

Use this utility to convert a FORM.DAT file into an HTML page.

Syntax

```
fd2htw32 [-i=<formdef>] [-ini= <inifile>] [-o=<outfile>] [-d=<dirname>]
```

Parameters

Parameter	Description
i	The form definition file from which to read. Defaults to the FORM.DAT file.
ini	The initialization file from which to read. Defaults to the FSIUSER.INI file.
o	The name of the output HTML file.
d	The name of the web server directory.

PTFMDW32

Use this utility to create a PDF file for each image in the FORM.DAT file.

Syntax

```
ptfmdw32 [-i=<formdef>] [-ini=<inifile>] [-f=<formlib>] [-x=<fxrfile>] [-o=<outdir>]
```

Parameters

Parameter	Description
-i=<formdef>	The form definition file from which to read. Defaults to *FORM.DAT*.
-ini=<inifile>	The initialization file from which to read. Defaults to *FSIUSER.INI*.
-f=<formlib>	Location of FAP files.
-x=<fxrfile>	Full name of font cross-reference file (FXR file).
-o=<outdir>	Location of output PDF files. Defaults to the same directory as FAP files.

FAP2HTML

Use this utility to convert FAP files into dynamic HTML files.

Program names

Windows 32-bit FAP2HTML.EXE

Syntax

```
FAP2HTML /I /TS /D /X /INI
```

Parameter	Description
/I	The name of the FAP file. You can use wildcards, such as *.FAP.
/ts	Include this parameter to tell the utility to produce HTML output for TerSub paragraphs. TerSub is a pre-and post-edit function which selects and assembles pre-written, standardized paragraphs as a time-saving feature.
/d	Include this parameter to include output that can help you diagnose any problems that occur.
/X	The name of the FXR file (font cross-reference file).
/INI	(Optional) The name of the INI file to use. The default is FSIUSER.INI.

This utility creates an HTML file for each FAP files you specify. The utility appends the extension *HTM* to the output files. You can then display the HTML files using a browser.

Keep in mind:

- Dynamic HTML commands require Microsoft Internet Explorer 4.0 or later.
- If the FAP form is complex, you may experience problems in older versions of Internet Explorer. To avoid such problems, use Internet Explorer 6.0 or later.
- Using absolute positioning, dynamic HTML commands make the HTML page the exact size of the FAP page. This means you cannot zone the HTML file in a browser.
- A multi-page FAP file is converted into multiple HTML files, one for each page. Pages have *_p#* appended to the FAP file name with the *HTM* extension. For example, the first page has *_1* appended to the end of the FAP file name with the *HTM* extension. The second page has *_2* appended to the end of the FAP file name with the *HTM* extension, and so on.
- True Type font names are retrieved from the FXR (font cross-reference) file. The computer used to display the output HTML files must have these same fonts or the output may differ.

  The mapping to the True Type font occurs in the Window32Subs control group. Here you can specify, for example, a Times family font and map it to the True Type equivalent, Times New Roman. You do not have to change your FXR file, just make sure you have the correct mappings in the INI file.
- This utility does not convert logos. It will, however, set up references to the logo file. You must use a graphics file conversion utility (not included) to convert the logo files into GIF or JPEG files.

- To create TerSub paragraphs, include the /TS and /D parameters.

INI Options

You can use these INI options to customize how the FAP2HTML utility works:

```
< PrtType:HTM >
    SplitText   =
    Field       =
    Text        =
    TextMerge   =
    Box         =
    Barcode     =
    Bitmap      =
    ImagePath   =
    ImageExt    =
    Table       =
    FieldFontFudge =
    DirLinks    =
    CollapsePage=
    PageBreaks  =
    HR          =
    AllowInput  =
```

Option	Description
SplitText	Use this option to specify the number of characters to output as a separate text label. If you set this set option to -1 (the default), each word is output as a separate word. If you set this option to zero (0), the system will not split the text. Splitting the text on every character produces a better fidelity HTML file, but slows the performance of the utility and of the browser. The default value seems to produce good results. If you plan to later edit the HTML file, set this value to zero (0) so all the text is output together without positioning commands.
Field	Enter No if you want to omit this kind of objects from the HTML file. The default is Yes.
Text	Enter No if you want to omit this kind of objects from the HTML file. The default is Yes.
TextMerge	Enter No if you want to omit this kind of objects from the HTML file. The default is Yes.
Box	Enter No if you want to omit this kind of objects from the HTML file. The default is Yes.
Barcode	Enter No if you want to omit this kind of objects from the HTML file. The default is Yes.
Bitmap	Enter No if you want to omit this kind of objects from the HTML file. The default is Yes.

Option	Description
ImagePath	Use this option to specify the location of the graphics files. Use a relative path, such as */images/*, so the reference to the logo will be output in the HTML file with the attribute `SRC="images/GRAPH1BB.jpg"` Be sure to use this option if you keep the graphics in a separate directory on the web server.
ImageExt	Use this option to specify the extension to use for references to graphic files. The default is *JPG*. You will need to set this option to the correct value, if your bitmaps are converted into some other file format, such as GIF, PNG, or TIFF.
Table	Use this option to specify if the dynamic HTML absolute positioning should be used for the text. The default is No. When HTML tables are used for text positioning, the FAP lines and boxes are not output and the fidelity of the output document is low. Set this option to Yes to edit the HTML text after the conversion.
FieldFontFudge	Use this option to increase or decrease the font size for the input fields. The default is 0.535. This option is necessary because browsers usually use fonts larger than the input fields, so the top or bottom of the text inside the input field is chopped off. When you use this option, the font size inside the input field is the font height times this value.
DirLinks	Use this option to add Next and Prev links on pages. The default is No.
CollapsePage	Use this option to eliminate white space between HTML pages. The default is No.
PageBreaks	Use this option to force a page break between pages during HTML print. The default is Yes.
HR	Use this option to display a line between HTML pages. You can configure the size, color and width. Here is an example: `HR = Size=2 Width=100% Color=Black`
AllowInput	Set this option to Yes to enable variable fields for entry. The default is No.

Handling Multi-Part/ Form-Data Forms

You can use JSP pages to process HTML forms encoded as *multi-part/form-data*. This encoding enables file uploading from HTML pages to IDS for processing.

Here is a sample HTML form that includes uploading an attachment that is loaded from a file on the user's machine. Any request type can be used. This example uses the LGN request type.

The attachment name should match an attachment you are expecting in your rules. In this example the attachment is named *IMAFILE*. Attachments are always sent in binary format.

```
<FORM METHOD=POST ENCTYPE="multipart/form-data" ACTION="login.jsp">
  <INPUT TYPE="FILE" NAME="IMAFILE"/><BR>
  <INPUT NAME="REQTYPE" value="LGN" TYPE="HIDDEN"> <BR><BR><BR><BR>
  <FONT FACE="Arial,Lucinda Sans,Gill Sans" COLOR="#0000FF">
  <H3>
  <B>User ID: </B> <INPUT SIZE=10 MAXLENGTH=8 NAME="USERID"
value="FORMAKER"> <BR><BR>
  <B>Password: </B> <INPUT TYPE=PASSWORD SIZE=8 MAXLENGTH=8
NAME="PASSWORD" VALUE="FORMAKER"><P>
  <B>Archive: </B>
  <SELECT NAME="CONFIG">
   <OPTION VALUE = "UTILITY">Utility Company
   <OPTION VALUE = "FINANCE">Worldwide Financial
  </SELECT>
  <P>
  <INPUT TYPE="submit" VALUE="Login">   <INPUT
TYPE="reset" VALUE="Reset"><P>
  </H3>
  </FONT>
  <BR><BR>
  </FORM>
```

Formatting Text with XML Markup

iPPS and other Documaker clients can format multi-line text fields in XML files. The XML import loader and export unloader, which are the same for Documaker and IDS, support these formatting attributes:

```
Italic: <I>
Bold: <B>
Underline: <U>

Font <FONT>
Attributes:
SIZE=99 (point size)
FACE=(font family name)
COLOR=(hex color value, such as #FFFFFF)

Paragraph: <P> or <BR>
Attributes: ALIGN="CENTER" or "RIGHT"
```

If you omit the alignment, the system left justifies the text. Empty paragraphs use a <BR> element instead of <P>.

Here is an example multi-line field input or output XML:

```
<P ALIGHN="CENTER">
<FONT SIZE="15" FACE="Universal"><B>This is bold 15 points size
font</B></FONT>
<FONT SIZE="10" FACE="Universal"><I><U>This is italic size 10 point
font with underline</U></I></FONT>
</P>
```

If the font does not exist, the font locator looks for the best match based on font family name, point size, style, and weight.

If you omit the font information from the import file, the system uses the default font for the text area.

Encrypting and Decrypting Data Files

IDS includes a utility you can use to encrypt and decrypt data files. The program is a Java class in the DocuCorpUtil.jar library. To run it, enter a command similar to the one shown here:

```
java -cp DocuCorpUtil.jar com.docucorp.util.DataCrypt
```

Here is a summary of the parameters:

Parameter	Description
-i	Treat the text argument as a file name instead of text to encrypt/decrypt.
text	The text to encrypt/unencrypt or the name of a file if the -i parameter is included. The input file is overwritten with the new information.
-u	Include this parameter if you want to decrypt the text or file instead of encrypting it. Encrypting is the default behavior for this utility.

If you omit all of the parameters, a usage message appears.

Using Multiple Attachment Values with the Same Name

IDSASP and IDSJSP let you send and receive messages with multiple attachment variables which have the same name. To enable support for multiple attachment variables, set the ProcessAll property to True at the beginning of an ASP or JSP page See the example pages below.

In ASP you can simply traverse through the Request and Result collections as before to retrieve all entries for a message. In JSP you can use the getEntries() API to return a list of MsgVarEntry objects.

Here is an example ASP page:

```
<%

    set dsi = server.createobject("IDSASP.DSI")

    dsi.ProcessAll = True

    dsi.addReq "USERID", "DOCUCORP"
    dsi.addReq "USERID", "FORMAKER"
    dsi.addReq "USERID", "DEMO1"
    dsi.addReq "USERID", "DEMO"

    dsi.addReq "PASSWORD", "P1"
    dsi.addReq "PASSWORD", "P2"

    dsi.addReq "REQTYPE", "TEST_MVARS"

    For I = 1 To dsi.Request.count
        Response.Write "Request " & I & ": "
        Response.Write dsi.Request(i).NAME & " = " &
dsi.Request(i).Value
        Response.Write "<BR>"
    Next

    dsi.processRq

    For I = 1 To dsi.Result.count
        Response.Write "Result " & I & ": "
      Response.Write dsi.Result(i).NAME & " = " & dsi.Result(i).Value
        Response.Write "<BR>"
    Next


%>
```

Here is an example JSP page:

```
<%@ page language="java" import="java.util.*,
                                 java.net.*,
                                 java.io.*,
                                 com.docucorp.messaging.data.*" %>


<jsp:useBean id='dsi' scope='page' class='com.docucorp.ids.jsp.dsi'/
>
<%
    int _OUTPUTQUEUE = 1;
    int _INPUTQUEUE = 2;
    List entries = null;

    dsi.setTimeout(30000);
    //dsi.debugOn(response);
    dsi.ProcessAll = true;

    dsi.addRequest("USERID", "DOCUCORP");
    dsi.addRequest("USERID", "FORMAKER");
    dsi.addRequest("USERID", "DEMO1");
    dsi.addRequest("USERID", "DEMO");

    dsi.addRequest("PASSWORD", "P1");
    dsi.addRequest("PASSWORD", "P2");

    dsi.addRequest("REQTYPE", "TEST_MVARS");


    entries = dsi.getEntries(_OUTPUTQUEUE);
    for (int I =0; I <entries.size(); i++){
            String k = MsgVarEntry.getName(entries, i);
            String v = MsgVarEntry.getValue(entries, i);

        out.println("Request: " +  k + "=" + v + "<br>");

    }

    dsi.ProcessRequest();

    entries = dsi.getEntries(_INPUTQUEUE);
    for (int I =0; I <entries.size(); i++){
            String k = MsgVarEntry.getName(entries, i);
            String v = MsgVarEntry.getValue(entries, i);

            out.println("Response: " +  k + "=" + v + "<br>");
    }

%>
```

getEntries

Use this API to return a list of MsgVarEntry objects. Each MsgVarEntry object contains a name and value property.

Parameters

Parameter	Description
queue	An integer value that indicates which queue the entries should be returned for: 1 = Output queue, 2 = Input queue.

Returns A list of MsgVarEntry objects (see the JSP example page).

Converting XML Files Using a Template

You can use the XsltTransformRule rule to transform input into the desired output based on an xsl template. For instance, you can transform an XML extract file located using the EXTRFILE input attachment variable into a new output file or a set of XML files located in the path specified by the XMLPATH input attachment variable, appending the results from each one to the end of the file. You can also transform a single XML file located by the XMLFILE or SOURCE input attachment variables.

This rule can also transform the result XML message in the queue. The output depends on the xsl template provided.

This rule takes an argument of name RUNMSG which can have a value of 1-4 to specify whether the rule should be run in the INIT (1), TERM (2), RUNF (3), or RUNR (4) message. The default is RUNF (3) message if no RUNMSG argument is specified. This is useful when the rule that outputs the XML source does not run in the default RUNF message.

This rule also supports transformations with XSL parameters.

Variable	Description
XMLPATH	(Optional) Specifies a path for multiple XML source files to be processed. The rule appends the transformation output for each source file to the end of the output file.
EXTRFILE	(Optional) Specifies the full path and name of the XML extract file to be transformed. If this variable is present, the rule transforms the extract file and replaces the EXTRFILE input attachment variable with the value of the new output file.
XMLFILE	(Optional) Specifies the full path and name of an XML file to use as the source of the transformation.
XSLTFILE	Specifies the full path and name of the xsl template to use for the transformation.
PRINTPATH	(Optional) Specifies the path where the output file will be written.
OUTFILE	(Optional) Specifies the output file name. If this variable is missing the rule generates a unique file name for the output file.
DOCTYPE	(Optional) Specifies the extension and file type of the output file. The default is .dat.
XSLPARAMETERS	(Optional) An XML rowset which contains the name/value pairs for parameters to use in the xsl transformation.
OUTPUTVAR	(Optional) Defines the name of an additional attachment variable that holds the full path and name of the output file. This is useful when running other rules after this rule that expect an attachment variable with a name other than the default of XSLOUTPUT.

Variable	Description
SOURCE	(Optional) Defines the name of an attachment variable in the output message that contains the full path and name of the XML source to be used for the transformation. This is useful when running other rules prior to this rule which might output the XML source that needs to be transformed to a variable other than the expected variables (XMLFILE or EXTRFILE). In addition, you can use this variable to indicate the source for the transformation should be the output XML message in the result queue instead of an XML file — set SOURCE equal to the value of RESULT in this case.

Here is an example of a request message:

```
<MSGVARS>
    <VAR NAME="doctype">htm</VAR>
    <VAR NAME="REQTYPE">TRANSFORM2</VAR>
    <VAR NAME="SOURCE">RESULT</VAR>
    <VAR NAME="XSLTFILE">X:\\XSL\transform1.xsl</VAR>
    <VAR NAME="XVALUE">2</VAR>
    <ROWSET NAME="XSLPARAMETERS">
       <ROW NUM="1">
          <VAR NAME="y">2</VAR>
          <VAR NAME="x">LOOKUPVAR.XVALUE</VAR>
       </ROW>
    </ROWSET>
</MSGVARS>
```

In addition, each row of parameters for a transformation can contain a value of the format:

```
LOOKUPVAR.ATTACHVARNAME
```

Where ATTACHVARNAME is the name of an attachment variable in the output message. The rule then retrieves the value for the ATTACHVARNAME variable and uses it as the value for the parameter in the transformation. This is useful when you do not know the value of a parameter until run-time.

Here are example request types for the DOCSERV.INI file used in IDS 1.8:

```
[REQTYPE:TRANSFORM]
    function = atcw32->ATCLogTransaction
    function = atcw32->ATCLoadAttachment
    function = atcw32->ATCUnloadAttachment
    function = dsijrule->JavaRunRule,;com/docucorp/ids/rules/
       XsltTransformRule;XSLTTRANSFORMER;transaction;transform;
    function = dprw32->DPRSetConfig
    function = atcw32->ATCSendFile,XSLOUTPUT,XSLOUTPUT,BINARY
    function = irlw32->IRLCopyAttachment
```

Here is a sample JSP page:

```
<%@ page language="java" import="java.util.,
                                 java.net.,
                                 java.io." %>

<jsp:useBean id='dsi' scope='page'
class='com.docucorp.ids.jsp.dsimsg'/>
```

```
<%


    dsi.setTimeOut(30000);
    dsi.debug_on(response);

dsi.addRequest("REQTYPE", "TRANSFORM");
dsi.addRequest("XMLFILE", "X:\\input\\data.xml");
dsi.addRequest("XSLTFILE", "X:\\XSL\\transform1.xsl");
dsi.addRequest("doctype", "htm");

String record = "XSLPARAMETERS";
String rec = dsi.addAttachRec(record);
    if (rec != null){

       dsi.addToAttachRec(rec, "x", "1");
       dsi.addToAttachRec(rec, "y", "1");
    }


dsi.processRequest();

byte buf[] = dsi.receiveFileAsBuffer("XSLOUTPUT");
    if (buf != null){
       out.println(new String(buf));
    }
%>
```

Here is a sample ASP page:

```
<%@ Language=VBScript %>
<%


    Set DSI = server.CreateObject("IDSASP.DSI")

    DSI.clearReq
    DSI.WaitTime = 250                      ' Polling interval
    DSI.Timeout = 1000000
    DSI.ShowAtt = 1

    DSI.AddReq "REQTYPE", "TRANSFORM"
    DSI.AddReq "XMLFILE", "X:\\input\\data.xml"
    DSI.AddReq "XSLTFILE", "X:\\XSL\transform1.xsl"
    DSI.AddReq "doctype", "htm"

    record = DSI.AddAttachRec("XSLPARAMETERS")
    DSI.AddToAttachRec record, "x", "1"
    DSI.AddToAttachRec record, "y", "2"


    'On Error Resume Next

     DSI.ProcessRq
    'If Err <> 0 then
    'Err.Clear
```

```
    'End if


  result = DSI.Result("RESULTS").Value

  Set DSI = Nothing
  Response.Write("result: " & result & "<br>")
  Response.End

%>
```

Customizing Your System

IDS includes several bridges and example applications. These applications typically include windows or dialogs based on HTML and either JSP or ASP. Some dialogs present query result sets for subsequent user selection. The result sets are typically returned as attachment variables, accessible via DSI (or DSICo) API calls. Custom rules and request types can create other query results, and custom HTML dialogs and scripts can implement custom user dialog presentations of the information.

IDS version 1.7 enhanced the internal message format to an XML format based on evolving SOAP standards. You can use this XML format and bypass the DSI API layer, or you can continue to use the DSI APIs.

To help you build alternative dialogs to replace the standard dialogs in the bridges and applications and create new dialogs for other custom applications, the system returns query result sets as structured data in XML format.

The system creates elements inside the <DSIMSG> structure to contain the results of a search as descendants <ROWSET> tag. Each record (ROW) in the result set is stored in a <ROW> XML element, as a child of the <ROWSET> element. Please see the following example.

For many situations, you can use a non-hierarchical (single level) SQL-like ROW to represent hierarchically structured data by *flattening* the columns into a single ROW. If, however, the resulting XML needs the multi-level hierarchy, send the XML as an attachment. For example, if a result set represented a Documaker form set (a list of available forms and images), this could be returned as an Oracle Insurance standard XML file attached to the message.

You can use DSI C APIs, Java, and COM to manipulate the XML result set. Java and VB return the XML from the <ROWSET> element as a string to be loaded into a DOM object by the client. The main reason the row set is returned as XML string or buffer and not as a DOM object is the versioning of DOM objects and XML parsers—IDS does not know what parser and what DOM object will be used by the client application. The C APIs will allow the calling application to get first/next values from a ROWSET, including the name, instance number, data value, and so on.

The existing APIs that access the attachment variables by their old concatenated names still work for backward compatibility, so you do not have to change existing client code unless you want to take advantage of the newer methods.

The results of SSS (server statistics request) are shown in the example below in SOAP-XML format, using both the original and the newer XML message layout for the result set. The number of rows in the LIBRARIES row set is reduced to two for a smaller sample.

Here is an example of the original XML layout:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
envelope">
<SOAP-ENV:Body>
<DSIMSG VERSION="100.018.0">
<CTLBLOCK>
<UNIQUE_ID>4C6482AC643F4B91A9EA347977B8E186</UNIQUE_ID>
<REQTYPE>SSS</REQTYPE>
<USERID>DSICoTB</USERID>
<RESULTQMGR>vaf.atl3nt03</RESULTQMGR>
```

```
</CTLBLOCK>
<MSGVARS>
<VAR NAME="ALLOCCOUNT">5750</VAR>
<VAR NAME="ERRORCOUNT">1</VAR>
<VAR NAME="FREECOUNT">2828</VAR>
<VAR NAME="LASTRESTART">Tue Jul 02 09:49:07 2002</VAR>
<VAR NAME="LIBRARIES">2</VAR>
<VAR NAME="LIBRARIES1.DATE">Jul 1 2002</VAR>
<VAR NAME="LIBRARIES1.NAME">ATC</VAR>
<VAR NAME="LIBRARIES1.TIME">07:40:37</VAR>
<VAR NAME="LIBRARIES1.VERSION">100.018.001</VAR>
<VAR NAME="LIBRARIES2.DATE">Jul 1 2002</VAR>
<VAR NAME="LIBRARIES2.NAME">DCB</VAR>
<VAR NAME="LIBRARIES2.TIME">07:35:16</VAR>
<VAR NAME="LIBRARIES2.VERSION">100.018.001</VAR>
<VAR NAME="RESTARTCOUNT">1</VAR>
<VAR NAME="RESULTS">SUCCESS</VAR>
<VAR NAME="SERVERTIMESPENT">0.015</VAR>
<VAR NAME="SUCCESSCOUNT">1</VAR>
<VAR NAME="UPTIME">Tue Jul 02 09:48:59 2002</VAR>
</MSGVARS>
</DSIMSG>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Here is an example of the newer XML layout:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
envelope">
<SOAP-ENV:Body>
<DSIMSG VERSION="100.018.0">
<CTLBLOCK>
<UNIQUE_ID>4C6482AC643F4B91A9EA347977B8E186</UNIQUE_ID>
<REQTYPE>SSS</REQTYPE>
<USERID>DSICoTB</USERID>
<RESULTQMGR>vaf.atl3nt03</RESULTQMGR>
</CTLBLOCK>
<MSGVARS>
<VAR NAME="ALLOCCOUNT">5750</VAR>
<VAR NAME="ERRORCOUNT">1</VAR>
<VAR NAME="FREECOUNT">2828</VAR>
<VAR NAME="LASTRESTART">Tue Jul 02 09:49:07 2002</VAR>
<ROWSET NAME="LIBRARIES">
<ROW NUM="1">
<VAR NAME="DATE">Jul 1 2002</VAR>
<VAR NAME="NAME">ATC</VAR>
<VAR NAME="TIME">07:40:37</VAR>
<VAR NAME="VERSION">100.018.001</VAR>
</ROW>
<ROW NUM="2">
<VAR NAME="DATE">Jul 1 2002</VAR>
<VAR NAME="NAME">DCB</VAR>
<VAR NAME="TIME">07:35:16</VAR>
<VAR NAME="VERSION">100.018.001</VAR>
</ROW>
</ROWSET>
```

```
<VAR NAME="RESTARTCOUNT">1</VAR>
<VAR NAME="RESULTS">SUCCESS</VAR>
<VAR NAME="SERVERTIMESPENT">0.015</VAR>
<VAR NAME="SUCCESSCOUNT">1</VAR>
<VAR NAME="UPTIME">Tue Jul 02 09:48:59 2002</VAR>
</MSGVARS>
</DSIMSG>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The sample below corresponds to the row set shown above in the XML layout:

```
<?xml version="1.0" encoding="UTF-8"?>
<ROWSET NAME="LIBRARIES">
<ROW NUM="1">
<VAR NAME="DATE">Jul 1 2002</VAR>
<VAR NAME="NAME">ATC</VAR>
<VAR NAME="TIME">07:40:37</VAR>
<VAR NAME="VERSION">100.018.001</VAR>
</ROW>
<ROW NUM="2">
<VAR NAME="DATE">Jul 1 2002</VAR>
<VAR NAME="NAME">DCB</VAR>
<VAR NAME="TIME">07:35:16</VAR>
<VAR NAME="VERSION">100.018.001</VAR>
</ROW>
</ROWSET>
```

You can use these API functions to support row sets:

Function	Description
DSIRowset2XML	Use this rule to get a row set back as XML in memory.
DSIRowset2XMLSize	Use this rule to get a size of row set back as XML in memory.

The GetRowsetXML function was also added to COM and Java APIs (DSICO, IDSASP and DSIJava). You can learn more about these functions in the SDK Reference.

Handling Security Issues

THere are several security issues to consider as you set up the Internet Document Server and build your applications. These include

- Using firewalls
- Implementing security for web applications

Using Firewalls

Typically, you will use the Internet Document Server with some kind of firewall between the web server and the NT server where the Internet Document Server is installed. This lets you give the Internet Document Server access to archives and other sensitive data without permitting the same access to anyone with an Internet connection.

The Internet Document Server (IDS) sample setup included on the installation CD assumes no firewall exists. As you set up the Internet Document Server and your web server with a firewall between the two, keep these points in mind:

- The messaging system can use either WebSphere MQ, JMS, or HTTP. All of these messaging systems communicate with other computers via TCP/IP. The firewall must be configured to open the TCP/IP ports for a messaging system. Read your messaging system's manuals and firewall manuals to find out which ports are used and how to set it up.
- If you are using NT servers, those servers running the Internet Document Server should have TCP/IP networking installed.
- The web server should have an FTP (file transfer protocol) service configured and started.
- File communications from the web server to the Internet Document Server must use FTP.
- For the FTP rules to work, your TCP/IP networking must be installed on the computer where the Internet Document Server is installed.
- Be sure to configure FTP server access with the appropriate user ID and passwords.

The Internet Document Server includes several rules for use with firewalls, such as the IRLFileFTP and IRLInitFTP (Windows) or the FTPRule (Windows and UNIX) rules. For more information on these rules, refer to the SDK Reference.

Implementing Security for Web Applications

Some ASP customers use the ASP IDS and a web server for *hot-key* applications. The system builds an URL with, for example, the account number and bill date. When this URL is executed, it returns a PDF or HTML bill presented in a browser window.

A potential security problem is if the user changes the account number on the URL and retrieve someone else's bill or document. The system, however, can encrypt parts of the URL to make it more difficult to see someone else's documents.

The COM object DSIEncr lets VB or an ASP page encrypt a value. The ASP syntax is as follows:

```
<%@ Language=VBScript %>
```

```
<%
        Response.Buffer=True
        Set DSI = Server.CreateObject("DSI.DSIEncrypt")

        val = "abc"
        DSI.Encrypt val

        Response.Write "Encrypted: abc as " + val + "<BR>"

        DSI.Decrypt val

        Response.Write "Decrypted as " + val
        Response.End
        Set DSI = nothing
%>
```

The COM object is created by Server.CreateObject() method.

Two methods are available in this COM object, Decrypt and Encrypt. The Decrypt method is provided for testing purposes.

A simple implementation includes an ASP page which encrypts the account number on the URL before redirecting the user to the presentment web site.

Here is a sample URL without the encryption:

```
http://webaddress/present.asp?ACCT_NO=1032714&BILLDATE=20020415
```

Here is a sample URL with encryption:

```
http://webaddress/
present.asp?ACCT_NO=0zJxWr96vm1ZkniK7Cp0n&BILLDATE=20020415
```

The result of the encryption is a safe string for the URL so no additional encoding is required. There will be no special characters.

On the IDS side, you can use the DPRDecryptValue rule to decrypt the value before executing a search in the database. Here is an example of how you use this rule:

```
function = dprw32->DPRDecryptValue,ACCT_NO
```

After this rule is executed on the RUNF message, the ACCT_NO in the attachment is replaced by the real value.

NOTE: The encryption algorithm is proprietary.

USING THE FAP2XML UTILITY

You can use the FAP2XML utility to convert a FAP file into an XML file.

NOTE: This utility is intended for iPPS manuscripting support and is not used by the Documaker system. Neither IDS, Image Editor, or Documaker Studio can read the resulting file and turn it into a FAP or NA file.

Program names

Windows FAP2XMLW.EXE

Syntax

```
FAP2XMLW  /i=FAPFile /o=XMLFile /x=FXRFile /ini=INIFile
```

Parameter	Description
/I	(Optional) Enter the name of the FAP file.
/O	(Optional) Enter the name you want assigned to the XML file. The default is the name of the FAP file with an XML extension.
/X	Enter the name of the font cross-reference file (FXR) file.
/INI	(Optional) Enter the name of the INI file which contains settings for this utility. The utility looks in the MasterResource control group find the location of FAP and other files.

Using LDAP Support

IDS supports the use of Lightweight Directory Access Protocol (LDAP), an application protocol for querying and modifying directory services running over TCP/IP.

IDS includes an LDAP API for Java. The JAVA DocucorpUtil package includes an LDAP class which you can use to query an LDAP server for group information for a user.

For more information please see the LDAP.html documentation that ships with IDS located in the dsi_sdk\java\docs\com\docucorp\util directory and see the ldapTest class example which ships with IDS and is located in the dsi_sdk\java\samples\ldap directory.

NOTE: If you are using JVM version 1.3, you must replace the jsse.jar file with the one from JVM version 1.4, which you can find at this location:

JAVA_HOME\jre\lib\ext

IDS also includes an LDAP API for C that you can use to query an LDAP server for group information for a user. These functions are supported in the API:

- LDAPInit
- LDAPTerm
- LDAPSearchDirectory
- LDAPGetErrorCode
- LDAPGetErrorMessage

For more information, see the SDK Reference.

Searching a Directory Information Tree

You can search a Directory Information Tree (DIT) in an LDAP server. IDS includes the following rules for conducting LDAP queries to determine a user ID group or role membership:

- DPRSearchLDAP (C)
- search (Java)

These rules will look for all configuration options in rule arguments, a properties file, INI options, and input attachment variables, in that order. Option values found in more than one source override the previous value.

- For information on the DPRSearchLDAP rule, see Using the Documaker Bridge.
- For information on the Java rule search, refer to the dsidocs/com/Docucorp/DSI/util/DSIJession.html documentation shipped with the Java SDK.

Using Default Time-outs for DSILIB-Based Client Applications

You can set default time-outs for DSILIB-based client applications. You set these defaults using these configuration entries in the docclient.xml file:

- DefaultTimeoutSeconds
- MaxTimeoutSeconds
- MinTimeoutSeconds

NOTE: Examples of client-based applications that benefit from this feature are ASP pages using IDSASP.DLL, JSP pages using IDSJSP.jar, and the test programs DSICOTB.EXE, DSITEST.EXE, and DSIEX.EXE.

For instance, suppose you have hundreds of web applications installed on a single IIS or Java server and all of these applications are talking to the same IDS setup. Suppose some of these web applications have large time-out values which are not suitable for production mode, such as values longer than a few minutes. In this scenario, a transaction that takes a long time can tie up one thread on the web server. Since the total number of threads in the web server is limited, this can affect other applications.

Using these options, the system administrator can make sure that no matter what was specified as the time-out value, the actual time-out period is what the system administrator thinks it should be.

These entries go under the DocumentClient section in the docclient.xml file. Here is an example:

```
<section name="DocumentClient">
    <entry name="DefaultTimeoutSeconds">45</entry>
    <entry name="MaxTimeoutSeconds">60</entry>
    <entry name="MinTimeoutSeconds">30</entry>
```

Entry	Description
DefaultTimeoutSeconds	Use this entry when DSILIB-based client applications, such as dsiex, dsitest, and dsicotb test programs, provide a time-out value of zero (0) to DSIGetQueueRec calls to wait for a response message. The default is 15 seconds.
MaxTimeoutSeconds	Use this entry to set the upper limit for the time-out value when waiting for a response message. If a time-out value is specified for DSIGetQueueRec calls and it is greater than MaxTimeoutSecondsvalue, the MaxTimeoutSeconds is used instead. There is no default.
MinTimeoutSeconds	Use this entry to set the lower limit for the time-out value when waiting for a response message. If a time-out value is specified for DSIGetQueueRec calls and it is less than MinTimeoutSeconds value, the MinTimeoutSeconds is used instead. There is no default.

NOTE: It is possible that Microsoft Server script execution time-out limits could be set lower than the values specified for this feature. In those instances, the values of the Microsoft Server script execution time-out limits would be used. Please consult your Microsoft documentation for more information.

Running Timed Requests

You can run timed requests repeatedly or just in the primary instance. Use the following entry attributes for a timed request entry under the Timers section in docserv.xml file:

Entry attribute	Description

RepeatInterval	Enter true or yes (case sensitive) to tell IDS to convert the text value provided for the entry into seconds and run the timed entry at each interval specified. Here are some few examples.
	This timed section runs every 120 seconds:
	`<entry RepeatInterval="yes" name="SSS">00:01:60</entry>`
	This timed section runs every 3720 seconds:
	`<entry RepeatInterval="yes" name="SSS">01:01:60</entry>`
	This timed section runs every 90 seconds:
	`<entry RepeatInterval="yes" name="SSS">90</entry>`
	If more than one IDS instance is running, any timed sections configured with the RepeatInterval attribute are run at a random interval using the interval seconds as the seed. Making sure they are not run at the same time by all IDS instances, allows other processing to take place.
	The default lower bound is 60 seconds, meaning any timed section that is configured to use a time interval of less than 60 seconds will instead use the default value.

RunOnPrimaryInstanceOnly	Enter yes or true (case sensitive) to make sure only the primary instance runs the timed section. For instance, you might want to do this when a timed section runs a synchronization rule that updates resources for a master resource library. This type of request would only need to be run once. If you omit this attribute, all IDS instances will run each timed section. Here is an example: `<entry RunOnPrimaryInstanceOnly="Yes" name="SSS">00:01:60</entry>`

In-Process Rendering for DPAView

With version 11.3 Shared Objects, you can create a bitmap representations of a DPA document without creating another instance of IDS. Before version 11.3, you had to have an additional dedicated instance of IDS to create the bitmaps.

DRLLIB detects whether it is running inside an instance of IDS or inside an external process. If DRLLIB detects that it is running inside IDS, it will use its own instance of IDS to render the bitmap.

DPAView lets you create bitmaps from archived transactions in Documanage for display in Documanage Workstation. To do this, you have to have the following items set up:

- Documaker Server (publishing engine)
- An MRL set up to archive into Documanage (DBHandler:DMIA)
- Documanage version 6.5 and higher
- Documanage Bridge version 3.3

The DPA archives created by Documaker Server through the GenArc program must have been archived into Documanage.

You can enable additional tracing by setting the environment variable DRLDEBUG.

This feature also adds the DRLGetConfig API.

DRLGetConfig

Use this API to retrieve the CONFIG name for the DPA file processed by DRLProcessDPAFile. You must call this API after running DRLProcessDPAFile.

These APIs are not supported in-process.

- DRLProcessPageDC
- DRLProcessPageBuffer

Syntax

```
DRLGetConfig (hInstance) (config) (len)
```

Parameters

Parameter	Description
hInstance	The instance handle returned by DRLInitInstance call.
config	The parameter to hold the config.
len	The maximum size the config parameter may hold.

Returns DRLERR_* value

See also DRLProcessDPAFile

Using DAL Functions for WIP Column Access

You can use the following DAL functions to set or retrieve WIP field data when Docupresentment processes WIP or archived transactions:

Function	Enhancement
WIPKEY1	Returns the value of the Key1 WIP field. Requires no input parameters. Here is an example: `Val=WIPKEY1()`
WIPKEY2	Returns the value of the Key2 WIP field. Requires no input parameters. Here is an example: `Val=WIPKEY2()`
WIPKEYID	Returns the value of the KeyID WIP field. Requires no input parameters. Here is an example: `Val=WIPKEYID()`
WIPFLD	Returns the value of the specified WIP field data. This field must be defined in the WIP.DFD file. Requires one input parameter to indicate which key value to return. Here is an example: `Val=WIPFLD("TranCode")`

SETWIPFLD	Sets the value of the specified WIP field key and keeps it in memory until the job finishes. For example, this DAL script sets/changes CURRUSER to DEMO1 and returns it:
	`SETWIPFLD("CURRUSER","DEMO1");`
	`Val=WIPFLD("CURRUSER");`
	`Return Val;`

There are several ways to run the DAL script, here are two examples:

- Using the ~DALRUN built-in INI function following a DAL script file, as shown in this example:

```
~DALRUN wipkey.dal
```

- Using the DPRExecuteDAL rule, as shown in the following request type:

```
[ ReqType:i_WipTest]
    function = atcw32->ATCLogTransaction
    function = atcw32->ATCLoadAttachment
    function = dprw32->DPRSetConfig
    function = atcw32->ATCUnloadAttachment
    function = dprw32->DPRGetWipFormset
    function = dprW32->DPRExecuteDAL,wipkey.dal,RUNF
```

You can also use the following built-in INI functions to retrieve WIP field data when Docupresentment processes WIP or archived transactions:

- ~KEY1
- ~KEY2
- ~KEYID
- ~ORIGUSER

- ~CREATETIME
- ~MODIFYTIME
- ~FORMSETID
- ~ORIGFSID
- ~TRANCODE
- ~DESC
- ~WIPFIELD

All of these built-in functions except WIPFIELD do not require input parameters.

The WIPFLD function requires an input parameter that indicates the field data to return. Here is an example:

```
~WIPFLD FORMSETID
```

In this example, FORMSETID is the input parameter that specifies the field name.

Using Enterprise Web Processing Services

Enterprise Web Processing Services (EWPS) make it easier to integrate applications, providing a set of web services for accessing the Documaker forms library and initiating real-time publishing operations. This helps you deliver the information requested by your clients, prospects, employees, and business partners.

Via EWPS you can use a number of essential mechanisms, such as WS-I SOAP interfaces for application integration, JSON for UI integration, or prepackaged business parts for design-time integration, to create a solution that uniquely meets your business needs.

Typical EWPS-enabled solutions include:

- Self-service publishing solutions
- Document search utilities
- Composition and workflow systems
- Systems that embed publishing artifacts into web pages
- Applications that help users create documents and forms

EWPS supports these protocols:

- SOAP (Simple Object Access Protocol).
- JSON (JavaScript Object Notation)

For more information on EWPS, see Introduction to Enterprise Web Processing Services.

NOTE: EWPS is not available on the UNIX platforms.

Chapter 3

Creating Output Files

This chapter discusses the types of output you can create, such as PDF or HTML files. If you are unsure as to which type of file you want to produce, see HTML vs. PDF on page 4.

For more information on creating these files, see these topics:

- Creating PDF Files on page 208
- Creating HTML Files on page 230
- Creating XML Output on page 236

In addition, this chapter includes information about the various paper sizes the system supports. For more information, see Creating Output Files on page 207.

Creating PDF Files

The PDF Print Driver creates Adobe Portable Document Format (PDF) files from output from the Rules Processor's GenPrint program.

PDF is a document language developed by Adobe Systems, Inc., that allows formatting and graphics information to be stored along with text. Using PDF files, you can make sure form sets viewed and printed match originals created by the GenPrint program.

A document stored in PDF format can be transmitted to and viewed on many types of systems. There are PDF viewer applications available for many platforms, both as stand-alone programs and as add-ons for existing applications (such as word processors and Internet web browsers). You can download Acrobat Reader from Adobe Systems' web site (www.adobe.com).

Print output directed to the PDF Print Driver is stored in one or more files. You can then view these files using the Acrobat Reader. This topic discusses...

- Setting Up the PDF Print Driver on page 209
- Creating PDF Files with Unicode Support on page 212
- Setting PDF Compression Options on page 212
- Handling Fonts on page 215
- Using the PDF Print Driver with GenPrint on page 220
- Font Cross Reference File Tips on page 223
- Using the 14 Base Fonts Distributed with Acrobat Reader on page 226
- Setting Up Bookmarks on page 227
- Limitations on page 229

Setting Up the PDF Print Driver

You must have installed the Rules Processor, including the GenPrint program, to use the PDF Print Driver. Adding the ability to output PDF files from the GenPrint program requires two steps:

1 First, copy the PDF DSO file from the CD into your DAP DSO directory. The PDF DSO is contained in the \DAP_PDF\Relxx directory on the CD. For windows, this file is named PDFW32.DLL. For UNIX it is named libpdf.so.

NOTE: The Relxx directory indicates the system release used for running the GenPrint program. Version 9.5 (Rel95) is the first release of the GenPrint program supported by the PDF Print Driver.

2 Second, make the following changes in your FSISYS.INI file. Start by adding a new PrtType control group named *PrtType:PDF* as follows:

```
< PrtType:PDF >
    Device          = TEST.PDF
    Bookmark        = Yes,Page
    DownLoadFonts   = No,Enabled
    Module          = PDFW32
    PageNumbers     = Yes
    PrintFunc       = PDFPrint
    SendOverlays    = No,Enabled
    SendColor       =
    PrintViewOnly   = No
    SplitText       = No
    SplitPercent    = 50
    Class           = PDF
    DisplayMode     = UseOutlines
    PaperSize       = 0
    ForceColorBitmaps =
    FontCompression= 2
```

Option	Description
Device	Enter the path and file name for the output.
Bookmark	The format is `Bookmark = Yes,Page,No,Group` The first parameter specifies whether bookmarks are created. The second parameter specifies the lowest level at which bookmarks are created (Formset, Group, Form, or Page). The third parameter specifies whether bookmarks for unnamed objects are included. The fourth parameter specifies the lowest level of bookmark to be displayed (Formset, Group, Form, or Page). If you omit this parameter, the system uses the level specified for the second parameter.
DownLoadFonts	Set to *No, Enabled*

Option	Description
Module	The name of the program module which contains the system's PDF print driver. See also the Class option.
PageNumbers	Set to *No* if you do not want page numbers. Defaults to *Yes*.
PrintFunc	The name of the program function that is the main entry point into the system's PDF print driver.
SendOverlays	Set to *No, Enabled*
SendColor	This option does not apply to PDF files. If the document contains color, the PDF Print Driver correctly processes that information so Acrobat Reader will display the color appropriately.
PrintViewOnly	Use this option to output view only forms when you create PDF files. Set to *Yes* to print these images. Defaults to *No*. Images marked as Entry only will not be printed, as these usually are the worksheet type images used for data collection.
SplitText	Use the SplitText and SplitPercent options if you see text that is not positioned accurately when the PDF file is viewed. If the SplitText option is set to *Yes*, every text string will be split and adjusted position according to the value of the SplitPercent option. See Handling Fonts on page 215 for more information.
SplitPercent	This value can be -1 or any integer from zero (0) to 100. -1 means every text string will be split on a space. The integer is used to calculate the threshold for split and adjustment.
Class	Specifies the printer classification, such as AFP, PCL, XER, PST, or GDI. If you omit this option, the system defaults to the first three letters from the Module option. Some internal functions expect a certain type of printer. For instance, all 2-up functions require an AFP printer. The internal functions check the Class option to make sure the correct printer is available before continuing.
DisplayMode	This option lets you control how the PDF file is initially displayed. To have the PDF file open… - with bookmarks (document outline) visible, enter *UseOutlines* - with thumbnail images visible, enter *UseThumbs* - in full-screen mode, with no menu bar or other window controls visible, enter *FullScreen* - in default mode, with neither bookmarks or thumbnails visible, enter *UseNone* Keep in mind that the PDF Print Driver includes your choice in the PDF file it creates. Acrobat and Adobe Reader honor this setting and display the PDF using your DisplayMode setting. The Acrobat OCX control which lets browsers display PDF files may not honor the DisplayMode setting. Other software, like GhostView or Xpdf, that displays PDF files may not honor the DisplayMode setting.

Option	Description
PaperSize	This option selects the paper size. The most commonly chosen options are shown here. For a complete listing of all the options you can choose from, see Creating Output Files on page 207. 0=Letter (default) 1=Legal 2=A4 3=Executive 98=Custom 99=Current type When deciding the size, the system first sets the page size to the size of the first image on the page. If the page size is *Custom*, the page size will be set to the form size. If the page size is now *Letter* (the default), the PDF Print Driver checks the PaperSize option. If the PaperSize option is specified, the system uses it to determine the size. If the PaperSize option is set to *Custom* and page size is less than *Letter*, the page size is set to *Letter*. Otherwise, the system uses the custom width and height.
ForceColorBitmaps	Enter Yes if you want the PDF Driver to print images in color even if the images are not set to print in color (the image does have to be a color image, of course). The default is No.
FontCompression	Use this option to compress embedded fonts. Enter zero (0), 1, 2, or 3 to indicate the level of compression. Zero indicates no compression and three indicates the highest level of compression. The default is two (2). Keep in mind that this option only compresses the ASCII portion of PostScript fonts, so the compression ratio is between 5-15%. For True Type fonts, the compression ratio is between 40-50%.

3 If you have a Printers control group, simply add this option to that control group:

```
PrtType = PDF
```

If you do not have a Printers control group, add this control group and option. For example, your Printers control group might look like this:

```
< Printers >
    PrtType = PDF
```

The GenPrint program can now use the PDF Print Driver.

Creating PDF Files with Unicode Support

With IDS version 1.8, the PDF Print Driver now supports Unicode data. Previously, PDF files produced by the PDF Print Driver could be viewed in Acrobat version 3.0 and above. For Unicode support, you must now use Acrobat version 4.0 and above. Oracle Insurance recommends you use version 7.0 and above.

Having Unicode support lets you create PDF documents that contain Far Eastern languages such as Chinese, Japanese, or Thai.

To create a PDF file containing Unicode data, you must specify a TrueType font to be embedded (downloaded) into the PDF file. You cannot use the internal Acrobat fonts or embed PostScript fonts when printing Unicode data.

NOTE: For information on embedding a TrueType font into a PDF file, see Embedding Fonts on page 217.

Setting PDF Compression Options

You can choose from these PDF compression methods:

Choose	For
0 (zero)	no compression
1	best speed
2	default compression
3	best compression

To override the default, add the Compression option in the PrtType:PDF control group in the DAP.INI file.

```
< PrtType:PDF >
    Compression    = 3
```

You can test the various compression options to see what works best for your implementation by comparing...

- The time difference between the request to view the transaction in Acrobat Reader and when it is displayed.
- The size of the PDF file after it is retrieved.

You can also control how much compression is used for fonts embedded into your PDF files. To do this, see the FontCompression option, discussed on page 211.

PRODUCING OPTIMAL PDF OUTPUT

To produce optimal PDF output, make the following changes in your font cross-reference (FXR) file.

- Remove font IDs built from the FORMSX font.

  This is not needed in Metacode conversions and there is no equivalent built-in Acrobat font. Furthermore, the original entry does not contain character width information.

- Fix point size settings for font IDs.

  Metacode fonts do not contain point size information. Therefore, a point size is approximated when a Metacode font is inserted into the FXR file. Sometimes, this approximation is incorrect. Many Metacode fonts include the point size as a part of its file name. For example, font ID 5 was built from the Metacode font AR07BP but is listed as having a point size of 8. However, AR07BP is really a 7-point font and the point size for this font ID should be changed to 7.00.

- Remove font IDs built from landscape fonts.

  For landscape fonts, where the equivalent portrait fonts are included in the FXR, the font IDs for the landscape fonts should be deleted and the landscape font name should be in the Rotated Fonts field of the portrait font. For example, font ID 4 was built from the landscape font AR07BL (Arial Bold 7-point). Font ID 5 was built from the portrait font AR07BP (Arial Bold 7-point). Therefore, font ID 4 was deleted and *;;AR07BL* was added to the Rotated Font Files field in the Metacode section of the Printers tab of the FXR file.

- Remove non-text fonts from the FXR file.

  You may decide to leave a non-text font in the FXR if you have an equivalent TrueType or PostScript font to embed and you have enabled font downloading. If not, remove the font IDs for non-text fonts.

  If the non-text font contains a signature or graphic, such as a company logo, convert the font to a logo (LOG) file. Fonts of this style have contiguous characters and are always referenced one way. For example, the font JOHNDO might contain only the letters *A*, *B*, and *C*. When the letters *ABC* are printed together using the JOHNDO font, the signature *John Doe* is printed. When a font is always used with a single contiguous set of characters, convert the font to a logo.

  For non-text fonts that contain characters which will be printed using a variety of combinations, you cannot use a logo. In this case, make the Xerox font available in the directory specified by the FontLib option in the MasterResource control group. Examples of these types of non-text fonts include OCR, MICR, and barcode fonts.

  For example, a ZIP code (barcode) font produces a different picture when different ZIP codes are used. The ZIP code for 30309 looks different than the ZIP code for 49501. Using this approach, a bitmap image is created from the Xerox font using the specified characters (30309, 49501, and so on.) in the print stream. Only use this approach for fonts that are used infrequently because it results in larger PDF files which affects the time it takes to create and download a PDF file.

- Make sure color bitmaps are not saved as Comp TIFF or Comp Pack. These compression methods are not supported by PDF.

  The Comp TIFF format is designed to compress monochrome bitmaps, not color ones. The Comp Pack format is only useful when the color bitmap is 16 or 256 colors. There is no reason to use Comp Pack on a full-color (24-bit) bitmap.

  In most cases, you can simply leave full color bitmaps as JPEG or bitmap files and not convert them at all. The only reason to convert these files to the LOG format is to move them to a platform that does not support those file types, like MVS.

- Run the FXRVALID utility to check the FXR file.

  The FXRVALID utility reports missing font files, incorrect built-in Acrobat font names, and so on. Correct any errors reported by the FXRVALID utility.

- Avoid specialty fonts when mapping to built-in Acrobat fonts.

  The built-in Acrobat fonts include Courier, Helvetica, and Times plus a couple of fonts containing non-text characters (Symbol and ZapfDingbats). Fonts such as Arial and Univers are pretty similar to Helvetica in terms of appearance and size and the built-in Acrobat font can often be used without any noticeable effect. Some font vendors also supply versions of the fonts where the characters are condensed (narrow) or expanded (wide). Although these fonts may have a similar character appearance, their size has been altered in a way that makes mapping to a built-in Acrobat fonts problematic.

- Use SplitText option when using built-in Acrobat fonts.

  When using built-in Acrobat fonts, many printer fonts are mapped to a single built-in Acrobat font. Many times, the printer fonts are not scaled perfectly in terms of their character widths. For example, the letter *A* in a 5-point printer font may have a width of 8 dots. Meanwhile, the letter *A* for the same font at 10-point (twice the size) may have a width of 14 dots (instead of 8*2 or 16 dots). This becomes a problem when mapping these non-scalable printer fonts to the built-in Acrobat fonts. The built-in Acrobat font is scalable and 10-point characters are twice the size as 5-point characters. You can use the SplitText and SplitPercent options to hide the differences that result from mapping non-scalable printer fonts into built-in (scalable) Acrobat fonts.

  If you set the SplitText option to Yes, every text string will be split and adjusted position according to the value of the SplitPercent option. The default is No.

  You can set the SplitPercent option to -1 or any integer between zero (0) and 100. When you set this option to -1, every text string will be split and adjusted one position on a space. When you set this option to a positive value, such as 45, the differences between the character widths of the font used for a text string and character widths of the base font are accumulated.

  Once the accumulated differences greater than 45/100*(width of the space character), the text string is split and a new accumulation is started. Therefore, smaller values for the SplitPercent option will produce better visual results at the cost of slightly slower performance and somewhat larger PDF files.

According to tests performed on a 55-page and a 100-page document, the size of the one with the SplitPercent option equal to -1 is about 12 percent greater than the one without any text split. The performance difference is hardly noticeable.

HANDLING FONTS

The PDF Print Driver lets you embed fonts into the PDF print stream. This topic discusses when to embed fonts and when not to. It also describes how the system determines which base font to use or which custom font to embed.

Generally, you want the PDF Print Driver to reproduce your document so that it looks exactly as it did when you created it. Embedding fonts lets you accomplish this if you are using fonts that do not match the criteria discussed below.

When not to embed fonts

If you do not need to reproduce the exact look of the original document, you do not need to embed fonts. If you use the base AGFA fonts and the FXR file Oracle Insurance distributes, you do not need to embed fonts. The AGFA fonts are scalable and the AGFA Courier, Times, and Univers fonts are mapped to the names of the 14 base fonts Adobe distributes with Acrobat Reader.

NOTE: For a list of the base fonts, see Using the 14 Base Fonts Distributed with Acrobat Reader on page 226.

The AGFA Letter Gothic (fixed pitch, sans-serif) font is mapped to the base Adobe Courier (fixed pitch, serif) font. If you prefer the sans-serif look of Letter Gothic, you should embed that font.

Adobe also uses a standard scaling algorithm. If your implementation uses fonts that scale exactly as Adobe expects, you do not need to embed fonts. The PDF Print Driver determines the fonts to use and scales them for you. See Not Embedding Fonts on page 216 for more information.

In summary, you do not need to embed fonts if the fonts you are using …

- Are already scalable
- Closely match the PDF base fonts

When to embed fonts

Embedding fonts lets you control the appearance of the document by letting you specify which fonts Acrobat Reader should use and what the font width will be. When you need to reproduce the exact look of the original document and you use custom fonts that do not scale exactly as Adobe expects, you should embed fonts.

To embed fonts, you need a set of PostScript Type 1 fonts or TrueType fonts, and you need to set the DownLoadFonts INI option to *Yes*. You also need to run the FXRVALID utility to prepare your FXR file. For detailed instructions, see Embedding Fonts on page 217.

Not Embedding Fonts

If you are not going to embed fonts, you must set the following INI option to *No*, as shown here:

```
DownloadFonts = No
```

When you use a font that is not included in the 14 base fonts distributed with Acrobat Reader, the PDF Print Driver uses the information in the following fields, in this order, to determine what to do with the font:

- Setup Data field on the Other tab of the Font Properties window in the Font Manager. The system checks this field to see if its contents matches one of the 14 base Adobe fonts or an equivalent AGFA font name.
- Font Name field under PostScript on the Printer tab of the Font Properties window in the Font Manager. The system checks this field to see if its contents matches one of the 14 base Adobe fonts or an equivalent AGFA font name.

TypeFace field on the Description tab of the Font Properties window in the Font Manager. The system checks this field to see if its contents matches one of the 14 base Adobe fonts or an equivalent AGFA font name.

NOTE: For a list of the base fonts, see Using the 14 Base Fonts Distributed with Acrobat Reader on page 226.

When the system matches a criteria, it then stops. If, after checking these fields, the system does not find information that matches one of the 14 base Adobe fonts or an equivalent AGFA font, it then maps…

- Proportional fonts to the Adobe Helvetica font (normal, bold, italic, or bolditalic)
- Fixed pitch or non-proportional fonts to the Adobe Courier font (normal, bold, italic, or bolditalic)

The system then checks these fields to determine additional font attributes:

- Spacing (fixed pitch or proportional)
- Style (italic or upright)
- Stroke Weight (bold or normal)

Embedding Fonts

If you are going to embed fonts, you must have either PostScript Type 1 or TrueType fonts. In addition, you must set the following INI option to *Yes*, as shown here:

```
DownloadFonts = Yes
```

You must also have the following information set up correctly for the PDF Print Driver to embed the font. If there is an error in any of the following fields, the PDF Print Driver will substitute one of the 14 base fonts using the criteria discussed in the topic, Not Embedding Fonts on page 216.

- Options field on the Other tab of the Font Properties window is set to one (1) if the field should be embedded or zero (0) if it should not be embedded.
- Font Index field on the Other tab of the Font Properties window specifies the width table to use for a group. Properly scaled font IDs have the same grouping value. This value is the font ID of one of the fonts in the group.
- Font File Name field on the Other tab and/or Postscript Font File Name field on the Printer tab. (Postscript => .PFB TrueType => .TTF). This field contains the file name of the PostScript or TrueType font you want to embed. This file should exist in the directory specified by the FontLib setting in your master resource library.

NOTE: The information stored in the A,R3 OTH record in the FXR appears in the fields on the Other tab of the Font Properties window in the Font Manager. You can edit the information there. The FXRVALID utility can also create this record. For more information about the FXRVALID utility, see the Docutoolbox Reference.

When you use internal Acrobat fonts in producing your PDF file, the PDF Print Driver maps your printer fonts from the FXR file to internal Acrobat fonts. Because printer fonts in FXR file differ from the internal Acrobat fonts, the PDF Print Driver adjusts the point size of the font to produce each piece of text so that the total character width of the text in the PDF file matches the character width of the text as it would have printed using the printer font. This may cause the height of text to vary slightly. This is done to maintain the proper character width of the text.

The following example may help to understand the issue. The two sentences below represent the text from a FAP file and that same text, rendered using an internal Acrobat font.

This is Arial 24 point.

This is Arial 24 point.

The first line uses an Arial 24 point font. The second line uses a 21 point Verdana font. If your system has both screen fonts installed, the periods at the end of each sentence will appear line up together (or nearly so).

While the two fonts are similar in design, you can see that the heights and widths of the characters are different. For this particular sentence, a 21 point Verdana font will approximate the width of the same text when using an Arial 24 point font.

Similarly, Oracle Insurance's PDF Print Driver determines the point size of the internal Acrobat font to use such that the total width of the text string is identical to the total width of the text string in the original form (FAP file).

If higher fidelity is required, you can embed a PostScript or TrueType font that has the exact metrics of the original printer font used in the FXR file. This creates a larger PDF file and a higher fidelity document.

Consider the alternatives and decide which approach best meets your needs — using internal Acrobat fonts (less fidelity, smaller file size) vs. embedded fonts (high fidelity, larger file size).

Handling Fonts with Multiple Width Tables

For each font family, only one font with one width table (called base font) is created in PDF. So if a text string is using a font with a different width table from base font, the length of this string may be greater or smaller than anticipated and it may overlap with other text strings.

To solve this problem, you need to split text strings and adjust positions, so that overlapping between text strings can be avoided. To help you do this, the PDF Print Driver lets you use the following INI options:

```
< PrtType:PDF >
    SplitText      = Yes
    SplitPercent   =
```

If you set the SplitText option to *Yes*, every text string will be split and adjusted position according to the value of the SplitPercent option. The default is *No*.

The value of the SplitPercent option can be -1 or any integer between zero (0) and 100. When you set this option to *-1*, every text string will be split and adjusted one position on a space. When you set this option to a positive value, such as *45*, the differences between the character widths of the font used for a text string and character widths of the base font are accumulated.

Once the accumulated differences greater than 45/100*(width of the space character), the text string is split and a new accumulation is started. Therefore, smaller value for the SplitPercent option produce better results but also slow performance and result in larger files.

According to tests performed on a 55-page and a 100-page document, the size of the one with the SplitPercent option equal to -1 is about 12 percent greater than the one without any text split. The performance difference is hardly noticeable.

NOTE: You can also use the SplitText and SplitPercent INI options in the AFP print driver to eliminate differences in text positioning on 240 dpi and 300 dpi AFP printers.

USING THE PDF PRINT DRIVER WITH GENPRINT

The GenPrint program creates the print stream for each recipient batch and sends it to a printer output file. A *batch active* flag tells PRTLIB's installed output function to keep the current file open and to append each output group to the active stream, only closing the file at the end of the batch.

In most GenPrint processing situations, this is how you want it to work. For PDF, however, you need to break each recipient record into a separate file. The following topics describe how to send each form set to a separate file.

Changing the GenPrint Program

With changes made for custom callback support in version 9.5 (described in the Version 9.5 Features and Information document), a new callback function is included in the GenPrint program. This callback function is called MultiFilePrint().

NOTE: The callback function applies when you are running the GenPrint program in multi-step mode.

To print multiple files when you are using Documaker in single step mode, use the PrintFormset rule. For more information on this rule, see the Rules Reference.

To use this callback function, you must change the FSISYS.INI file as shown below:

```
< Print >
   CallbackFunc    = MultiFilePrint
   MultiFileLog    = { full path file name of a log file (optional) }

< RunMode >
   DownloadFAP     = Yes
   LoadFAPBitmap   = Yes
   CheckNextRecip  = No
```

Here is an example setup:

```
< Printer >
   PrtType         = PDF
< PrtType:PDF >
   Device          = e:\test.pdf
   DownLoadFonts   = No,Disabled
   LanguageLevel   = Level2
   Module          = PDFW32
   PageNumbers     = Yes
   PrintFunc       = PDFPrint
   SendOverlays    = No,Disabled
   SendColor       = Yes,Enabled
< Printer1 >
   PORT            = ..\RPEX1\DATA\BAT10001.PDF
< Printer2 >
   PORT            = ..\RPEX1\DATA\BAT20001.PDF
... (and so on)
```

Setting the CheckNextRecip option

When you use the PDF Print Driver, make sure the CheckNextRecip INI option is set to No (the default is No.) The GenPrint program uses this option to look ahead to subsequent recipient records and queue up recipient records that match the same form set.

This improves system performance when many recipient batch records are placed in the same print batch and sorted together. However, it is essential that each recipient record be viewed as a separate print transaction for this to work. Without this option disabled, each file will contain multiple recipients for the same form set, which is probably not what you want.

Using overlays

You cannot use overlays with the PDF Print Driver. There is no way to generate PDF overlays or use them at print time. Because of this, the GenPrint program ignores the SendOverlays option when it prints to the PDF Print Driver. FAP files and bitmaps must be loaded, which is indicated by setting these INI options:

```
DownloadFAP    = Yes
LoadFAPBitmap  = Yes
```

Using the MultiFilePrint Callback function

If you specify the MultiFileLog option in the Print control group, the specified file is created at the start of the GenPrint program when the callback is installed. The file is closed at the end of the GenPrint program when the callback is uninstalled.

At the end of each transaction, a new output file name is constructed and the GenPrint program's normal behavior of only outputting to one file is overridden. MultiFilePrint makes the following assumption about the output file name:

```
XXXX####
XXXX = four characters that are preserved.
#### = four characters set to a zero-filled sequence number.
```

MultiFilePrint assumes that the original print batch name ends in 0001. The second file opened will be 0002, and so on, up to 9999. MultiFilePrint assumes that no single recipient batch contains more than 9999 recipient batch records. If this is the case, a custom version of MultiFilePrint is required.

Avoid this approach, however, since this is a large number of output files to attempt to track and manage. MultiFilePrint does work with multiple print batches, and each batch can contain up to 9999 recipient records.

If you turned on logging, as each file is completed the system creates a log record in the log file you specified.

NOTE: The MultiFilePrint option should only be used with the PDF, RTF, HTML, and XML print drivers.

Using the log file

The log record has the following format:

```
;FIELD1;FIELD2;FIELD3;FIELD4;FIELD5;FIELD6;FIELD7;FIELD8;FIELD9;

FIELD1= Logical recipient batch file name
FIELD2 =Physical (full file name) recipient batch file
FIELD3= Group name 1 (e.g., Company)
FIELD4= Group name 2 (e.g., L.O.B.)
```

```
FIELD5= Group name 3 (usually empty)
FIELD6= TransactionId (e.g., Policy no)
FIELD7= Transaction type
FIELD8= Recipient type (as specified in POLFILE or FORM.DAT)
FIELD9= Print output file (full file name)
```

The log file is provided for use by a custom application and implementation to handle the management and distribution of the many individual output files.

Generating Separate Files

You can generate separate files for each transaction when you choose PDF (or RTF) from WIP or batch print.

The name of the files will have a rolling number appended to the end of the name that starts the process and is filled in on the Print window. This is automatically handled and you do not have to set INI options to get the WIP or batch print to work as long as your PrtType name is PrtType:PDF.

There are several INI options you can use to override the naming process and also name other print drivers that require this unique handling.

```
< BatchPrint >
    NoBatchSupport = PDF
    PreLoadRequired= PDF
```

These are the default settings and cannot be overridden. However, you can specify other PrtType print driver definitions you want to fall into these same categories.

Option	Description
NoBatchSupport	Indicates that the named PrtType items, separated by semicolon, do not really support batch transactions and require special handling.
PreLoadRequired	Lets you specify all the PrtType items, separated by semicolon, that should be forced to load the form set prior to the starting print. Most print drivers don't require this special requirement, but some, such as PDF do.

Also, you can name PrtType specific items under the BatchPrint control group to override the normal Device naming option. Here is an example:

```
< BatchPrint >
    PDF = ~HEXTIME .PDF
    RTF = ~HEXTIME -~KeyID .RTF
```

Any batch print sent to PrtType:PDF (picking PDF on the Print window) will override the name and store the current hexidecimal date and time, such as BCF09CA4.PDF, which is an eight-character name, as the name of each transaction's output.

Also, you can combine INI built-in calls as shown in the RTF example. Here any WIP or batch print sent to RTF will name the files using the HEXTIME and the KeyID from the WIP transaction. This will result in names similar to this: BCF099A4-123456.RTF

Note that you must leave a space after the built-in INI function name for it to work properly. That space will not appear in the resulting output name.

Font Cross Reference File Tips

The quality of the created PDF files is in large part influenced by the setup information contained in the font cross-reference (FXR) file. Keep the following tips in mind when looking at your FXR file to optimize the quality of your PDF output.

PostScript font names should be present in your FXR, and all font IDs should contain one of the following PostScript font names in the Setup Data field for PostScript printing. The names of the PostScript fonts are case sensitive.

- Courier
- Courier-Bold
- Courier-Oblique
- Courier-BoldOblique
- Helvetica
- Helvetica-Bold
- Helvetica-Oblique
- Helvetica-BoldOblique
- Times-Roman
- Times-Bold
- Times-Italic
- Times-BoldItalic
- Symbol
- ZapfDingbats
- Courier-Italic
- Courier-BoldItalic
- Univers-Medium
- Univers-Bold
- Univers-MediumItalic
- Univers-BoldItalic

The point size value should be present and should be within 33% of the font height. Font heights are measured in 2400 dots per inch while point sizes are measured in 72 dots per inch, so some conversions to equivalent units will be necessary to determine their relative values.

NOTE: All of the fonts listed above, except for the Univers fonts, are included in Adobe's Acrobat Reader and do not have to be embedded in PDF files.

The spacing value (either fixed or proportional) should be present and accurate. Here is a list of PostScript fonts sorted by spacing value.

Fixed Pitch Fonts	Proportional Fonts
Courier	Helvetica
Courier-Bold	Helvetica-Bold
Courier-Oblique	Helvetica-Oblique
Courier-BoldOblique	Helvetica-BoldOblique
Courier-Italic	Times-Roman
Courier-BoldItalic	Times-Bold
	Times-Italic
	Times-BoldItalic
	Univers-Medium
	Univers-Bold
	Univers-MediumItalic
	Univers-BoldItalic
	Symbol
	ZapfDingbats

The font style value (upright or italic) should be present and accurate; it should match a PostScript font with an equivalent font style.

The font weight (bold or normal) should be present and accurate; it should match a PostScript font with an equivalent font weight.

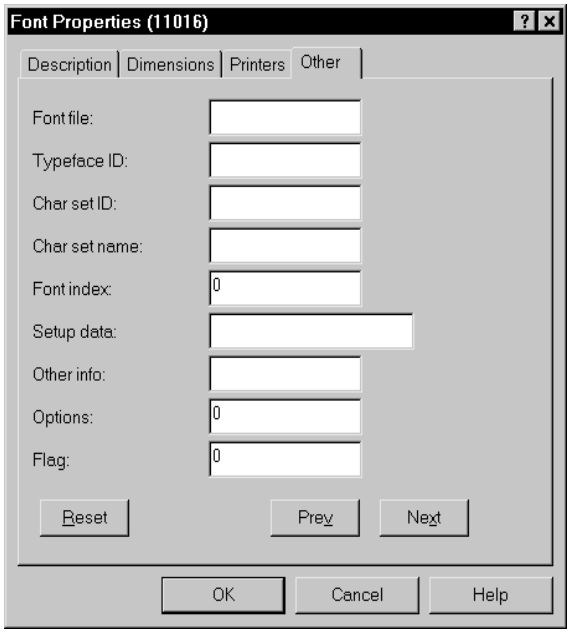
Embedding Fonts

Follow these instructions to embed fonts.

NOTE: You can embed TrueType or PostScript fonts. The PDF Print Driver uses the font file extensions to distinquish between the two types of fonts. TrueType fonts have a *TTF* extension. PostScript fonts have a *PFB* extension.

1. Use a text editor to open the INI file. Then set the DownLoadFonts option in the PrtType:PDF control group to Yes.
2. Next, use the Font Manager to set up your font cross reference file (FXR). Start the Font Manager and select the font cross reference file you want to edit. Then select the font you want to embed and click Edit.

3 On the Font properties window, use the Other tab to set up the font for downloading.



- Enter 1 in the Options field to indicate that the font should be downloaded (a zero (0) tells the system not to download the font).
- Use the Font Index field to group fonts.
- Enter the name of the font file in the Font File field.

4 Set up the remaining fields (Char set ID, Setup Data, and so on) as you would for a PostScript font.

Keep in mind...

- If you set the DownloadFonts option to No, all fonts used in the image are mapped to one of the 14 Type 1 fonts distributed with the system and no fonts are downloaded.
- If you do not define the fields on the Other tab, the system will not download the font—even if you set option to Yes. Instead, the system will map the font to one of the 14 Type 1 fonts.
- For symbol fonts, such as DocuDing, make sure the Char set ID field on the Other tab is set to *WD*.

If you set the DownLoadFont option to Yes,

- The system will download fonts used in the document as long as the Options fields on the Other tab are set to 1.
- The system will map the fonts used in the document to one of the 14 Type 1 fonts if the Options fields on the Other tab is set to zero (0).
- Downloadable fonts with the same value in their Index fields are considered in the same group. Only one font will be downloaded (embedded) for each group.

NOTE: Each font you embed increases the PDF file size by approximately 40kb. See FontCompression on page 211 for information on compressing fonts.

Using the 14 Base Fonts Distributed with Acrobat Reader

Adobe includes the following fonts with Acrobat Reader. You do not have to embed these fonts in PDF files.

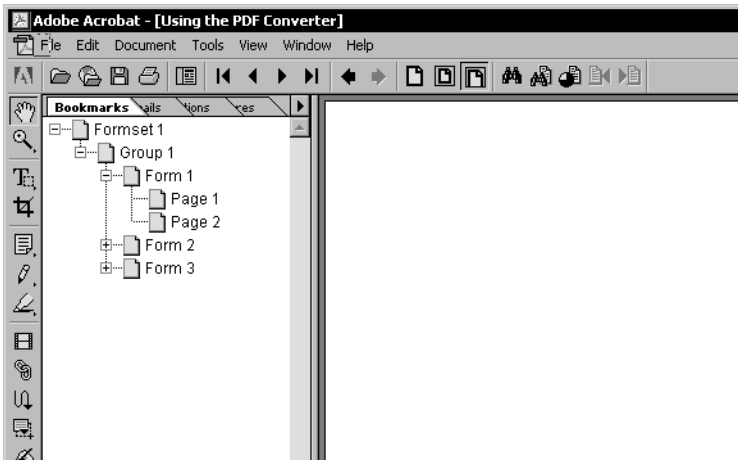
Fixed Pitch Fonts	Proportional Fonts
Courier	Helvetica
Courier-Bold	Helvetica-Bold
Courier-Oblique	Helvetica-Oblique
Courier-BoldOblique	Helvetica-BoldOblique
	Times-Roman
	Times-Bold
	Times-Italic
	Times-BoldItalic
	Symbol
	ZapfDingbats

SETTING UP BOOKMARKS

The PDF Print Driver sets up bookmarks at these levels by default:

- Form set - The text for this bookmark is the recipient name if applicable, otherwise it is the name of the form set.
- Group - The text for this bookmark is the name of the group.
- Form - The text for this bookmark is the description of the form. If there is no form description, the PDF Print Driver uses the name of the form.
- Page - The text for this bookmark is the name of the page.

If the text for any of the bookmarks is blank, the PDF Print Driver inserts text to describe the bookmark level, such as Formset, Group, Form, or Page, and the index for that level. Here is an example:



Creating custom bookmarks

The PDF Print Driver lets you use custom rules to create custom bookmarks in the PDF file. To do this, you must create the custom rule.

With a custom rule, you can use the *extra info* in the FAP objects to store custom bookmark titles. Currently, the system uses *extra 1* and *extra 2*, leaving *extra 3* for bookmark titles.

If you choose do develop a custom rule to use *extra 3* for this purpose, keep these things in mind:

- The setting for the Bookmark option remains the same.
- The maximum length for a custom bookmark title is 128 characters.
- This feature is not a callback, so all bookmark settings using *extra 3* must be finished before you send them to the PDF Print Driver.
- The PDF Print Driver expects to receive a character string for extra 3 and the first eight characters must be *BOOKMARK*. The actual bookmark text begins with the ninth character and is NULL terminated.

- If you do not set *extra 3* (NULL handle) or the first eight characters are not *BOOKMARK*, the PDF Print Driver ignores *extra 3* and uses its original logic to create bookmarks.

Refer to the discussion of the FAPGetExtraInfo and FAPPutExtraInfo functions in the API documentation, available from the DOSS site, for information on getting and setting *extra 3*.

Here's how the system determines the text for a bookmark in a form set:

If you are filtering by recipient, the system...

1. Checks extra3 of the recipient (128 character limit).
2. Checks extra3 of the form set (128 character limit).
3. Check the recipient name (15 character limit).

If you are not filtering by recipient, the system...

1. Checks extra3 of the form set (128 character limit).
2. Checks the form set name (8 character limit).

Collapsing bookmarks

You can create PDF files with collapsed bookmarks. Use the Group parameter for the Bookmark option to control whether bookmarks are expanded or collapsed. Here is an example:

```
< PrtType:PDF >
    Bookmark = Yes,Page,No,Group
```

In this example, the system creates bookmarks down to the Page level. It does not include unnamed objects and it collapses bookmarks below the Group level.

LIMITATIONS

The PDF Print Driver does not currently support the full set of Adobe Acrobat PDF capabilities. The following are a some of the product's limitations.

Type 1 fonts

If the PostScript Font Name/Setup Data setting in the FXR does not match a PDF base font, the PDF Print Driver maps the following PostScript font names into PDF base font names:

- Courier-Italic maps to Courier-Oblique
- Courier-BoldItalic maps to Courier-BoldOblique
- Univers-Medium maps to Helvetica
- Univers-Bold maps to Helvetica-Bold
- Univers-MediumItalic maps to Helvetica-Oblique
- Univers-BoldItalic maps to Helvetica-BoldOblique

Finally, if the PostScript font name fails to map to a PDF base font name using these rules, then fixed pitch fonts map to Courier and proportional fonts map to Helvetica. If a font has bold, italic or bold and italic attributes, the Courier or Helvetica PDF base font with corresponding attributes will be used.

Code pages

Currently, only the ANSI code page (also known as code page 1004) is supported for PDF files. Normally, this will only be an issue if you are trying to support international characters. If you have used AGFA fonts for printing, this should not be an issue.

PDF objects

Although Acrobat Reader supports variable fields, radio buttons, push buttons, list boxes, and hypertext links, the PDF Print Driver does not support the creation of these objects within a PDF file.

Searching for text in PDF files

If you retrieve archived Documerge print streams or Documaker archives via Documanage and then use the PDF print driver in Docupresentment, you will produce *PDF Normal* files. PDF Normal files can include both text and graphics and you can search for text in these files.

Documanage can also convert archived Documerge print streams into TIFF (bitmap) files. If you send the TIFF files to Docupresentment, the PDF Print Driver produces *Image Only* PDF files. Since these TIFF files are bitmaps, there is no text to search.

CREATING HTML FILES

You can create HTML files by simply printing to the HTML print driver. The HTML print driver includes support for:

- Boxes – solid and shaded colors only
- Bar codes
- Charts
- Vectors – solid and shaded colors only
- Logos – converted to JPG files by the driver
- Lines – solid and shaded colors only. Dashed lines are supported but do not take the line characteristics as specified. The spacing and length of dashed lines are defaulted by the HTML 4 specification.
- Shaded areas – solid and shaded colors only
- Text areas
- Text
- Variable fields

Use these INI options to set up the HTML print driver:

```
< PrtType:HTML >
    Device          = Sample.htm
    DirLinks        = Yes
    CollapsePage    = Yes
    PageBreaks      = Yes
    HR              = Size=2 Width=100% Color=Black
    AllowInput      = No
    DownloadFonts   = Yes,Enabled
    TemplateFields  = No,Enabled
    Module          = HTMW32
    PrintFunc       = HTMPrint
    JavaScript     = format.js
    JavaScript     = help.js
    JavaScript     = data.js
    ColorSheet      = iecolor.css
    MultiPage       = No
    ImagePath      = e:\rpex1
    ImagePathCreate = e:\rpex1\new_images
    PageNumbers    = Yes
    SendColor      = Yes,Enabled
    AllowColorSheetLink = Yes
    CreateScriptFile   = Yes
    DumpScript         = Yes
    HiddenFieldScript  = textMergeP(this);
    ScriptPath =  e:\rpex1\deflib
    ScriptPathCreate = e:\scripts
    EntryBackColor = #BBEBE6
    EntryFontColor = #FFFFFF
    SplitText      = -1
    BmSub          = Yes
    BmSubChar      = '_'
    IMG_ZIndex     = 100
```

Option	Description
Device	Sample.htm is the name of the HTML file generated when printed from Studio, Image Editor, or Entry.
DirLinks	Enter Yes to include Next and Prev links on pages. The default is No.
CollapsePage	Enter Yes to remove white space at the bottom of page. The default is No.
PageBreaks	Forces a page break between pages during HTML print. The default is Yes.
HR	(Header Rule) Displays a line between pages. You can configure the size, color and width. See example to see how to configure the rule.
AllowInput	Enter Yes if you are running iPPS or iDocumaker. The default is No.
DownloadFonts	Enter Yes to download of PCL fonts. Include *Enabled* to tell the system to display a check box the user can use when printing from Documaker Workstation, Studio, or Image Editor.
TemplateFields	Enter No to omit the *X*s from variable fields.
Module	Enter the name of the print driver DLL file.
PrintFunc	Enter the name of function within the print driver for print.
JavaScript	Specify any script files which contain functions you want the system to use.
ColorSheet	Specify the style sheet that contains the color information. The default is WSCOLOR.CSS. The WSCOLOR.CSS file contains web safe colors. The IECOLOR.CSS file contains Internet Explorer colors.
AllowColorSheetLink	Use this option to tell the HTML print driver if a color style sheet link exists. The default is Yes. If the color information is inline in the HTML page, omit the link to an external color style sheet.
MultiPage	Enter Yes only when running the Image Editor and you want to print multiple page FAP files with each page as a separate HTML file. This feature is not supported when printing from Documaker and Documaker Workstation.
ImagePath	Enter the path for JPG files.
ImagePathCreate	Enter a path to indicate where you want the images created.
PageNumbers	Enter Yes to print page numbers. The default is No.
SendColor	Enter Yes,Enabled to print in color. The default is No.

Option	Description

CreateScriptFile	Enter Yes is if you want the HTML driver to generate script files. The script files correspond to the name of the FAP. For example, if the Q1ADDR.FAP file contained either an inline script or a call to a function in an external script file, the generated script file is named Q1ADDR.JS.
	The corresponding function for a function that occurred in an external DAL script file would be called. For example:
	`_q1addr_dal_1ST NAME ZIP(obj)`
	where the *q1addr* represents the name of the FAP file, *_dal* implies it was contained in an external script file, and *_1ST NAME ZIP* is the name of the field. If the DAL script is inline and not in an external file, the example would look like this:
	`_q1addr_1ST NAME ZIP(obj).`

DumpScript	Enter Yes to have the inline script written to the Java script file as a comment. For example if the inline script for an effective date field existed it would be written to the js file as:
	`function _samplefap_EFFDTE(obj);`
	`{`
	`/*HOLDDTE = DATEADD(@("EFFDTE"),,,,1); SETFLD ( HOLDDTE, "EXPDTE");*/`
	`}`
	If you enter No for this option, the inline script is not written as a comment. Instead, it is handled as if it were in an external file. The corresponding function would be:
	`_samplefap_EFFDTE(obj);`
	`{ /*alert( EFFDTED code goes here);*/ }`

HiddenFieldScript	This tells the system to display a box the user can use to enter data onto the form. In this example, `textMergeP(this);` The hidden field has a length of one.
ScriptPath	Enter the path to the script (.js) files generated by the HTML print driver. This is used for script files references in the HTML files.
ScriptPathCreate	Enter the path that points to where the driver creates the script files.
EntryBackColor	This is the background color for the data entry fields. This can be overridden in formatting script functions. The default is #B0E0E6.
EntryFontColor	Enter the color for the font used in data entry fields. The default is #FFD2D2.
SplitText	The default (-1) tells the system to split the text so a single word appears on each HTML line. Enter zero (0) to tell the system not to split text in the HTML file.

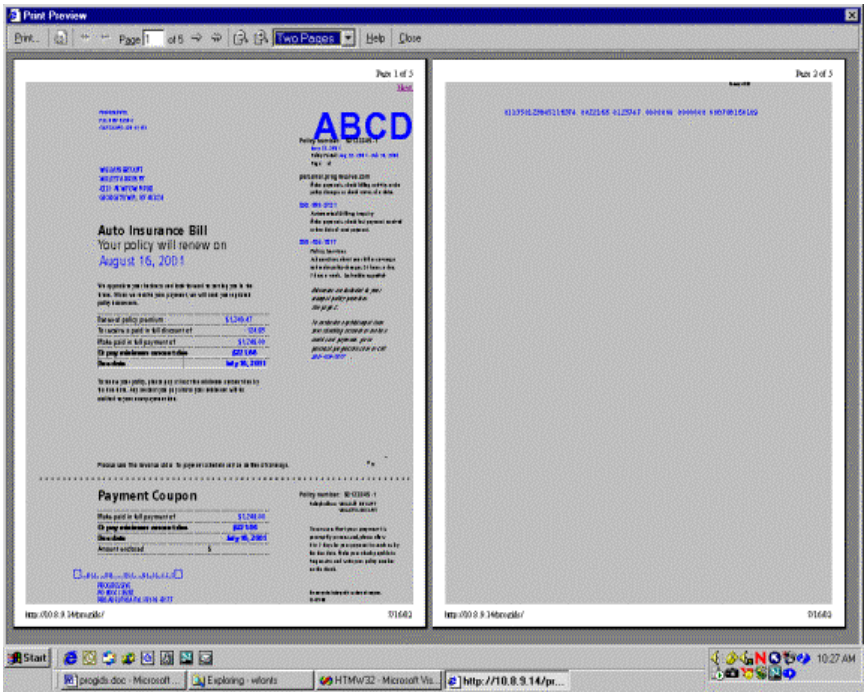
Option	Description
BmSub	Enter No if you do not want the system to substitute invalid characters with the character you specify in the BmSubChar option. The default is Yes. For instance, if you omit this option but specify *X* in the BmSubChar option, the system substitutes *X*s for invalid characters.
BmSubChar	Enter the character you want the system to use to replace invalid characters. The default is an underscore (_).
IMG_ZIndex	The z-index indicates the stacking order of objects based on the order in which those objects appear in the HTML file. Higher values place objects closer to the front while lower values place them further to the back. Objects with the same value are stacked based on the order in which they appear in the HTML source. For instance, a positive value positions an object above text that has no defined z-index. A negative value would place the object below the same text. If you omit this option or leave it blank, the system will not layer objects.

Also, you can run the GenData program in single step mode and create separate files for each transaction. Each transaction will have its own HTML file if you include INI options similar to these:

```
< PrintFormSet >
    MultiFilePrint = Yes
    LogFileType    = XML
    LogFile        = d:\fap\mstrres\rpex1\data\jlog
< Printer >
    PrtType        = HTML
; Printers is for the GUI dropdown selection (uncomment many)...
< Printers >
    PrtType        = AFP
    PrtType        = PCL
    PrtType        = XER
    PrtType        = HTML
```

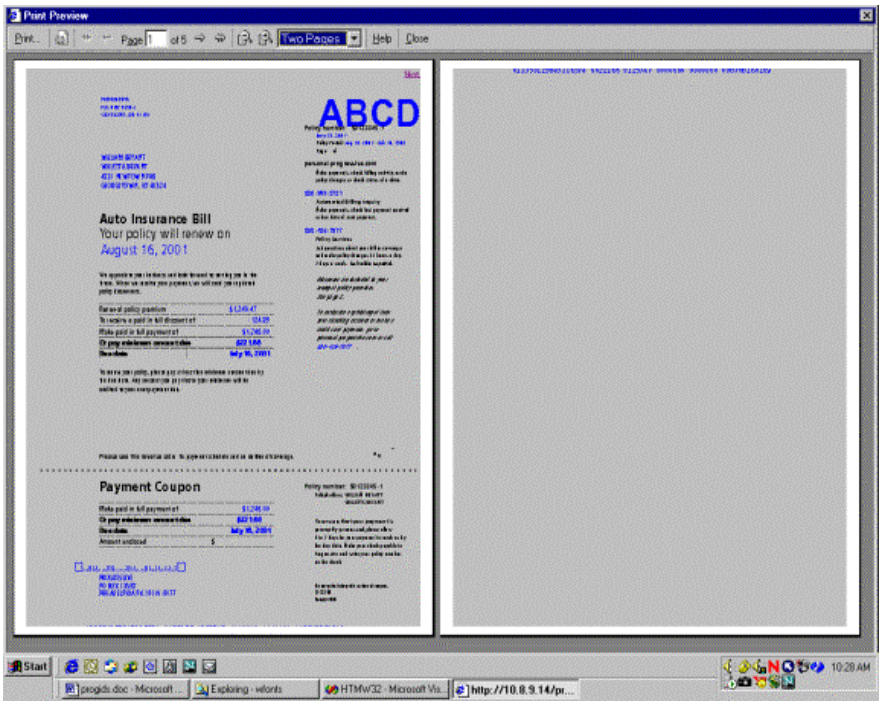
NOTE: To make sure the page looks the same to all users, only use fonts which all users have. Otherwise, fonts are substituted.

The look of the printed HTML file depends on the settings in affect for the user's browser. For example, the following shows printed output when the page setup includes a header and footer with a top and bottom margin of 0.166":



Note how the last line of the first page has overflowed onto a new page.

The following example shows the printed output when the margins remain at 0.166" but there are no header and footers:



Note how the last line of the page has moved up but still has run over onto a second page.

NOTE: The company logo appears as *ABCD* in this example. This can indicate a font problem. For instance, Internet Explorer can display HTML that uses raster fonts as long as the raster font is on the user's machine. Not all Windows print drivers, however, support printing with raster fonts. In this example, if this happened the output would print as *ABCD*.

PRODUCING TABLE INFORMATION FOR TEXTMERGE PARAGRAPHS

The system can produce table information for HTML documents generated from FAP documents which include text areas that have the TerSubstitute Pre-Edit procedure enabled.

The following attribute information is now produced for the hidden input tag that corresponds to an <iframe> element for a text merge paragraph:

```
attribType = tersub
TABLEID = ID value of the table.
TABLEFILE = The file value of the table.
```

Here is an example:

```
<IFRAME ID="PSELECTION" NAME="PSELECTION"
ONFOCUS="window.frames.editBar.displayToolbar(this,true);"
FRAMEBORDER="0" STYLE="BORDER-BOTTOM: buttonface 1px solid; BORDER-
LEFT: buttonface 1px solid; BORDER-RIGHT: button face 1px
solid;BORDER-TOP: button face 1px solid;height: 0.80 in; width: 6.10
in; " TABINDEX="3"></IFRAME>
<INPUT TYPE="hidden" NAME="PSELECTION" ACCESSKEY="G" TABLEID="Denial
Reasons" TABLEFILE="LTRENTRY" attribType="tersub"/>
```

You can use this information to call the DPRTblLookUp rule in another request to retrieve a list of paragraphs for insertion into the text merge area. This information can in turn be used to call the DPRFap2Html rule to retrieve an HTML representation of the appropriate TerSub paragraph so it can be inserted into the <iframe> element.

This lets iPPS or iDocumaker query real-time TERSUB information from IDS.

Creating XML Output

The XMPLIB library lets you use IDS and Documaker to create Oracle Insurance standard XML output. This lets you unload Oracle Insurance standard XML output from the GenPrint or GenData programs (using the PrintFormset rule).

Here is an example of the INI setup this feature requires:

```
< Printers >
    PrtType = XMP
< PrtType:XMP >
    Module = XMPW32
    PrintFunc = XMPPrint
```

When using with Documaker, be sure to use the MultiFilePrint functionality to create a separate XML file per transaction. If multiple XML files are written into the same file, the file will not load in an XML parser, browser, or editor.

NOTE: For more information on the Oracle Insurance standard XML format, see Working with XML.

Chapter 4

Using Print Preview

Print Preview lets you view work-in-progress (WIP) on-line using DPRWIP rules and the Internet Document Server (IDS). Source files coded in JSP generate the HTML pages IDS displays.

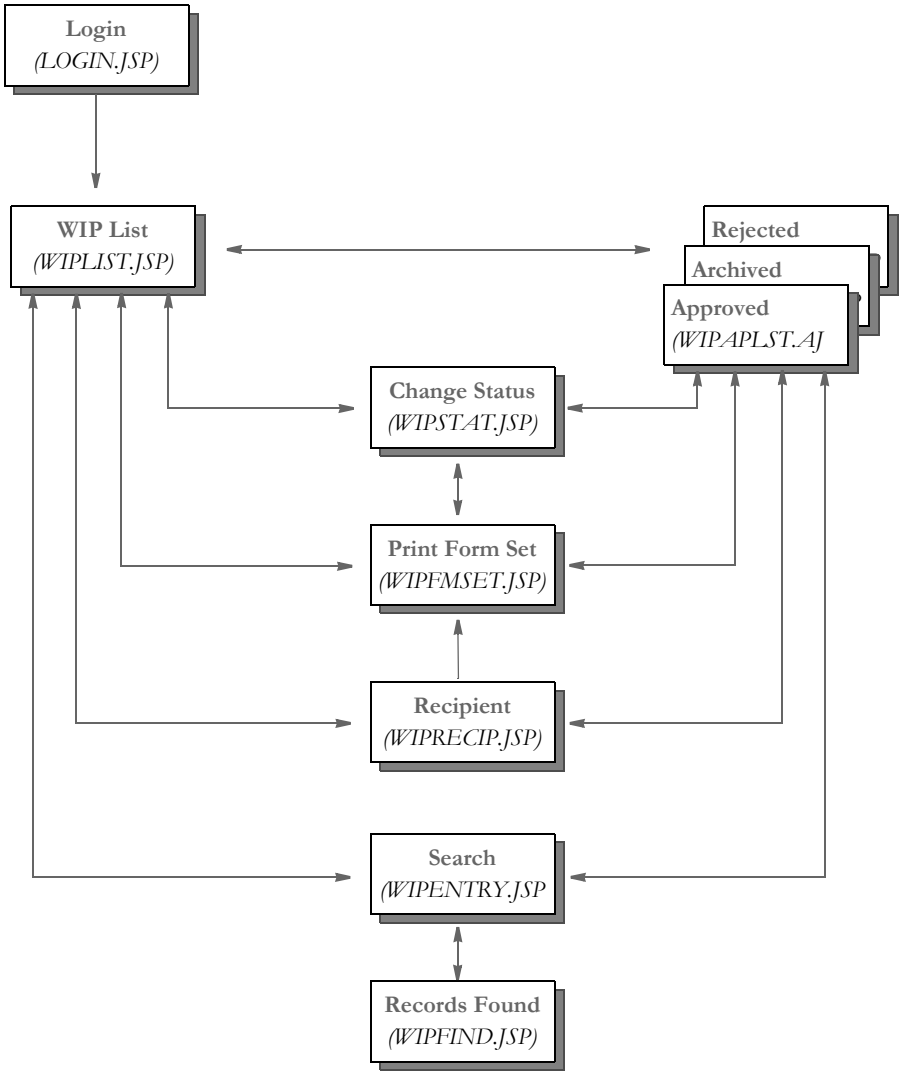
NOTE: For more information on setting up IDS and using the DPRWIP rules, see Using the Internet Document Server on page 9 and the SDK Reference, respectively.

Print Preview includes several HTML pages which let you log into the system, select WIP transactions, view transactions, select recipients, and print form sets.

This appendix includes information about...

- Touring Print Preview on page 239
- Using the JSP Files on page 245
- Customizing Print Preview on page 247

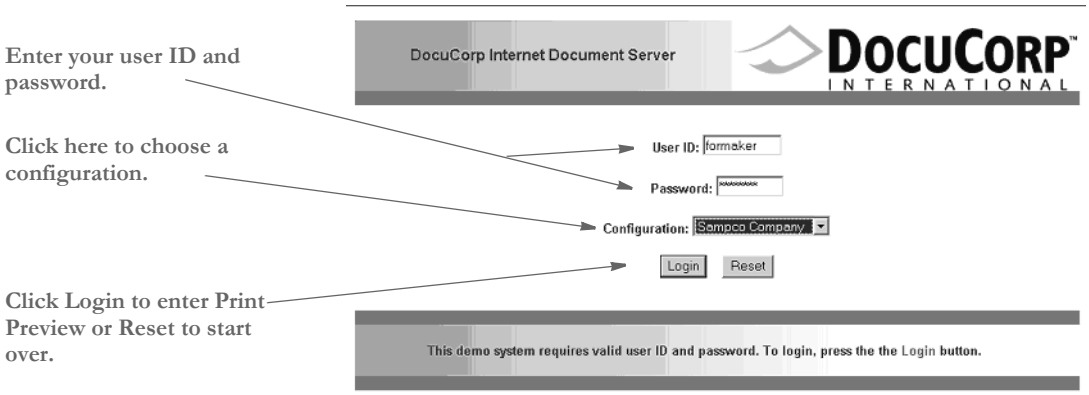
This illustration shows the HTML pages and some, but not all, of the ways you can navigate among the pages.



Touring Print Preview

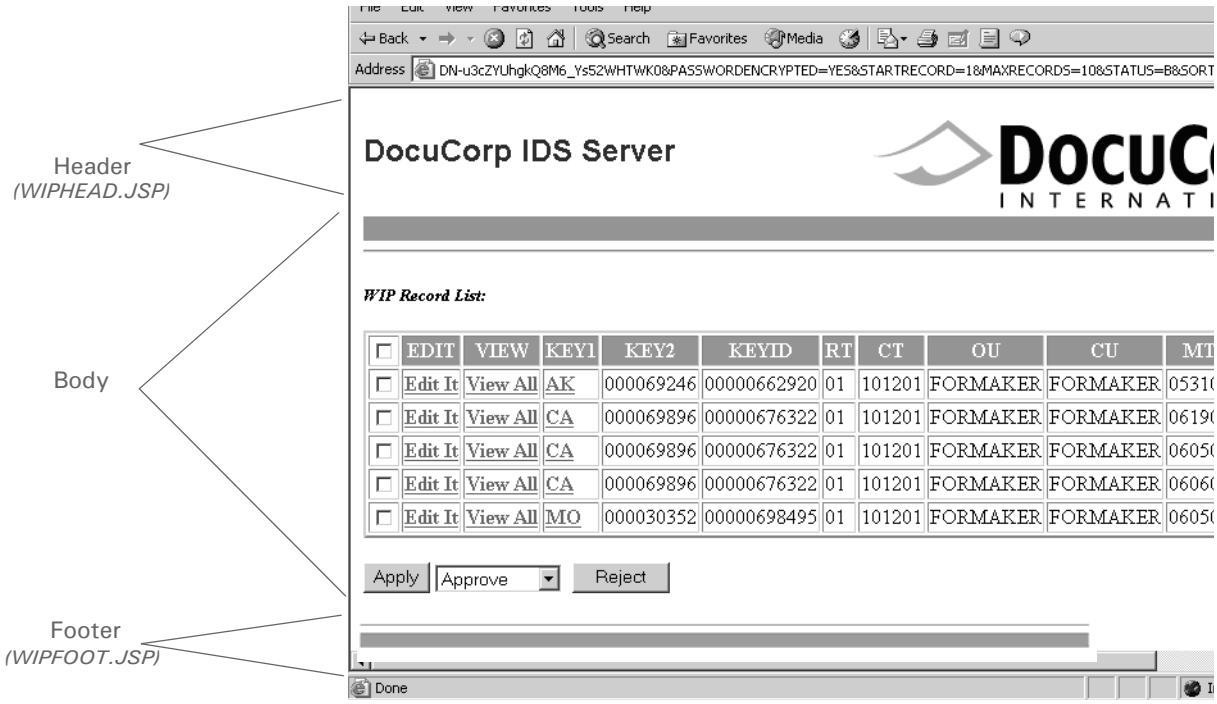
Follow these steps for a tour of Print Preview:

1 To login onto the system, enter a user ID and password, select a configuration from the list, and click on the Login button.



If the login is unsuccessful, an error message appears and you can try again. After you log in, your browser takes you to the WIP List page.

2 The WIP List page lets you view WIP records. The page has three parts: header, body, and footer. You can customize the JSP files that create these parts to change appearance of the page.



NOTE: The status codes are defined in the Status_CD control group. The options are WIP (W), Approved (AP), Archived (AR), and Rejected (RJ), and so on.

These buttons on the WIP List link this page to other pages:

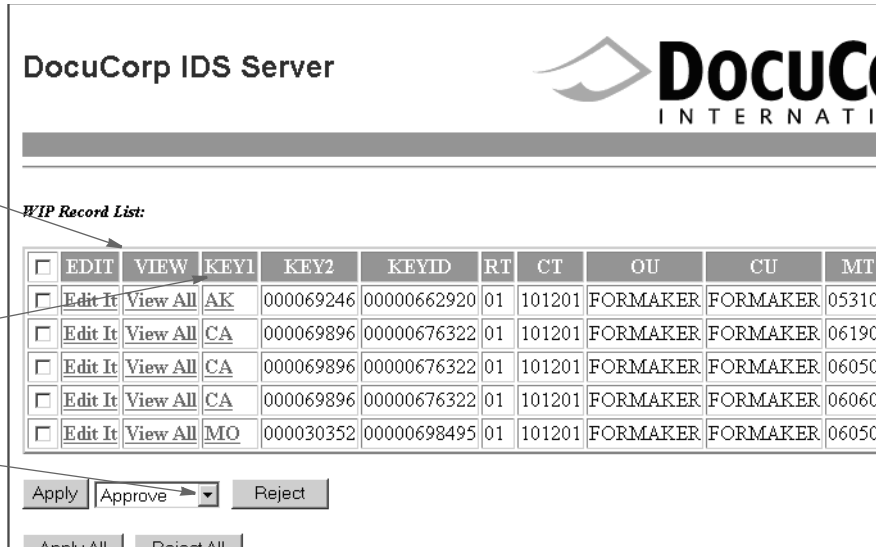
Use this button	To...
Go	Display a drop-down list of sort keys defined in the WIP.DFD file. The USER TAG and DOC TAG keys are hard coded. Other keys are based on the WIPKey array defined in the PrtView_WIPTable INI control group. When you select a key and click Go, the records are re-arranged based on the key you selected. This key is then used until you change it on the WIP List page. You can hide the button and list by setting the Sort variable to *Mask* in the LOGIN.JSP file.
Next Set Prev Set	Move through the set of records. These buttons only appear when relevant. Retrieved records are displayed based on the record number.
Apply Apply All	Approve or archive records. You define the drop down list that appears using the OptKey and AppTxt arrays in the PrtView_WIPTable INI control group. To approve or archive a record, you select the records, select what to apply, and then click Apply. Clicking Apply All tells the system to approve or archive all records on the page. The system prompts you for confirmation before it changes the status code. Approved or archived records are removed from the WIP List page as soon as their status changes.
Reject Reject All	Reject records. The system prompts you to confirm the action before it changes the status code.
Show Show Rejected	Show approved, archived, or rejected records. You define the list that appears using the OptKey and ShwTxt arrays in the PrtView_WIPTable INI control group.
Find Records	Search for records. The Search page appears so you can enter the criteria you want the system to use.

In addition to the buttons, you can click in the first column to go to the Print Form Set page where you can print copies for all recipients. Clicking in the first field in the second column takes you to the Recipient page where you can first select a recipient and then print a copy.

Click in this column to go to the Print Form Set page where you can print copies for all recipients.

Click a field in the Key1 column to go to the Recipient page where you can print copies for a single recipient.

Click here to select a specific record status to display.

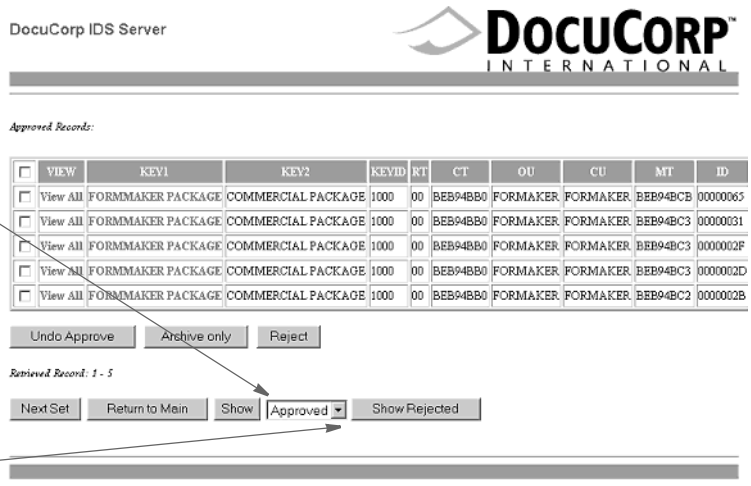


You can click on the arrow next to the Show button to select the type of records you want to display. For instance, this lets you view approved, archived, or rejected records.

3 Make sure *Approved* appears next to the Show button, then click the Show button to go to the Approved Records page.

You can select a different status to display the Archived Records page and review records with an archived status code.

Use the Show Rejected button to see rejected records.



NOTE: This example HTML page shows two ways to select statuses. You can use the drop-down list to select from Approved and Archived or you can click a button to see Rejected records. You can customize the JSP pages to meet your needs.

This page includes several buttons also found on the WIP List page: Next Set, Prev Set, Show Approved, Show Rejected, Archive Only, and Reject. These buttons function the same way as on the WIP List page.

The buttons unique to this page are described in the following table:

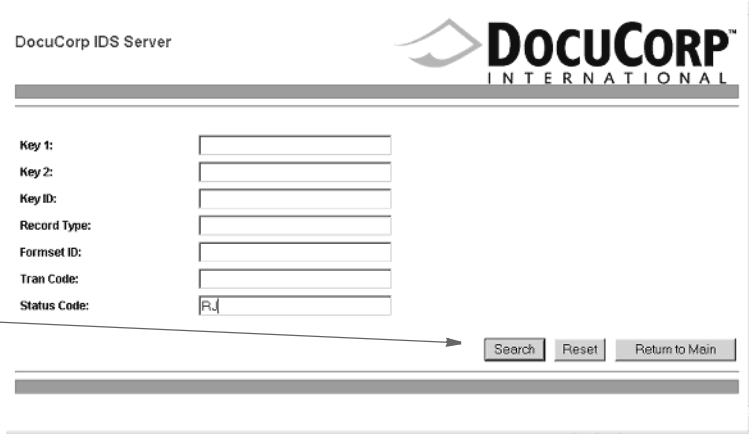
Use this button	To...
Undo Approve	Click this button to change status code of selected records back to the *WIP* status. The changed records are removed from the Approved Records page and added back to the WIP List page. Similarly, on the Archived Records page there is Undo Archive button and on Rejected Records page there is Undo Reject button.
Return to Main	This button takes you back to the WIP List page.

Just like the WIP List page, clicking the first or second columns takes you to the Print Form Set page or the Recipient page, as described previously.

4 Click the Return to Main button and then click the Find Records button on the WIP List page. The Search page appears.

Use the fields on this page to enter the criteria you want the system to use when it searches for records.

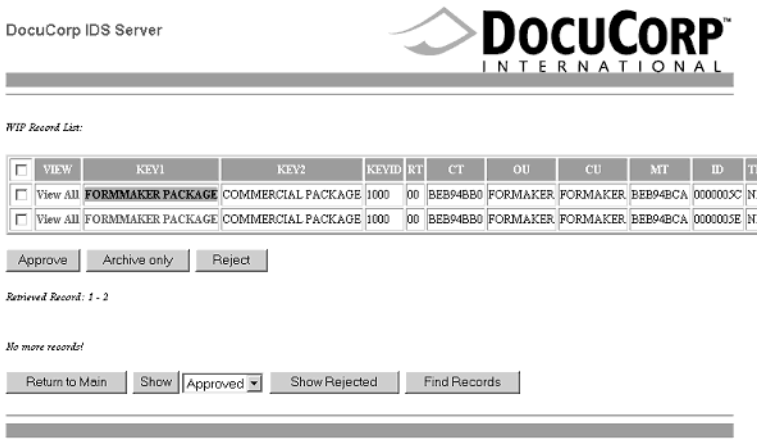
Click the Search button to start the search.



Use this page to enter criteria you want the system to use to search for records. You can change the PrtView_WIPTable control group to customize the fields that appear on this page and to decide which keys to use for searching.

This page has three buttons. The Search button starts the search, Cancel clears the entry fields, and Return to Main takes you to the WIP List page or other parent list page, such as the Approved Records page.

5 Click the Search button, the Records Found page appears. This page lists the records the system found.



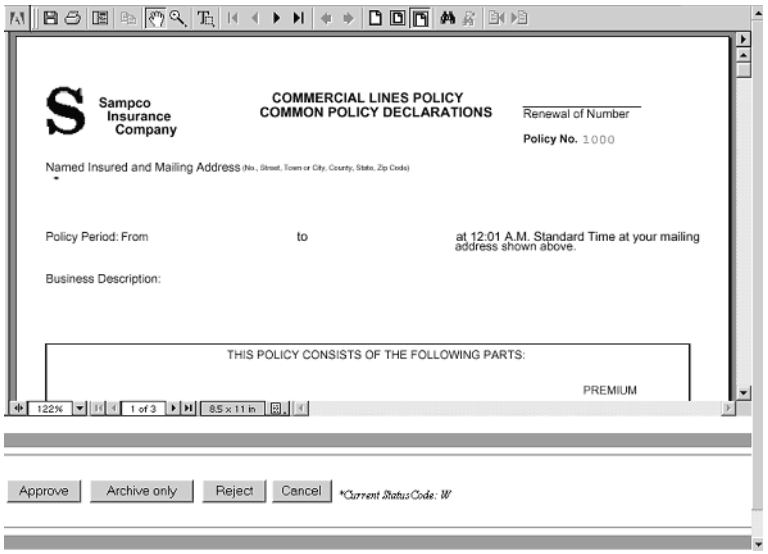
This page includes the same buttons as those on the WIP List page.

6 Click in the first column. The system creates a PDF file of the form set and displays it in the Print Form Set page. Here is an example:

The form set appears as a PDF file using Acrobat Reader or the Acrobat plug-in used by your browser.

Acrobat includes tools you can use to view the form set on the toolbar.

These buttons let you change the status of the form set.



Anytime you click in the first column of a list page, such as the WIP List, Approved Records, Rejected Records, or Records Found, the system creates a PDF file that shows *all* recipient copies of the form set.

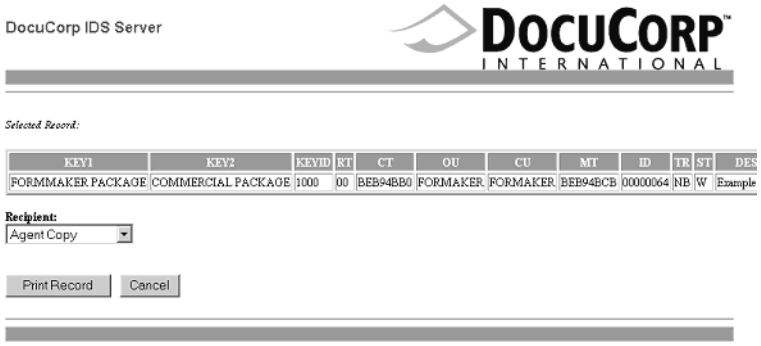
NOTE: To display a PDF file, you must have the Acrobat® Reader from Adobe® Systems. You can download the free reader at Adobe's web site: www.adobe.com. You will also find a copy of Acrobat Reader on the Internet Document Server Installation CD. For best results, use version 7.0 or later.

The Print Form Set page includes these buttons:

Button	Click to...
Approve	Change the form set's status to *Approved*.
Archive Only	Change the form set's status to *Archived*.
Reject	Change the form set's status to *Rejected*.
Cancel	Return to the list page you came from.

7 Click Cancel to return to the Records Found page. Next, click in the second column to display the Recipient page.

Click here to select the recipient.



Anytime you click in the second column of a list page, the Recipient page appears to show you the information for the record you selected. Pick a recipient from a list and then click on Print Record to go to the Print Form Set page.

Use this page to print a copy of an individual form set.

Using the JSP Files

Print Preview includes these ASP files which you maintain:

LOGIN.JSP	WIPLIST.JSP
WIPAPLST.JSP	WIPENTRY.JSP
WIPFIND.JSP	WIPSTAT.JSP
WIPFMSET.JSP	WIPRECIP.JSP

Plus these JSP files which you can customize:

WIPHEAD.JSP	WIPFOOT.JSP

This table describes the files you maintain:

ASP file	Description	Customizing
LOGIN.JSP	This code creates the Login page	You can change the appearance of the login page by... - Changing the JSP code - Including separate JSP codes, such as WIPHEAD.JSP and WIPFOOT.JSP Including separate JSP codes can make the system easier to maintain. The value of passing parameters: MAXRECORDS determines how many records appear on each list page. SORT determines whether the sorting list appears on the WIP List page. PASSWORDENCRYPTED determines if the system should use an encrypted password.
WIPLIST.JP	The Login page links to the WIP List page. The WIP List page and all other list pages consist of three parts: header, body, and footer. You can customize the body using the PrtView_WIPTable control group to specify a different configuration.	You customize the header and footer by changing the WIPHEAD.JSP,and WIPFOOT.JSP files. These JSP files are included in the WIPLIST.JSP file as *include files*.
WIPAPLST.JSP	The WIP List (WIPLIST.JSP) and Records Found (WIPFIND.JSP) pages link to this page. This JSP file creates an approved, archived or rejected record list page, as necessary.	You customize this page just as you do the WIP List page.
WIPAPPR.JSP	You use this page to change the status code for all WIP records. It is linked only to the WIP List page. After you approve all or reject all records, you go back to an empty WIP List page, so you can decide your next step.	You cannot customize this page.

ASP file	Description	Customizing
WIPENTRY.JSP	The Find Record button is linked to this entry table page. To find records, you enter one or more fields as search keys. The Search button links this page to the WIPFIND.ASP which creates the Record Found List.	You can determine the keys used in searches. Customize this page just as you would the WIP List page.
WIPFIND.JSP	This code creates a list of the records have a given status, such as *WIP, Approved, Archived*, or *Rejected*.	You customize this page just as you do the WIP List page.
WIPSTAT.JSP	This code is used to change the status code for selected records. Once the status codes are changed, you return to the parent list page, such as the WIP List.	You cannot customize this page.
WIPFMSET.JSP	This code is used to retrieve a form set from a parent record list such as the WIP List, Approved Records, Archived Records, Rejected Records, or Records Found page. If the request type specifies DPRPrint rule, the form set is converted to PDF format and the PDF file is displayed on this page. If you select and press the Approve or Reject button, you return to the WIPSTAT.JSP where you can approve or reject the record. From there you are taken to the parent list page.	You can customize the header and footer of this page by including WIPHEAD.JSP and WIPFOOT.JSP files.
WIPRECIP.JSP	Clicking in the second column on a parent list takes you this page, called the Recipient page. This page shows the selected record and asks you to select a recipient from the list. You can click on the Print Record button to go to the page created by the WIPFMSET.JSP file where you can view and print a single copy of the form set. Click Cancel to return to a parent list page.	You customize this page just as you do the WIP List page.

Customizing Print Preview

You can customize these JSP files to make Print Preview meet your needs:

- WIPHEAD.JSP
- WIPFOOT.JSP

WIPHEAD.ASP

You can modify this JSP code to customize the header part of the page. It is in HTML format, and you can modify it as necessary.

Here is an example:

```
<TABLE BORDER=0 WIDTH=100% VSPACE="0" HSPACE="0" CELLPADDING="0"
CELLSPACING="0">
<TR>
 <TD WIDTH="80%">
  <FONT FACE="Arial,Lucinda Sans,Gill Sans" # COLOR="d01010">
  <BR>
  <H2><B> Docucorp IDS Server </B></H2> <BR>
  </FONT>
 </TD>
 <TD>
  <IMG ALIGN=RIGHT TOP SRC="..\nlogo30s.gif" ALT=Orcom> <BR>
 </TD>
</TR>
</TABLE>
<TABLE BORDER=0 # BGCOLOR="#80a0a0" WIDTH=100% VSPACE="0" HSPACE="0"
CELLPADDING="0" CELLSPACING="0">
 <TR><TD> </TD></TR>
</TABLE>
<HR>
<% string bkgdcolor=”#808080">
<% string border=”=2">
```

Where *bkgdcolor* determines the color for the background and *border* determines the style of border for the tables which comprise the body.

WIPFOOT.JSP

You can modify this JSP code to customize the footer part of the page. It is in HTML format, and you can modify it as necessary.

```
<HR>
<TABLE BORDER=0 # BGCOLOR="#80a0a0" WIDTH=100% VSPACE="0" HSPACE="0"
CELLPADDING="0" CELLSPACING="0">
 <TR><TD> </TD></TR>
</TABLE>
```

Using INI Options to Customize Print Preview Screens

You can also customize the WIP index layout using INI options, instead of having to change the JSP scripts.

You do this by moving the arrays of column names and headings from the JSP code into an INI file. The INI options are based on the CONFIG value and can be specific to each WIP resource setup. You can store the INI options in the DAP.INI file if you need them to be global in scope in the particular CONFIG.INI file.

You customize the WIP list by changing the options in the PrtView_WIPTable control group. To read the control group from the INI file, such as RPEX1.INI, add the DPRWipTableParms rule in every request type that displays a record list. Here is an example of setting the request type for a WIP list:

```
< ReqType:WLT >
    function = atcw32->ATCLogTransaction
    function = atcw32->ATCLoadAttachment
    function = dprw32->DPRSetConfig
    function = atcw32->ATCUnloadAttachment
    function = dprw32->DPRDecryptLogin
    function = dprw32->DPRDefaultLogin
    function = dprw32->DPRCheckLogin
    function = dprw32->DPRWipTableParms
    function = dprw32->DPRGetWipList
```

This rule passes the value of each array option to the client side. The JSP code, such as WIPLIST.JSP, gets the value and translates it to an array for the WIP table customization by calling:

```
wipkey = DSI.getWipKeys(Rst, "WIPKEY");
```

Here is an example of the PrtView_WIPTable control group:

```
< PrtView_WIPTable >
    ;table
Fields = KEY1,KEY2,KEYID,RECTYPE,CREATETIME,ORIGUSER,CURRUSER,
MODIFYTIME,FORMSETID,TRANCODE,STATUSCODE,FROMUSER,FROMTIME,
TOUSER,TOTIME,DESC,INUSE,ARCKEY,APPDATA,RECNUM
Table = KEY1,KEY2,KEYID,RT,CT,OU,CU,MT,ID,TR,ST,DESC,RECNUM
WIPKey = KEY1,KEY2,KEYID,RECTYPE,CREATETIME,ORIGUSER,CURRUSER,
MODIFYTIME,FORMSETID,TRANCODE,STATUSCODE,DESC,RECNUM
;dropdown
OPTKey = AP,AR
APPTxt = Approve,Archive only
SHWTxt = Approved,Archived
;entry table
EntryTBL = Key 1,Key 2,Key ID,Record Type,Formset ID,Tran Code,Status
Code
EntryKey = KEY1,KEY2,KEYID,RECTYPE,FORMSETID,TRANCODE,STATUSCODE
```

If the control group is not found, the default arrays are used.

Make sure you include all of the options. These include: Fields, Table, WIPKey, OPTKey, APPTxt, SHWTxt, EntryTBL, and EntryKey. The required Java modules are: IDSJSP.jar, DSIJava.jar, and DocucorpUtil.jar.

NOTE: The Print Preview plug-in can download additional resource files on demand.

CUSTOMIZING THE WIP LIST PAGE

You can customize the WIP List page by passing arrays from IDS to the client side. You set the data in the arrays by changing the options of the PrtView_WIPTable control group. This control group includes these options:

Option	Description
Fields	Defines all fields defined in the WIP BDF file.
Table	Defines the name of each column in the record list.
WIPKey	Retrieves field values from the WIP retrieval base. Retrieved field values are filled into each column defined by the corresponding table array.
OptKey	Includes the status code of the Show or Apply drop-down list.
AppTxt	Includes the options of the Apply list.
ShwTxt	Includes the options of the Show list.
EntryTbl	Includes the name of each field to be entered.
EntryKey	Defines the search keys. You enter the value for each key so the system can search the records.

Here is an example:

```
< PrtView_WIPTable>
;table
    Fields  = Key1, Key2, KeyID, RecType, ...
    Table   = Key1, Key2, KeyID, RecType, ...
    WIPKey  = Key1, Key2, KeyID, RecType, ...
;dropdown
    OptKey  = AP, AR
    AppTxt  = Approve, Archive only
    ShwTxt  = Approved, Archived
;entry table
    EntryTbl= Key1, Key2, KeyID, ...
    EntryKey= Key1, Key2, KeyID, ...
```

If you omit the PrtView_WIPTable control group, the system uses its default settings. Be sure to include *all* INI options in the control group.

To read the control group from a client INI file, such as the RPEX1.INI file, you must include the DPRWipTableParms rule in every REQTYPE that requires displaying a record list. Here is an example of how to set the REQTYPE for a WIP list:

```
< ReqType:WLT >
    function        = atcw32->ATCLogTransaction
    function        = atcw32->ATCLoadAttachment
```

```
function        = dprw32->DPRSetConfig
function        = atcw32->ATCUnloadAttachment
function        = dprw32->DPRDecryptLogin
function        = dprw32->DPRDefaultLogin
function        = dprw32->DPRCheckLogin
function        = dprw32->DPRWipTableParms
function        = dprw32->DPRGetWipList
```

NOTE: The WFD and WRC request types also display records and need the DPRWipTableParms rule.

This rule passes the value of each array option to the client side. The JSP code, such as that in the WIPLIST.JSP file, gets the value and translates it to an array for the WIP table customization by calling the following:

```
wipkey = DSI.getWipKeys(Rst, "WIPKEY");
```

The required Java modules are:

- IDSJSP.jar
- DSIJava.jar
- DocucorpUtil.jar

Creating User Lists for Print Preview

The standard Print Preview rules use the same ReportTo logic as Documaker Workstation to get the list of WIP records from WIP. If you need to create a custom version of user authentication which is not built from the USERINFO database, use the USERLIST attachment variable.

This attachment variable contains a comma-delimited list of user IDs. The system uses this list instead of the information in the USERINFO database. If you want to use the current user ID (logged on user ID) to get the WIP list, you must also include it in USERLIST attachment variable.

Request Types for Print Preview

Here are examples of the request types you can use with Print Preview:

```
; Login and get WIP list
< ReqType:WLG >
    function = atcw32->ATCLogTransaction
    function = atcw32->ATCLoadAttachment
    function = atcw32->ATCUnloadAttachment
    function = dprw32->DPRSetConfig
    function = dprw32->DPRDecryptLogin
    function = dprw32->DPRDefaultLogin
    function = dprw32->DPRLoginUser
    function = dprw32->DPRGetWipList

;Get WIP list and display records
< ReqType:WLT >
    function = atcw32->ATCLogTransaction
    function = atcw32->ATCLoadAttachment
    function = atcw32->ATCUnloadAttachment
    function = dprw32->DPRSetConfig
    function = dprw32->DPRDecryptLogin
    function = dprw32->DPRDefaultLogin
    function = dprw32->DPRCheckLogin
;   function = dprw32->WipTableParams ;Used only with JSP to
customize display
    function = dprw32->DPRGetWipList

; Approve all WIP records
< ReqType:WAL >
    function = atcw32->ATCLogTransaction
    function = atcw32->ATCLoadAttachment
    function = atcw32->ATCUnloadAttachment
    function = dprw32->DPRSetConfig
    function = dprw32->DPRDecryptLogin
    function = dprw32->DPRDefaultLogin
    function = dprw32->DPRCheckLogin
    function = dprw32->DPRApproveWipRecords

; Get recipients and display records
< ReqType:WRC >
    function = atcw32->ATCLogTransaction
    function = atcw32->ATCLoadAttachment
    function = atcw32->ATCUnloadAttachment
    function = dprw32->DPRSetConfig
    function = dprw32->DPRDecryptLogin
    function = dprw32->DPRDefaultLogin
    function = dprw32->DPRCheckLogin
;   function = dprw32->WipTableParams ;Used only with JSP to
customize display
    function = dprw32->DPRGetWipRecipients

; Update WIP records with new status code
< ReqType:WST >
    function = atcw32->ATCLogTransaction
    function = atcw32->ATCLoadAttachment
```

```
    function = atcw32->ATCUnloadAttachment
    function = dprw32->DPRSetConfig
    function = dprw32->DPRDecryptLogin
    function = dprw32->DPRDefaultLogin
    function = dprw32->DPRCheckLogin
    function = dprw32->DPRUpdateWipRecords

;Get form set
< ReqType:WFS >
    function = atcw32->ATCLogTransaction
    function = atcw32->ATCLoadAttachment
    function = atcw32->ATCUnloadAttachment
    function = dprw32->DPRSetConfig
    function = dprw32->DPRDecryptLogin
    function = dprw32->DPRDefaultLogin
    function = dprw32->DPRCheckLogin
    function = dprw32->DPRGetWipFormset
    function = dprw32->DPRPrint

; Find records and display them
< ReqType:WFD >
    function = atcw32->ATCLogTransaction
    function = atcw32->ATCLoadAttachment
    function = atcw32->ATCUnloadAttachment
    function = dprw32->DPRSetConfig
    function = dprw32->DPRDecryptLogin
    function = dprw32->DPRDefaultLogin
    function = dprw32->DPRCheckLogin
;   function = dprw32->WipTableParams ;Used only with JSP to
customize display
    function = dprw32->DPRFindWipRecords
```

Chapter 5

Using Docucorp Publishing Services

Docucorp Publishing Services (DPS) for print and archive are a set of objects you can use to interface VB, C#, or Java with Documaker to execute print or archive through Docupresentment (IDS). You can run IDS on a local or remote system.

Only one transaction can be processed per API invocation. The return file types can be PCL, PDF, or XML. For print requests, the input file (attachment) contains data representing a single transaction. For archive retrieval requests, the first transaction that matches the search criteria is returned.

Docucorp Publishing Services (DPS) consists of:

- An action object. (DPSPrint, DPSArchive)

  This object passes a group of variables (properties) that define input parameters for performing the specific action.

- An interface object. (DPSIDS)

  This object parses or constructs the request variables based on the input object properties and interfaces with IDS to send the request variables and receive the result variables.

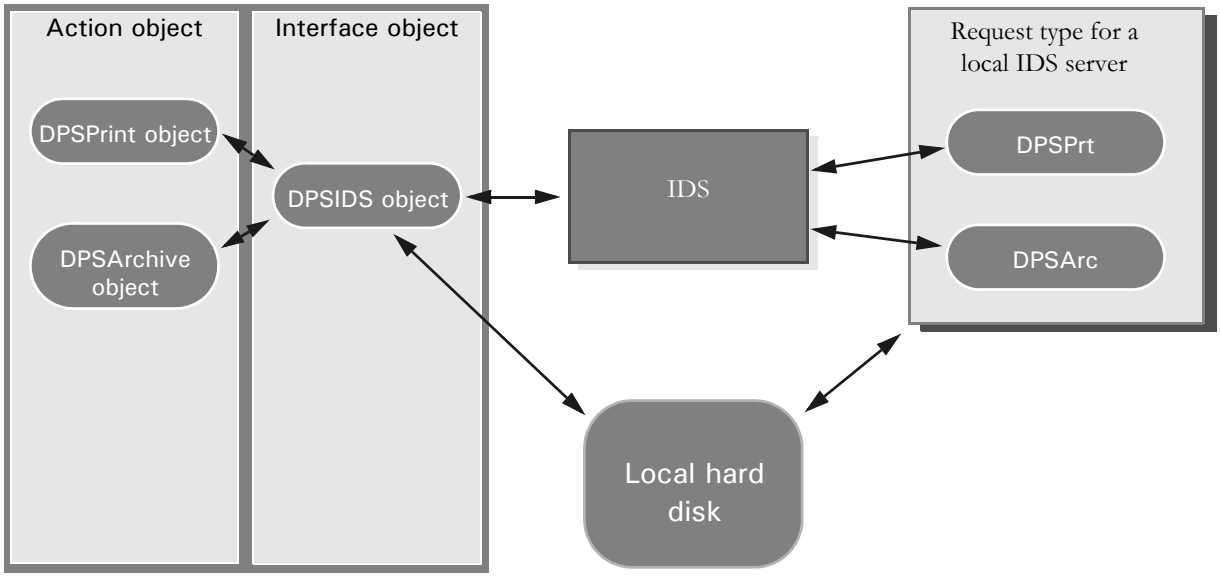
- IDS

  IDS performs the task based on the request. There are two groups of the requests: remote and local. If IDS is local, it is running on the same machine as the client and shares the same physical hard drive — so it is not necessary to send the output file back to the client since it is on the same machine. If IDS is remote, the output file needs to be sent from the machine that is running IDS to the client machine.

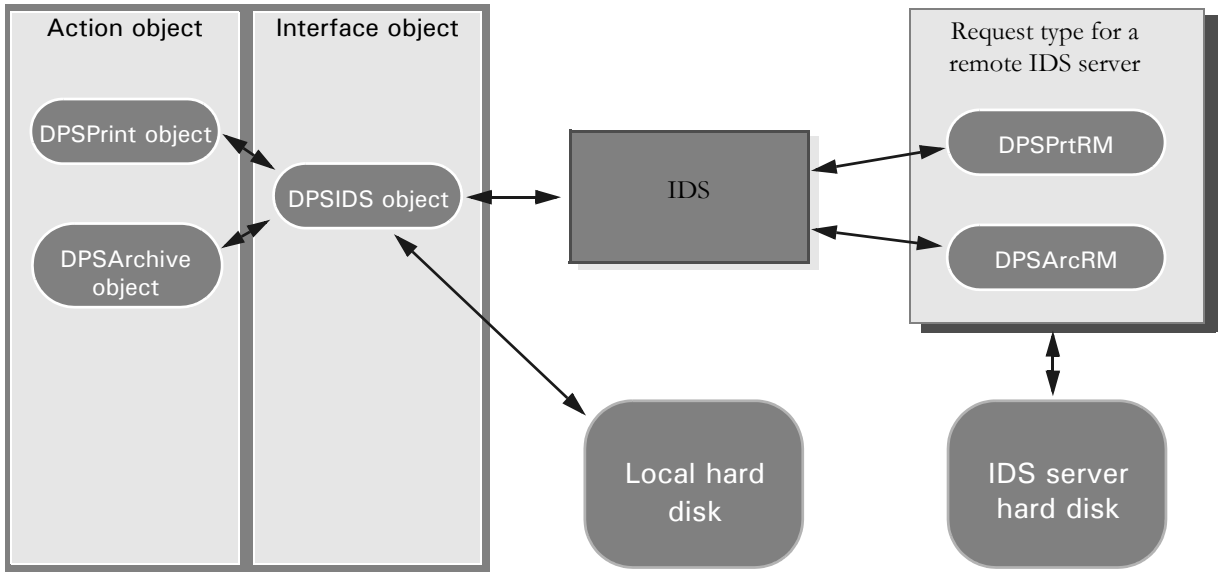
Here is a summary of the DLL and class files for DPS:

Required By	File name	Description
IDS	IDSRules.jar	com/docucorp/ids/rules/dps
VB API	DPSClient.dll	DPSPrint, DPSArchive, DPSIDS objects.
C# API	Docucorp.DPS.dll Docucorp.IDS.dll	DPSPrint, DPSArchive, DPSIDS objects.
Java API	dps.jar	DPSPrint, DPSArchive, DPSIDS objects.

The following illustration shows how this works on a local IDS setup, with IDS running on the same machine:



This illustration shows how it works with IDS running on a different machine:



DPS Object Properties

The following tables describe the properties of these objects:

- DPSPrint
- DPSArchive

DPSPrint object

The DPSPrint object has these properties:

	Property Name	I/O	Type	Description
VB	ApplicationName	I/O	String	(Optional) Use to pass application-specific data to the API.
C#	ApplicationName	I/O		
Java	setapplicationName getapplicationName	Input Output		
VB	ConfigurationName	I/O	String	Use to set the DAP configuration.
C#	ConfigurationName	I/O		
Java	setconfigurationName getconfigurationName	Input Output		
VB	DestinationName	I/O	String	(Optional) Use to pass application-specific data to the API.
C#	DestinationName	I/O		
Java	setdestinationName getdestinationName	Input Output		
VB	EffectiveDate[1]	I/O	String	(Optional) Specifies the effective date of this transaction in YYYYMMDD format.
C#	EffectiveDate[1]	I/O		
Java	seteffectiveDate[1] geteffectiveDate[1]	Input Output		
VB	FormDescription	I/O	String	(Optional) Specifies the form description to be processed for this transaction. Leave this property blank to tell the system to process all requested forms. You can pass multiple form descriptions by entering the form names separated by commas.
C#	FormDescription	I/O		
Java	setformDescription getformDescription	Input Output		

[1] Not yet implemented.

	Property Name	I/O	Type	Description
VB	FormName	I/O	String	(Optional) Specifies the form name to be processed for this transaction. Leave this property blank to tell the system to process all requested forms. You can pass multiple forms by entering the form names separated by commas.
C#	FormName	I/O		
Java	setformName getformName	Input Output		
VB	InputFile	I/O	String	Specifies the name of the attached input data file, which is usually an extract data file for the transaction, such as an XML extract file or an import file name.
C#	InputFile	I/O		
Java	setinputFile getinputFile	Input Output		
VB	Key1, Key2, Key3, and KeyID	I/O	String	(Optional) Specifies transactional keys or application-specific values.
C#	Key1, Key2, Key3, and KeyID	I/O		
Java	setkey1, setkey2, setkey3, and setkeyID getkey1, getkey2, getkey3, and getkeyID	Input Output		
VB	OutputFile	I/O	String	(Optional) Specifies a string that contains the name of the returned output file. If the buffer contains no input value (blank or empty) the Documaker system generates this output file name. Use the OutputFile property to get the generated file name.
C#	OutputFile	I/O		
Java	setoutputFile getoutputFile	Input Output		
VB	OutputPath	I/O	String	(Optional) Specifies the desired location of the returned output files. If you leave this property blank, the system uses the current directory from which the client was executed.
C#	OutputPath	I/O		
Java	setoutputPath getoutputPath	Input Output		
VB	Password	I/O	String	(Optional) Specifies the password if one is required by the request.
C#	Password	I/O		
Java	setpassword getpassword	Input Output		

[1] Not yet implemented.

	Property Name	I/O	Type	Description
VB	PrinterType	I/O	String	(Optional) Specifies the type of output, such as PCL, PDF, or XML. Future versions may support additional output types.
C#	PrinterType	I/O		
Java	setprinterType getprinterType	Input Output		
VB	Priority[1]	I/O	Priorities DPS_IMMEDIATE DPS_DEFERRED	Specifies immediate processing, deferred (batch) processing of the transaction, or end-of-day/end-of-period transaction. The default is DPS_IMMEDIATE.
C#	Priority[1]	I/O		
Java	setpriority[1] getpriority[1]	Input Output		
VB	RecipientName	I/O	String	(Optional) Specifies the recipient name to be processed for this transaction. Leave this property blank if you do not want recipient filtering performed. You can specify a list of recipients, using commas to separate the recipient names.
C#	RecipientName	I/O		
Java	setrecipientName getrecipientName	Input Output		
VB	ReturnCode	I/O	String	Set to DPS0000 if the transaction was processed successfully. Otherwise, it will contain an error code set by the DPS interface. This error code can be an error returned from the Documaker system or Docupresentment or a failure of the DPS API.
C#	ReturnCode	I/O		
Java	setreturnCode getreturnCode	Input Output		
VB	RunDate[1]	I/O	String	(Optional) Specifies the run date of this transaction in YYYYMMDD format.
C#	RunDate[1]	I/O		
Java	setrunDate[1] getrunDate[1]	Input Output		
VB	SourceName	I/O	String	(Optional) Use to pass application-specific data to the API.
C#	SourceName	I/O		
Java	setsourceName getsourceName	Input Output		

[1] Not yet implemented.

	Property Name	I/O	Type	Description
VB	TrnCode[1]	I/O	String	(Optional) Specifies a transaction code or other application-specific value.
C#	TrnCode[1]	I/O		
Java	settrnCode[1] gettrnCode[1]	Input Output		
VB	UserID	I/O	String	(Optional) Specifies a user ID if one is required by the request.
C#	UserID	I/O		
Java	setuserID getuserID	Input Output		
VB	WaitForResult	I/O	Boolean	Specifies whether the invoking application expects to receive results. The default is True.
C#	WaitForResult	I/O		
Java	setwaitForResult getwaitForResult	Input Output		

[1] Not yet implemented.

DPSArchive object

The DPSArchive object has these properties:

	Property Name	I/O	Type	Description
VB	ConfigurationName	I/O	String	Use to set the DAP configuration.
C#	ConfigurationName	I/O		
Java	setconfigurationName getconfigurationName	Input Output		
VB	FormDescription	I/O	String	(Optional) Specifies the form description to be processed for this transaction. Leave this property blank to tell the system to process all requested forms. You can pass multiple form descriptions using commas to separate the form names.
C#	FormDescription	I/O		
Java	setformDescription getformDescription	Input Output		
VB	FormKey1	I/O	String	(Optional) Specifies the form set filtering Key1.
C#	FormKey1	I/O		
Java	setformKey1 getformKey1	Input Output		

[1] Not yet implemented.

	Property Name	I/O	Type	Description
VB	FormKey2	I/O	String	(Optional) Specifies the form set filtering Key2.
C#	FormKey2	I/O		
Java	setformKey2 getformKey2	Input Output		
VB	FormName	I/O	String	(Optional) Specifies the forms to be processed for this transaction. Leave this property blank to tell the system to process all requested forms. You can pass multiple forms using commas to separate the form names.
C#	FormName	I/O		
Java	setformName getformName	Input Output		
VB	Key1, Key2, Key3, and KeyID	I/O	String	(Optional) Specifies the columns in application index file you want to use in the query
C#	Key1, Key2, Key3, and KeyID	I/O		
Java	setkey1, setkey2, setkey3, and setkeyID getkey1, getkey2, getkey3, and getkeyID	Input Output		
VB	OutputFile	I/O	String	(Optional) Specifies a string that contains the name of the returned output file. If the buffer contains no input value (blank or empty) the Documaker system generates this output file name. Use the outputFile property to get the generated file name.
C#	OutputFile	I/O		
Java	setoutputFile getoutputFile	Input Output		
VB	OutputPath	I/O	String	(Optional) Specifies the desired location of the returned output files. If you leave this property blank, the Docucorp publishing system (such as Documaker) uses its own INI options to determine the locations of the returned output files and the caller will have to know to retrieve the files from that location.
C#	OutputPath	I/O		
Java	setoutputPath getoutputPath	Input Output		
VB	Password	I/O	String	Specify password required to retrieve from the database.
C#	Password	I/O		
Java	setpassword getpassword	Input Output		

[1] Not yet implemented.

	Property Name	I/O	Type	Description
VB	PrinterType	I/O	String	(Optional) Specifies the type of output, such as PDF or XML. Future versions may support additional output types.
C#	PrinterType	I/O		
Java	setprinterType getprinterType	Input Output		
VB	RecipientName[1]	I/O	String	(Optional) Specifies the recipients to be processed for this transaction. Leave this property blank if you do not want recipient filtering performed. You can specify a list of recipients, using semicolons to separate the recipient names.
C#	RecipientName[1]	I/O		
Java	setrecipientName[1] getrecipientName[1]	Input Output		
VB	ReturnCode	I/O	String	Set to DPS0000 if transaction was processed successfully. Otherwise it contains an error code set by the DPS interface. This error code can be an error returned from the Documaker or Docupresentment system or a failure of the DPS API.
C#	ReturnCode	I/O		
Java	setreturnCode getreturnCode	Input Output		
VB	UserID	I/O	String	Specifies the user ID required to access the database.
C#	UserID	I/O		
Java	setuserID getuserID	Input Output		
VB	WaitForResult	I/O	Boolean	Specifies if the invoking application expects to receive results. The default is True
C#	WaitForResult	I/O		
Java	setwaitForResult getwaitForResult	Input Output		

[1] Not yet implemented.

DPSIDS object

The DPSPrint object has these methods:

	Methods	Description
VB	Send(actionObject as Variant)	Call to parse an action object's properties to request variables and send the request variables to the IDS Server to perform the function specified by the requestType.
C#	Send(DpsPrint dpsPrintObject) Send(DpsArchive dpsArchiveObject)	
Java	send(DPSPrint oDPSPrint) send(DPSArchive oDPSArchive)	

Setting Default Parameters

Use the DPSINI.XML file to set default parameters. The system looks for the file in the current directory. If it is not found, it looks in the Windows system directory, such as c:\winnt\system32.

Here is an example of the DPSINI.XML file:

```
<?xml version='1.0'?>
<DPS>
  <DPSIDS>
    <RemoteIDS>Yes</RemoteIDS>
    <PathSeparator>\</PathSeparator>
  </DPSIDS>
  <DPSPrint>
    <RequestType>DPSPRT</RequestType>
    <RequestTypeRM>DPSPRTRM</RequestTypeRM>
    <OutputAttachName>DPSPRTOUTPUT</OutputAttachName>
    <InputAttachName>DPSPRTINPUT</InputAttachName>
  </DPSPrint>
  <DPSArchive>
    <RequestType>DPSARC</RequestType>
    <RequestTypeRM>DPSARCRM</RequestTypeRM>
    <PartialMatch>Yes</PartialMatch>
    <CaseSensitivity>No</CaseSensitivity>
    <MaxRecords>1</MaxRecords>
    <OutputAttachName>DPSARCOUTPUT</OutputAttachName>
  </DPSArchive>
</DPS>
```

DPSIDS section

Parameter	Description
RemoteIDS	This parameter determines the location of the IDS Server, local or remote.
PathSeparator	

DPSPrint section

Parameter	Description
RequestType	This is the request type when RemoteIDS is set to No. The default is DPSPRT.
RequestTypeRM	This is the request type when RemoteIDS is set to Yes. The default is DPSPRTRM.
OutputAttachname	This is the AttachName used by the IDS Server to send a file to the client when RemoteIDS is set to Yes. The default is DPSPRTOUTPUT.
InputAttachName	This is the AttachName used by the client to send an input extract file to the IDS Server when RemoteIDS is set to Yes. The default is DPSPRTINPUT.

DPSArchive section

Parameter	Description
RequestType	The request type when RemoteIDS is set to No. The default is DPSARC.

Parameter	Description
RequestTypeRM	The request type when RemoteIDS is set to No. The default is DPSARCRM.
OutputAttachname	The AttachName used by the IDS Server to send a file to the client when RemoteIDS is set to Yes. The default is DPSARCOUTPUT.
PartialMatch	The search condition uses a partial match.
CaseSensitivity	If set to Yes, the search is case sensitive. The default is No
Maxrecords	The maximum number of records to return. It must be set to one (1).

SAMPLE VB CODE

Here are examples of Visual Basic code for print and archive:

Print

```
Private Sub CmdPrint_Click()
   Dim oDPSVar As New DPSPrint
   Dim oDPSIDS As New DPSIDS
   oDPSVar.inputFile = "D:\MRL\TEST\EXTRACTS\ExtrFile.dat"
   oDPSVar.configurationName = "DPS"
   oDPSVar.outputFile = "PrintOutput"
   oDPSVar.outputPath = "D:\MRL\TEST\PrintFile.PCL"
   oDPSVar.printerType = "PCL"
   oDPSIDS.send oDPSVar
   if oDPSVar.ReturnCode = "DPS0000" Then
     FinalOutput.Text = oDPSVar.outputPath + oDPSVar.outputFile
   End If
End Sub
```

Archive

```
Private Sub CmdArchive_Click()
   Dim oDPSVar As New DPSArchive
   Dim oDPSIDS As New DPSIDS
   FinalOutputA.Text = ""
   oDPSVar.configurationName = "DPS"
   oDPSVar.userID = "UserID"
   oDPSVar.password = "Password"
   oDPSVar.outputFile = FldOutputFileA.Text
   oDPSVar.outputPath = FldOutputPathA.Text
   oDPSVar.key1 = "SAMPCO"
   oDPSVar.printerType = "PDF"
   oDPSIDS.send oDPSVar
   if oDPSVar.ReturnCode = "DPS0000" Then
     FinalOutputA.Text = oDPSVar.outputPath + oDPSVar.outputFile
   End If
End Sub
```

NOTE: The DPSClient.dll file must be referenced before you can use the DPSPrint, DPSArchive, and DPSIDS objects.

SAMPLE C CODE

Here is some sample C# code:

```
using System;
using Docucorp.DPS;
.
.
.
private void printButton_Click(object sender, System.EventArgs e)
{
     DpsPrint dpsVarObject = new DpsPrint();
      DpsIds dpsIdsObject = new DpsIds();

     dpsVarObject.InputFile = inputFile.Text;
     dpsVarObject.ConfigurationName = config.Text;
     dpsVarObject.OutputFile = outputFile.Text;
     dpsVarObject.OutputPath = outputPath.Text;
     dpsVarObject.PrinterType = printerType.Text;
      dpsVarObject.FormName = formName.Text;
    dpsVarObject.FormDescription = formDescription.Text;
     dpsVarObject.RecipientName = recipientName.Text;
       try
         {
            dpsIdsObject.Send(dpsVarObject);
                 finalOutput.Text = dpsVarObject.OutputPath +
dpsVarObject.OutputFile;
       }
      catch (Exception ex)
        {
          Console.Out.WriteLine(ex.ToString());
       }
}

private void archiveButton_Click(object sender, System.EventArgs e)
{
     DpsArchive dspVarObject = new DpsArchive();
      DpsIds dpsIdsObject = new DpsIds();

       finalOutputA.Text = "";
      dspVarObject.WaitForResult = true;
      dspVarObject.UserID = userID.Text;
      dspVarObject.Password = password.Text;
      dspVarObject.ConfigurationName = "DPS";
     dspVarObject.OutputFile = outputFileA.Text;
     dspVarObject.OutputPath = outputPathA.Text;
      dspVarObject.Key1 = "SAMPCO";
     dspVarObject.PrinterType =  printerTypeA.Text;
      dspVarObject.FormName = formNameA.Text;
       try
         {
            dpsIdsObject.Send(dpsVarObject);
                 finalOutputA.Text = dpsVarObject.OutputPath +
dpsVarObject.OutputFile;
       }
      catch (Exception ex)
        {
          Console.Out.WriteLine(ex.ToString());
```

```
            }
    }
```

NOTE: You must install the Docucorp.IDS.dll file in GAC and the Docucorp.DPS.dll file must be referenced before you can use the DPSPrint, DPSArchive, and DPSIDS objects.

Sample Java Code

Here is some sample Java code:

```
import com.docucorp.dps.*;
.
.
.
  void BtnPrint_actionPerformed(ActionEvent e) {
    DPSPrint oDPSVar = new DPSPrint();
    try {
     DPSIDS oDPSIDS = new DPSIDS();
     oDPSVar.setinputFile(FldInputFile.getText());
     oDPSVar.setconfigurationName(FldConfig.getText());
     oDPSVar.setoutputFile(FldOutputFile.getText());
     oDPSVar.setoutputPath(FldOutputPath.getText());
     oDPSVar.setprinterType(FldPrinterType.getText());
     oDPSVar.setformName(FldFormName.getText());
     oDPSVar.setformDescription(FldFormDescription.getText());
     oDPSVar.setrecipientName(FldRecipientName.getText());
     oDPSIDS.send(oDPSVar);
     FldFinalOutput.setText(oDPSVar.getoutputPath() +
oDPSVar.getoutputFile());
     FldReturnCode.setText(oDPSVar.getreturnCode());
    } catch (DPSJException ex) {
      FldReturnCodeA.setText(oDPSVar.getreturnCode());
      ex.printStackTrace();
    } catch (DSIJException ex) {
      FldReturnCodeA.setText(oDPSVar.getreturnCode());
      ex.printStackTrace();
    } catch (Exception ex) {
      FldReturnCodeA.setText(oDPSVar.getreturnCode());
      ex.printStackTrace();
    }
  }

  void BtnArchive_actionPerformed(ActionEvent e) {
    DPSArchive oDPSVar = new DPSArchive();
    try {
      DPSIDS oDPSIDS = new DPSIDS();
      oDPSVar.setuserID(FldUserID.getText());
      oDPSVar.setpassword(FldPassword.getText());
      oDPSVar.setconfigurationName(FldConfigA.getText());
      oDPSVar.setoutputFile(FldOutputFileA.getText());
      oDPSVar.setoutputPath(FldOutputPathA.getText());
      oDPSVar.setformName(FldFormNameA.getText());
      oDPSVar.setprinterType(FldPrinterTypeA.getText());
      oDPSIDS.send(oDPSVar);
      FldFinalOutputA.setText(oDPSVar.getoutputPath() +
oDPSVar.getoutputFile());
      FldReturnCodeA.setText(oDPSVar.getreturnCode());
    } catch (DPSJException ex) {
      FldReturnCodeA.setText(oDPSVar.getreturnCode());
      ex.printStackTrace();
    } catch (DSIJException ex) {
      FldReturnCodeA.setText(oDPSVar.getreturnCode());
      ex.printStackTrace();
    } catch (Exception ex) {
```

```
            FldReturnCodeA.setText(oDPSVar.getreturnCode());
            ex.printStackTrace();
        }
    }
```

Setting Up IDS

Add the following request types to the DOCSERV.INI file to set up IDS:

```
[ ReqType:DPSARC ]
  function = atcw32->ATCLogTransaction
  function = atcw32->ATCLoadAttachment
  function = dprw32->DPRLocateOneRecord
  function = dprw32->DPRInitLby
  function = atcw32->ATCUnloadAttachment
  function = dsijrule->JavaRunRule,;com/docucorp/ids/rules/
dps;DPS;global;duplicateAttach;,RV,REMOTEPRINTFILE,O,ARCOUTPUTFILE,
O
  function = dsijrule->JavaRunRule,;com/docucorp/ids/rules/
dps;DPS;global;duplicateAttach;,RV,SENDBACKPAGE,O,ARCOUTPUTFILE,O
  function = dprw32->DPRRetrieveFormset
  function = dprw32->DPRFilterFormsetForms
  function = dprw32->DPRPrint
  function = dprw32->DPRProcessTemplates
[ ReqType:DPSARCRM ]
  function = atcw32->ATCLogTransaction
  function = atcw32->ATCLoadAttachment
  function = dprw32->DPRSetConfig
  function = dprw32->DPRLocateOneRecord
  function = dprw32->DPRInitLby
  function = atcw32->ATCUnloadAttachment
  function = dprw32->DPRRetrieveFormset
  function = dprw32->DPRFilterFormsetForms
  function = atcw32->ATCSendFile,DPSARCOUTPUT,OUTPUTFILE,Binary
  function = dsijrule->JavaRunRule,;com/docucorp/ids/rules/
dps;DPS;global;duplicateAttach;,RV,OUTPUTFILE,O,ARCOUTPUTFILE,O
  function = dsijrule->JavaRunRule,;com/docucorp/ids/rules/
dps;DPS;global;duplicateAttach;,RV,REMOTEPRINTFILE,O,OUTPUTFILE,O
  function = dsijrule->JavaRunRule,;com/docucorp/ids/rules/
dps;DPS;global;duplicateAttach;,RV,SENDBACKPAGE,O,OUTPUTFILE,O
  function = dprw32->DPRPrint
  function = dprw32->DPRProcessTemplates
[ ReqType:DPSPRT ]
  function = atcw32->ATCLogTransaction
  function = atcw32->ATCLoadAttachment
  function = atcw32->ATCUnloadAttachment
  function = dprw32->DPRSetConfig
  function = dsijrule->JavaRunRule,;com/docucorp/ids/rules/
dps;DPS;global;duplicateAttach;,RV,PRINTER1,O,PRINTOUTPUTFILE,O
  function = rpdw32->RPDCheckRPRun
  function = rpdw32->RPDCreateJob
  function = rpdw32->RPDProcessJob
[ ReqType:DPSPRTRM ]
  function = atcw32->ATCLogTransaction
  function = atcw32->ATCLoadAttachment
  function = atcw32->ATCUnloadAttachment
  function = atcw32-
>ATCReceiveFile,DPSPRTINPUT,EXTRFILE,d:\temp\*.dat,KEEP
  function = dprw32->DPRSetConfig
  function = atcw32->ATCSendFile,DPSPRTOUTPUT,PRINTER1,Binary
  function = dsijrule->JavaRunRule,;com/docucorp/ids/rules/
dps;DPS;global;duplicateAttach;,RV,PRINTER1,O,PRINTOUTPUTFILE,O
  function = dsijrule->JavaRunRule,;com/docucorp/ids/rules/
dps;DPS;global;duplicateAttach;,RV,PRINTER1,O,OUTPUTFILE,O
  function = rpdw32->RPDCheckRPRun
```

```
  function = rpdw32->RPDCreateJob
  function = rpdw32->RPDProcessJob
```

Add these control groups and options to the DAP.INI file:

```
< Config:DPS >
    INIFile = DPS.INI
< Configurations >
    Config  = DPS
```

Here is a sample DPS.INI file:

```
< MasterResource >
    XRFFile  = rel102
    DefLib   = D:\MRL\DEMO\MstrRes\Def\
    FormLib  = D:\MRL\DEMO\MstrRes\FAP\
    LbyLib   = D:\MRL\DEMO\MstrRes\FAP\
    FontLib  = D:\AGFA\
    FormDef  = form.dat
< RPDRunRP >
    Executable = d:\rel111\rps100\w32bin\GENDAW32.EXE
    Directory  = D:\MRL\DEMO\MstrRes
;   UserINI    = D:\MRL\DEMO\RunBatch\fsiuser.s.ini
    UserINI    = D:\MRL\DEMO\RunBatch\FSIUSER.PCL.INI
    Debug      = Yes
< RPDCheckRPRun >
    Debug      = Yes
< IDSServer >
    BaseLocation = http://10.1.10.209/doc-data/
    PrintPath    = d:\MRL\DEMO\PrintFiles
    GENSemaphoreName = GenData
    RPDSemaphoreName = RPDRunRP
< Debug >
    RULServerJobProc = Yes
    RPDProcessJob = Yes
< ARCRet >
;   path to CAR files
    CARPath = D:\MRL\DEMO\ARC\
    CARFile = ARCHIVE
;   full file name for application index
    APPIDX = D:\MRL\DEMO\ARC\APPIDX
;   full file name for temporary index
    TempIDX=D:\MRL\DEMO\ARC\TEMP
;   full file name for CAR files catalog
    Catalog = D:\MRL\DEMO\ARC\CATALOG
    APPIDXDFD  = D:\MRL\DEMO\mstrres\def\appidx.dfd
< Control >
    XrfExt     = .fxr
    ImageEXT   = .fap
    DateFormat = 24%
< PrtType:PDF >
    LanguageLevel= Level2
    Module       = PDFOS2
    PrintFunc    = PDFPrint
    Linearize    = No
```

Setting Up Documaker

Set up Documaker to run in single-step mode. Please note in the following sample AFGJOB.JDT file that the RULServerJobProc rule is required to run in the IDS environment and the ServerFilterFormRecipient rule is required for DPS to run properly.

Here is a sample AFGJOB.JDT file:

```
/* This base (this implementation) uses these rules. */
<Base Rules>
;RULServerJobProc;1;Always the first job level rule;
/*;RULStandardJobProc;;Always the first job level rule;
;JobInit1;;;
;BuildMasterFormList;1;4;
;InitPrint;;required to execute gendata/genprint in single step;

/* Every form set in this base uses these rules. */
<Base Form Set Rules>
;NoGenTrnTransactionProc;;required to combine gentrn/gendata into
single step;
;BuildFormList;;;
;LoadRcpTbl;;;
;ServerFilterFormRecipient;;;
;RunSetRcpTbl;;;
;PrintFormset;;required to combine gendata/genprint into single
step;
;WriteOutput;;;required to combine gentrn/gendata into single step;
;WriteNaFile;;;required to combine gentrn/gendata into single step;
;BatchingByPageCountINI;;;
;ProcessQueue;;PostPaginationQueue;
;PaginateAndPropogate;;FooterMode(2) Debug;

/* Every image in this base uses these rules. */
<Base Image Rules>
;RULStandardImageProc;;Always the first image level rule;

/* Every field in this base uses these rules. */
<Base Field Rules>
;RULStandardFieldProc;;Always the first field level rule;
```

Chapter 6

Customizing iDocumaker, iPPS, and WIP Edit

This appendix describes how you can customize how iDocumaker, iPPS, and the WIP Edit plug-in work with IDS.

In this appendix, you will find the following topics:

- Setting Up a Favorites List for iDocumaker on page 274
- Attaching Files to Transactions as Forms on page 276
- Designating Read-Only Multi-Line Text Field Paragraphs on page 282
- Printing on Your Workstation Printer on page 283
- Preventing the Session from Expiring on page 284
- Passing WIP Record IDs to the MergeWIP Rule on page 285
- Automatically Updating iDocumaker on page 286
- Using the WIP Edit Plug-in on page 292

Setting Up a Favorites List for iDocumaker

You can create a *favorites list* — a list of frequently used forms — for use in iDocumaker. This increases your ability to get MRL information from IDS via XML and is similar to Documaker Workstation feature that lets you set up personal forms lists.

For example, if your company has a large number of forms, each user can set up a favorites list to more quickly find the forms he or she typically works with. Keep in mind that users can still select any available form, you are not limited to just those forms on your favorites list.

To understand this feature you need to understand how iDocumaker uses the i_GetMRLResource request type to get a list of the groups and forms an MRL supports.

First, iDocumaker requests a list of groups by running the i_GetMRLResource request without submitting any XML. A list of groups is returned to iDocumaker in an XML attachment called DOCUMENTSTREAM.

If this feature is enabled, one of the groups is identified as the favorites group. Either the user or iDocumaker can then select one or more of these groups to get a forms list.

You get a forms list by sending the list of desired groups in the XMLIMPORT XML attachment to the i_GetMRLResource rule. The i_GetMRLResource request type returns an XML attachment that contains forms, form descriptions, and recipient data for all of the requested groups, including the favorites group if you enabled favorites and one of the submitted groups contains the attribute FAVORITES=TRUE.

You can store one favorites list per configuration. The list is stored in this location:

```
Config\UserID\profile.xml
```

For example, if the user ORACLE has a favorites list for the configuration SAMPCO, the favorites list will be stored in the following location:

```
SAMPCO\ORACLE\profile.xml
```

Here is an example of the XML file that contains the favorites list:

```
<DOCSET>
  <GROUP NAME1="FAVORITES" NAME2="FAVORITES" NAME3="">
     <FORM NAME="FIL 1010 04 92"/>
     <FORM NAME="FIM 0100 11 92"/>
     <FORM NAME="FCG 0010 11 92"/>
     <FORM NAME="Barcode Samples"/>
     <FORM NAME="DAL Locale"/>
     <FORM NAME="Auto Increment Names"/>
      <FORM NAME="A128">
     </FORM>
  </GROUP>
</DOCSET>
```

You can use these INI options in your MRL INI file to control the favorites list:

```
< Favorites >
    Enabled = Yes
    Path    = z:\sharedir
    Name1   = Favorites
    Name2   = Favorites
```

Option	Description
Enabled	Enter Yes to turn on the use of favorites. The default is No.
Path	Enter the name of the path into which you want the favorites list saved. For instance, if your user ID is *Oracle* and you enter... z:\sharedir The favorites list will be stored in this directory: z:\sharedir\oracle\profile.xml If you have IDS installed on multiple PCs, set the Path option to point to the same location. If you only have IDS installed on a single PC, you can omit this option.
Name1	Enter the name of the first favorites group.
Name2	Enter the name of the second favorites group.

NOTE: When you select a form from the favorites list, the Key1/Key2 in WIP is set to whatever is in the Name options in your Favorites control group (*Favorites* in the preceding example).

Attaching Files to Transactions as Forms

Using the Documaker Bridge, you can now attach external files as forms to Documaker transactions. These external files can be in the following formats: TIFF, JPG, PDF, and other bitmap formats supported by Documaker.

You can also attach RTF files. The RTF import is limited to the same level of support here as it has in other places in the system. For instance, if something will not work in Studio, it will not work here either.

When you attach one of these types of files, it becomes an embedded bitmap in a form in the Documaker transaction. The attached form has an option to indicate it is an attachment (the letter *A* in form options).

NOTE: This feature was implemented for use with iDocumaker. The Documaker Bridge rules DPRUpdateFormsetFromXML and DPRLoadImportFile were enhanced to support attachment forms.

You can attach a file by:

- Placing it on disk and specifying its name and type in IDS attachment variables.
- Sending the file to IDS in a message.
- Placing the file on a disk accessible to the Documaker Bridge.
- Placing the file in a Documanage repository.

In all cases, the information needed to find the file is located in the form metadata. Special metadata tag names are reserved for each case.

Specifying the File Name and Type in IDS Attachment Variables

Use these tags in the form's metadata specify how to locate the file name.

Tag	Description
DPR_ATTACHVARNAME	The name of the DSI attachment variable where the file name is stored.
DPR_FILETYPE	(Optional) The file type. The file type is determined by the program by looking at this value. If missing, the file extension is checked. If the extension is missing, the default is TIFF.
DPR_FILETYPEVAR	(Optional) The name of the DSI attachment variable with the file type. The file type is determined by the program by looking at this value. If missing, the file extension is checked. If the extension is missing, the default is TIFF. If the DPR_FILETYPE variable is present, this variable is ignored.

Here is an example fragment of an XML import file with this information. The file name is located in a DSI variable named DPRFILE and its type is in the DSI variable DPRTYPE.

```
<FORM NAME="Test form name" OPTIONS="RA">
```

```
<INFO NAME="DPR_ATTACHVARNAME">DPRFILE</INFO>
<INFO NAME="DPR_FILETYPEVAR">DPRTYPE</INFO>
<DESCRIPTION>Test description of the form</DESCRIPTION>
<RECIPIENT NAME="AGENT" COPYCOUNT="1"/>
<RECIPIENT NAME="HOME OFFICE" COPYCOUNT="1"/>
</FORM>
```

Sending the File to IDS in a Message

The following tags in form metadata specify how to locate the file data.

Tag	Description
DPR_ATTACHNAME	The name of the DSI attachment in which the file was sent, such as via the SendFile API.
DPR_FILETYPE	(Optional) The file type. The file type is determined by the program by looking at this value. If missing, the file extension is checked. If the extension is missing, the default is TIFF.
DPR_FILETYPEVAR	(Optional) The name of the DSI attachment variable with the file type (optional). The file type is determined by the program by looking at this value. If missing, the file extension is checked. If the extension is missing, the default is TIFF. If the DPR_FILETYPE variable is present, this variable is ignored.

At least one of the file type values is required even though both are listed as optional.

Here is an example fragment of an XML import file with this information. The file is sent to IDS inside the message and the name of the attachment used to send it is SENTFILE. The type of file is in the DSI variable DPRTYPE.

```
<FORM NAME="Test with DSI message" OPTIONS="RA">
<INFO NAME="DPR_ATTACHNAME">SENTFILE</INFO>
<INFO NAME="DPR_FILETYPE">TIF</INFO>
<DESCRIPTION>Test description of the form</DESCRIPTION>
<RECIPIENT NAME="AGENT" COPYCOUNT="1"/>
<RECIPIENT NAME="HOME OFFICE" COPYCOUNT="1"/>
</FORM>
```

Storing the File on a Disk Accessible to Documaker Bridge

Use these tags in the form's metadata to tell Documaker Bridge how to locate the file.

Tag	Description
DPR_FILENAME	The name of the file.
DPR_FILETYPE	(Optional) The file type. The file type is determined by the program by looking at this value. If missing, the file extension is checked. If the extension is missing, the default is TIFF.

Tag	Description
DPR_FILETYPEVAR	Optional) The name of the DSI attachment variable with the file type. The file type is determined by the program by looking at this value. If missing, the file extension is checked. If the extension is missing, the default is TIFF. If the DPR_FILETYPE variable is present, this variable is ignored.

Here is an example fragment of an XML import file with this information.

```
<FORM NAME="Test with filename" OPTIONS="RA">
<INFO NAME="DPR_FILENAME">c:\docs\Image_0001.jpg</INFO>
<INFO NAME="DPR_FILETYPE">JPG</INFO>
<DESCRIPTION>Test description of the form</DESCRIPTION>
<RECIPIENT NAME="AGENT" COPYCOUNT="1"/>
<RECIPIENT NAME="HOME OFFICE" COPYCOUNT="1"/>
</FORM>
```

NOTE: If you are using relative paths in the file name, the path has to be relative to the directory where Docupresentment is running.

Storing the File in a Documanage Repository

Include these tags in the form's metadata to specify how to locate the file:

Tag	Description
DMG_CABINET	The name of the Documanage cabinet.
DMG_DOCID	The value of the Documanage DOCID.
DMG_VERSION	The major version of the document.
DMG_REVISION	The minor version of the document.
DMG_VERS	The minor and major version of the document. The format is minor.major, such as 1.0 or 2.5. If this value is present, the values of DMG_VERSION and DMG_REVISION are ignored.

Here is an example fragment of an XML import file with this information.

```
<FORM NAME="Test with Documanage" OPTIONS="RA">
<INFO NAME="DMG_CABINET">DOCCDEMO</INFO>
<INFO NAME="DMG_DOCID">22401</INFO>
<INFO NAME="DMG_VERSION">1</INFO>
<INFO NAME="DMG_REVISION">0</INFO>
<DESCRIPTION>Test description of the form</DESCRIPTION>
<RECIPIENT NAME="AGENT" COPYCOUNT="1"/>
<RECIPIENT NAME="HOME OFFICE" COPYCOUNT="1"/>
</FORM>
```

Here is another example with the DOC_VERS value:

```
<FORM NAME="Test with Documanage" OPTIONS="RA">
```

```
<INFO NAME="DMG_CABINET">DOCCDEMO</INFO>
<INFO NAME="DMG_DOCID">22401</INFO>
<INFO NAME="DMG_VERS”>1.0</INFO>
<DESCRIPTION>Test description of the form</DESCRIPTION>
<RECIPIENT NAME="AGENT" COPYCOUNT="1"/>
<RECIPIENT NAME="HOME OFFICE" COPYCOUNT="1"/>
</FORM>
```

Note that to use the file in Documanage, the Documanage Bridge must be available on the same Docupresentment server. The Documaker Bridge executes these Documanage Bridge rules when it encounters the form with the metadata. No configuration changes are needed:

- DmgBrsCopyAttachment
- DmgBrsValidateSession
- DmgBrsCacheContentsFile

You do not have to specify the file type in this case, the Documanage document type is used instead.

Error Messages

These error messages can be produced by the DPR rules listed above if the attached form did not work or was specified incorrectly.

Message	Description
DPR0097	Attachment form <FORM> metadata specified the DSI attachment variable <VARIABLE> but this variable was not found. The file will not be loaded.
DPR0098	Attachment form <FORM> metadata specified the DSI file attachment with the delimiter <VARIABLE> but this file was not attached to the DSI message. The file will not be loaded.
DPR0099	Attachment form <FORM>metadata is missing the required value <INFO>. The file will not be loaded.
DPR0100	Failed to load the attached file specified by the attachment form <FORM>. File name <FILE> of type <TYPE>.
DPR0101	Failed to load the dynamic link library <LIBRARY>.
DPR0102	Cannot locate variable <VARIABLE> in the attachment list after executing the Documanage Bridge rules. Examine the Documanage Bridge errors.

Specifying Duplex Options for the Attached Form

When it contains multiple pages, the attached form might have to be printed in duplex mode. The duplex options in Documaker are specified on sections (images), so to provide the duplex information the form in XML must specify a section and section duplex options.

Your choices are:

- F – front
- B – back
- T – short bind

If there are no options or no section is specified, the rule assumes simplex mode. At the end of the options you must to specify *#1* to indicate it is a dummy image. Here is an example:

```
(OPTIONS="S#1")
```

The name of the section is ignored. Here are a few examples:

Start on back page bind example

```
<FORM NAME="Test with PDF filename" OPTIONS="RA">
<INFO NAME="DPR_FILENAME"> mytifftest.tif</INFO>
<DESCRIPTION>Test of TIFF form</DESCRIPTION>
<RECIPIENT NAME="AGENT" COPYCOUNT="1"/>
<RECIPIENT NAME="INSURED" COPYCOUNT="1"/>
<SHEET>
<PAGE>
<SECTION NAME="TESTSECTION" OPTIONS="B#1"/>
</PAGE>
</SHEET>
</FORM>
```

Long bind example

```
<FORM NAME="Test with PDF filename" OPTIONS="RA">
<INFO NAME="DPR_FILENAME"> mytifftest.tif</INFO>
<DESCRIPTION>Test of TIFF form</DESCRIPTION>
<RECIPIENT NAME="AGENT" COPYCOUNT="1"/>
<RECIPIENT NAME="INSURED" COPYCOUNT="1"/>
<SHEET>
<PAGE>
<SECTION NAME="TESTSECTION" OPTIONS="F#1"/>
</PAGE>
</SHEET>
</FORM>
```

Short bind example

```
<FORM NAME="Test with PDF filename" OPTIONS="RA">
<INFO NAME="DPR_FILENAME"> mytifftest.tif</INFO>
<DESCRIPTION>Test of TIFF form</DESCRIPTION>
<RECIPIENT NAME="AGENT" COPYCOUNT="1"/>
<RECIPIENT NAME="INSURED" COPYCOUNT="1"/>
<SHEET>
<PAGE>
<SECTION NAME="TESTSECTION" OPTIONS="T#1"/>
</PAGE>
</SHEET>
</FORM>
```

Debugging

Include this INI option in the DAP INI files to help you resolve any problems.

```
< Debug >
    DPRProcessFormsetAttachments = Yes
```

The default is No. If you enter Yes, the NA and POL files are unloaded with the names dprattach.dat and dprattach.pol. Here is an example of the log file (dprtrc.log) the system produces:

```
DPRProcessFormsetAttachments: DMG_CABINET=<DOCCDEMO> Form <Test with
Documanage>. Adding CABINET attachment variable

DPRProcessFormsetAttachments: DMG_DOCID=<22401> Form <Test with
Documanage>. Adding DOC_ID attachment variable
DPRProcessFormsetAttachments: DMG_VERSION=<1> Form <Test with
Documanage>. Adding DOC_MAJORVERSION attachment variable
DPRProcessFormsetAttachments: DMG_REVISION=<0> Form <Test with
Documanage>. Adding DOC_MINORVERSION attachment variable
DMG Rule DmgBrsCopyAttachment(DSI_MSGINIT) Time spent: 0.000
DMG Rule DmgBrsValidateSession(DSI_MSGINIT) Time spent: 0.000
DMG Rule DmgBrsCacheContentsFile(DSI_MSGINIT) Time spent: 0.000
DMG Rule DmgBrsCopyAttachment(DSI_MSGRUNF) Time spent: 0.078
DMG Rule DmgBrsValidateSession(DSI_MSGRUNF) Time spent: 0.109
DMG Rule DmgBrsCacheContentsFile(DSI_MSGRUNF) Time spent: 0.094
DPRProcessFormsetAttachments: found Documanage bridge attachment
variables CONTENTS_DECOMPRESSED_PATH=<cache\22401f0v1x0.tif> and
CONTENTS_DECOMPRESSED_TYPE=<TIF>
DMG Rule DmgBrsCacheContentsFile(DSI_MSGRUNR) Time spent: 0.016
DMG Rule DmgBrsValidateSession(DSI_MSGRUNR) Time spent: 0.000
DMG Rule DmgBrsCopyAttachment(DSI_MSGRUNR) Time spent: 0.000
DMG Rule DmgBrsCacheContentsFile(DSI_MSGTERM) Time spent: 0.000
DMG Rule DmgBrsValidateSession(DSI_MSGTERM) Time spent: 0.000
DMG Rule DmgBrsCopyAttachment(DSI_MSGTERM) Time spent: 0.000
```

Designating Read-Only Multi-Line Text Field Paragraphs

The following attributes are included in the XML export file on the <P> tag for read-only multi-line text field paragraphs:

```
contenteditable="false"
unselectable="on"
```

These attributes are used by iPPS and iDocumaker TERSUB functionality to prevent a user from selecting or modifying the paragraphs.

Here is an example:

```
<P contenteditable="false" unselectable="yes" />
```

Printing on Your Workstation Printer

When using iPPS or iDocumaker, you can send print files to workstation printers. The print files are created on the server, downloaded, and then printed on your workstation's printer.

The system displays the Printer window so you can select the printer you want to use or cancel the print job. The printer you select must support either PCL or PostScript.

The print files have a DPP file extension and will be in PCL or PostScript format. This DPP file is generated by IDS via a request from iPPS or iDocumaker. There are no changes on the client side (plug-in) you need to make.

NOTE: When you install or update iPPS or iDocumaker, the installation process creates the necessary file association.

Preventing the Session from Expiring

You can prevent the web server session from timing out when you are working on documents in iDocumaker. The time-out occurs if you leave the document open in iDocumaker for an interval longer than the time the web server allows a session to remain active. For instance, when a session times out, a save request will fail.

The session is kept current by telling iDocumaker to contact the web server when you change the current page. To do this, set this INI option in the CONFIG.INI file:

```
< INI2XML >
    RefreshScript = iwip18/test.htm
```

NOTE: The value of the RefreshScript option is specfic to your installation. The value shown above is only an example.

By default, iDocumaker will not contact the web server if it has done so in the last five minutes. You can change this interval using this INI option in the WIPEDIT.INI file.

```
< WIPEdit >
    RefreshSessionTime = 600
```

Specify the interval in seconds. The example above specifies an interval of 600 seconds, or 10 minutes.

Passing WIP Record IDs to the MergeWIP Rule

iDocumaker can designate a single WIP transaction to be processed by Documaker. IDS then passes the WIP record ID to Documaker so the MergeWIP rule will process that record.

This table describes how it works:

On the...	This happens
IDS side	The RPDCreateJob rule checks the WIPRECORDID input attachment variable and adds the XML element <WIPRECORDID> to the job ticket. Keep in mind that the WIPRECORDID input attachment variable is required when the RPD request is submitted. This requirement is in addition to the normal requirements for running Documaker as a subordinate process of IDS.
Documaker Server side	The ServerJobProc rule receives the job ticket and looks for the WIPRECORDID variable. If WIPRECORDID is found, the rule creates the WIPRECORDID GVM. The MergeWIP rule uses the WIPRECORDID GVM to retrieve the WIP record for batch processing by Documaker.

NOTE: This only affects Documaker when you are running Documaker via IDS.

For more information about the MergeWIP rule, see the Rules Reference. For more information about running Documaker Server as a subordinate process of IDS, see Using IDS to Run Documaker on page 144.

AUTOMATICALLY UPDATING IDOCUMAKER

You can use IDS to update a user's workstation with a new version of the iDocumaker executables. Users are notified of an update when a document is opened if a new version has been made available by the administrator.

NOTE: You must be using version 11.2 or higher of iDocumaker to use the automatic update feature.

The user can then begin the installation by clicking the Begin Installation button. If the user clicks Exit, the update is skipped until he or she opens the next document.

CONFIGURING IDS TO UPDATE IDOCUMAKER

To configure IDS to update iDocumaker, follow these steps:

1. Select the location where the iDocumaker executables will be kept under IDS. This example shows the default location if IDS is running from the \docserv directory.

   ```
   Docserv\data\CONFIG\wipedit
   ```

   If you need to change this you can set the following INI option in the configuration-specific INI file:

   ```
   < WIPEdit >
       ExecDir = d:\docserv\data\sampco\wipedit
   ```

Option	Description
ExecDir	This option tells the update utility (VERSUPD) where the executables are located.

2. Copy the executables for iDocumaker into this directory.

   IDS keeps a file that contains version information for these executables. The contents of this file are included inside the DPW file. The presence of this file determines whether the update process occurs. Here is the default path for this file:

   ```
   Docserv\data\CONFIG\CONFIG.wipedit
   ```

   If this is not an acceptable location you must set the following INI option in the CONFIG.INI file to the appropriate location:

   ```
   < WIPEdit >
       VersionFile = d:\docserv\data\sampco
   ```

3. The update program needs to know the location of the installation file from the web server so you must set up the following INI options. To set up these options, you must know the web site address and the relative path to the installation file within the web site.

   ```
   < INI2XML >
       DownloadURL        = localhost
       DownloadScript     = doc-prog/data/sampco/wipedit.dpi
       DownloadUserID     = (user ID)
       DownloadPassword   = (password)
   ```

Option	Description
DownloadURL	You can enter the web site address or a machine name on the network. For example, you could enter localhost, pd.docucorp.com, www.docucorp.com, or an IP address. The exact value is specific to your implementation. This option is similar to the PUTURL option which may already be in the INI2XML control group.
DownloadScript	This is the part of the URL which points to the location within the host for the installation file. It should contain the name of the installation file. This is similar to the SCRIPT option which usually points to the wipsave.asp or wipsave.jsp file for iDocumaker Workstation. The exact value is specific to your implementation.
DownloadUserID	Enter the user ID for authentication purposes. This entry may be encrypted.
DownloadPassword	Enter the password for authentication purposes. This entry may be encrypted.

Here is an example of how you can use the CRYRUW32 utility at a command prompt to encrypt the data:

```
C:\docserv1.8>cryruw32 password
Encrypted string (2XAUnkxUYlx7i5AnQ4m4E1m00)
```

4 Next, use the VERSUPD utility to build the installation file and version file.

USING THE VERSUPD UTILITY

Use this utility to create the installation file and the version file. These files are created by the VERSUPD utility:

- A version file which contains XML entries for version number, patch level, and accumulated CRC for files without patches.
- An installation file which has all of the executables for iDocumaker compressed into one file. The executable is installed on client machines via the UPDWDT utility.

NOTE: The UPDWDT utility is typically executed by iDocumaker when needed. You do not have to run it.

Both the version file and the installation file need to be created where the IDS rules expect them to be. By default, the VERSUPD utility creates them in same location IDS expects to find them.

Program names

Windows	versupd.exe
UNIX	versupd

Syntax `versupd /config /ini /versionfile /installation /base /debug`

Parameter	Description
/config	Enter the name of the configuration, such as SAMPCO.
/ini	(Optional) Enter the path to the configuration INI file used by IDS. Specifying an INI file helps you make sure IDS and the VERSUPD utility look for shared files in the same location.
/versionfile	(Optional) Enter the path to the version file. This overrides the entry in the INI file.
/installation	(Optional) Enter the path to the installation file. This overrides the entry in the INI file.
/base	(Optional) Enter the path to the executables for iDocumaker Workstation. By default, this utility looks for executables in the data\config\wipedit directory under the current directory. This overrides the entry in the INI file
/debug	(Optional) Include this option to send a list of each file included in the installation to stdout.

Here is an example:

```
versupd /config=SAMPCO
```

This command puts both files in the following path:

```
c:\docserv\Data\SAMPCO
```

Here is another example:

```
versupd /config=SAMPCO /versionfile=c:\docserv\VersionControl
    /installation=c:\docserv\WIPEditInstallation
```

This command creates these files:

File type	Name and path
Version	c:\docserv\VersionControl
Installation	c:\docserv\WIPEditInstallation

INI options These INI options from the CONFIG.INI files are read by the VERSUPD utility:

```
< WIPEdit >
    ExecDir    =
    VersionFile=
```

Option	Description
ExecDir	Enter the location for iDocumaker Workstation's executables.
VersionFile	Enter the location of the file that contains the version information.

Error messages

The following error messages may be generated by versupd. These errors will go both to stdout and to a file named *trace* that will be in the current directory of the VERSUPD utility.

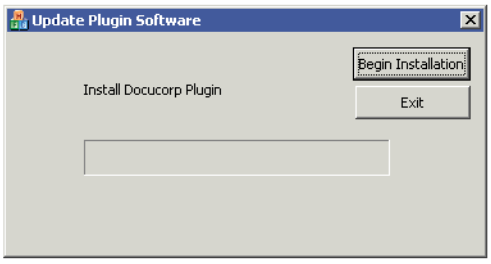
```
Could not create installation file (version file path)
Could not find files to build installation in directory (iDocuMaker
Workstation directory)
Not able to add file (executable name) to installation (installation
file name)
Could not access directory where the iDocuMaker Workstation
executables are suppose to be.
Could not access directory where custom wipedit executables are
suppose to be (custom executable directory)
Unable to create version file (installation file)
Unable to retrieve version information from directory (iDocumaker
Workstation directory)
Unable to create document (version file)
Could not lock version file (version file name).
```

On the Client Side

When iDocumaker is installed the version information is stored in registry. When iDocumaker parses the DPW file, it finds the most current version according to IDS. It then compares the version stored under IDS and the local version. If there is a discrepancy, iDocumaker's installation tool starts.

- The installation tool performs these steps:
- Downloads the archived file of iDocumaker executables.
- Backs up the current iDocumaker executable directory.
- Makes sure you have write access to all of the program files for iDocumaker.
- Erases all of the files in iDocumaker's executable directory.
- Installs the new executables.
- Updates the local version information in the registry.

Once you have set up IDS to automatically update iDocumaker, those computers with version 11.2 or higher of iDocumaker display the following window when you open a document.



Click the Begin Installation button to update iDocumaker. Click Exit to skip the update. The next time the user tries to open a document, IDS will again prompt the user to update iDocumaker.

Additional Utilities

You can also use these additional utilities:

- The VERS2REG utility gets the local version information and updates the registry. This utility executes on the workstation side from a command prompt. Typically, it is only executed during the original installation of iDocumaker. You can, however, start it from a command prompt. You can use the /v and /i parameters to determine which files and patch levels are in the current installation.

  vers2reg /v /i /p /r

Parameter	Description
/v	This parameter writes to stdout the values it will put in the registry.
/i	This parameter tells the utility to simply display the patch and CRC information and not change the registry.
/p	Use this parameter to set the path to the iDocumaker executables instead of using the registry to find the installed location
/r	This parameter indicates the registry key where the utility can find iDocumaker executables.

NOTE: CRC (Cyclic Redundancy Check) is a way to check for data transmission errors.

- The UPDWDT utility gets the iDocumaker installation file from the IDS and then updates iDocumaker.

NOTE: The UPDWDT utility is typically executed by iDocumaker when needed. Information about running this utility is only included in case you are having problems with the iDocumaker and need to update your system.

updwdt /b /i /r

Parameter	Description
/b	This tells the utility to copy the reboot/backup directory contents to the installation directory. This option is used when there are files to update during the reboot process.
/i	This tells the utility to install from a file specified by the next parameter. This is used if the installation file is already present on the local machine.
/r	The registry location where iDocumaker has been installed. The defaults is: `HKEY_CLASSES_ROOT\\wipedit.Document\\protocol\\StdFileEditing\\server`

- Documaker Bridge rules to get the version information and CRC from the iDocumaker executables. You should have a separate location for each CONFIG value for these executables.

CHECKING VERSION INFORMATION

You can use the WDTValidateDPI API to check the version information for iDocumaker from a menu. Place this API function in the WIPEDIT.RES file. When you use this function, these tests are performed:

1. Make sure local version information has been created. This identifies whether the VERS2REG utility has been run during the install process. If the version information does not exist, this message appears:

   ```
   Local version information does not exist for plug-in
   ```

2. Make sure the server version information has been updated. This indicates that server information was created and downloaded in the DPW file. If the version information cannot be found, this message appears:

   ```
   Version information does not exist on the server for plug-in
   ```

3. Compare server side information with local version information. If the version information matches, this message appears:

   ```
   You are running the correct version of the plug-in
   ```

   If the versions do not match, this message appears:

   ```
   Incorrect version of the plug-in - please update
   ```

4. Make sure the compressed file can be downloaded from the web server based on the download information in the registry. If it cannot, this message appears:

   ```
   Not able to locate the installation file on the web server - (followed
   by web address attempted)
   ```

5. Check the format of the installation file. If there is a problem, this message appears:

   ```
   Plug-in installation file is corrupt contact server administrator
   ```

If everything is Ok, you will see at least two messages. If tests 1, 2, and 3 pass the following message appears.

```
You are running the correct version of the plug-in
```

If tests 1, 2, and 3 fail but tests 4 and 5 pass, this message appears:

```
Installation file can be accessed successfully
```

Using the WIP Edit Plug-in

The WIP Edit plug-in lets you present WIP information inside a browser. The plug-in is an Active Document Server application, which means it can run inside Internet Explorer whenever it opens a DPW file. The document is opened inside the browser. The browser menu is not replaced and you can access it by right clicking inside the document.

NOTE: The WIP Edit plug-in only runs inside Microsoft's Internet Explorer. It will work with other browsers, but it will not run inside other browsers.

The WIP Edit plug-in dynamically requests the downloading of the following resources from IDS. The DPRGetResource rule looks in your INI options to locate any resources requested.

- FAP files (The default location is your FormLib directory)
- DAL scripts (The default location is your DefLib directory)
- Tables (The default location is your TableLib directory)
- Help files (The default location is your HelpLib directory)

You must include this request type in the DOCSERV.INI file to dynamically download resources:

```
[ReqType:GETRESOURCE]
    function = atcw32->ATCLogTransaction
    function = atcw32->ATCLoadAttachment
    function = atcw32->ATCUnloadAttachment
    function = dprw32->DPRSetConfig
    function = dprw32->DPRDecryptLogin
    function = dprw32->DPRDefaultLogin
    function = dprw32->DPRCheckLogin
    function = atcw32->ATCSendFile,RETURNFILE,RETURNFILE,Binary
    function = dprw32->DPRGetResource,RETURNFILE
```

You can add entries to WIP by including this request type in the DOCSERV.INI file. This request type creates a DPW file that triggers the Form Selection window.

```
[ReqType:GETEMPTYWIP]
    function = atcw32->ATCLogTransaction
    function = atcw32->ATCLoadAttachment
    function = atcw32->ATCUnloadAttachment
    function = dprw32->DPRSetConfig
    function = dprw32->DPRDecryptLogin
    function = dprw32->DPRDefaultLogin
    function = dprw32->DPRCheckLogin
    function = atcw32->ATCSendFile,RETURNFILE,RETURNFILE,Binary
    function = dprw32->DPRCreateEmptyWipXML,RETURNFILE
    function = dprw32->DPRFile2Dpw,RETURNFILE
    function = dprw32->DPRIni2XML
```

Set this request type to determine if a policy number is already being used.

```
[ReqType:WFIND]
    function = atcw32->ATCLogTransaction
    function = atcw32->ATCLoadAttachment
    function = atcw32->ATCUnloadAttachment
```

```
function = dprw32->DPRSetConfig
function = dprw32->DPRDecryptLogin
function = dprw32->DPRDefaultLogin
function = dprw32->DPRCheckLogin
function = dprw32->DPRFindWipRecords
```

Here are examples of entries in the INI2XML control group:

```
< INI2XML >
    PolicyScript   = doc-prog/iwip/sampco/wipfound.asp
    GetScript      = doc-prog/iwip/sampco/wipdownload.asp
    Key1           = FORMMAKER PACKAGE
    Key2           = PROPERTY;INLAND MARINE
```

Option	Description
PolicyScript	This is a script to run on the web server to check for duplicate policy numbers.
GetScript	This is a script to run on the web server to get resources dynamically.
Key1	If a transaction is created, this sets the Key1 value on the Form Selection window.
Key2	If a transaction is created, this sets the Key2 value on the Form Selection window.

The WIPCTL.DLL file lets you control the document through an ASP page. The WIPCTL.DLL file contains the WIP Edit interface. This lets custom web applications send messages to the WIP Edit plug-in to do things like zoom in or out, advance to the next page or form, and so on. You must register this component with regsvr32.

This component supports these methods:

```
cmd( int cmd);

GotoForm(BSTR formname,int formno,int pageno);
Save( void);
FitToWidth( void);
FitToWindow( void);
ZoomIn( void);
ZoomOut( void);
ZoomNormal( void);
FormPrevious( void);
FormNext( void);
FormSelect( void);
Refresh( void);
FieldTemplat(void);
AutoFocus( void);
Information( void);
FixedEdit( void);
FixedPrompt( void);
Cascade( void);
Tile( void);
Stack( void);
StackOnly( void);
HelpContents( void);
HelpHowTo( void);
```

```
HelpGlossary( void);
UsingHelp( void);
PagePrevious( void);
PageNext( void);
SelectSection( void);
ProductionInformation( void);
HelpShortcuts( void);
```

Here is a example of ASP code for wipctl:

```
Function Save
    set aspobj = CreateObject("Wipctl.WipEd.1")
    aspobj.Save
    set aspobj = Nothing
End Function
```

Here are some things to keep in mind as you use the WIP Edit plug-in:

Turning on debugging

You can use the Debug option to turn on debugging. This lets you turn on debugging without having to individually set the environment variable on client machines running the WIP Edit plug-in.

To turn on debugging using the Debug option, include this option in the wipedit.ini file:

```
< WIPEdit >
    Debug = Yes
```

Automatically sending the WIPEDIT.FXR file

The IDS rules that create the DPW file can automatically send the WIPEDIT.FXR file to the WIP Edit plug-in when these conditions are met in the INI file:

- The DownloadDPWFonts option in the WIP2DPW control group is set to No.
- The XRFToken option is not set in the File2DPW control group.

In the sampco.ini file, comment out the XRFToken option, as shown in this example:

```
< File2DPW >
;   XRFToken = mstrres\sampco\deflib\rel102sm.fxr
```

NOTE: You can only have one installation of the WIP Edit plug-in on a PC.

Saving documents with invalid certificates

The WIP Edit plug-in ignores invalid web certificates, such as when the web certificate has expired. If the certificate is invalid, the system can save the document from the WIP Edit plug-in with the following INI options:

To begin, download an INI file to the WIP Edit plug-in. For this example, use the USER.INI file. Add the following to the configuration specific INI:

```
< File2DPW>
    INIToken = user.ini
```

The USER.INI file should contain the following.

```
< ICMLib>
    IgnoreInvalidCertificate = Yes
```

Running the plug-in outside the browser

You can run the WIP Edit plug-in in its own window (outside the browser) by changing the content type header in the WIPEDIT.ASP or WIPEDIT.JSP web page.

To run	Change the content type header to
Inside the browser	Response.ContentType ="application/octet-stream"
Outside the browser	Response.ContentType ="application/dpw"

When you run it outside the browser, you must delete the file type setup by registering the program and manually setting the file type and association.

Registering the plug-in

The installation routine should register the plug-in for you, but if for some reason you need to register the plug-in, simply run the WIPEDW32 program with no parameters.

Changing values in the WIP index

Use the UpdateDpwIndex INI option to change values in the WIP index based on session variables created in the wipedit.jsp or wipedit.asp page. This option will probably always be used with a customization to the web page to update the WIP index with data from an external source.

If you need to change a WIP index field with a value that originates in the ASP/JSP page, you can use the UpdateDPWIndex option to modify the WIP record when the document is saved. For example, you can use this option to track some other user ID than the login ID the page prompts for.

The following lines are in the wipedit.asp page. A session variable is created called SETORIGUSER. This information is passed to IDS in the form of an attachment variable by the DSI.ProcessQ:

```
session("SETORIGUSER") = "testchange"
On Error Resume Next
DSI.ProcessQ 'Execute Request From Attachment
```

The configuration specific INI must have the following UpdateDpwIndex option:

```
< UpdateDPWIndex >
    OrigUser = #SETORIGUSER
```

The # character tells the system to get the data from the attachment variable named SETORIGUSER. Without the #, the WIP index is updated with the text in the INI file.

The DPRIndex2Xml rule reads the UpdateDpwIndex control group and makes changes in the index portion of the DPW file. When the DPW file is saved, the DPRDpw2Wip rule updates the WIP index with the change.

Changing the WIP index field

You can change the WIP index field in a document from the web page while the document is being edited by the WIP Edit plug-in. This is mainly used with iDocumaker. These methods let you change and retrieve the WIP index fields from the current document:

- SetWipField
- GetWipField
- GetWipIndex

This Visual Basic script sets the DESC field in the WIP index and retrieves it with both methods.

```
Dim wvalue1
```

```
Dim wvalue2
set aspobj = CreateObject("Wipctl.WipEd.1")
        aspobj.SetWipField "DESC", "testvalue"
    wvalue1 = aspobj.GetWipIndex("DESC")
MsgBox(wvalue1)
aspobj.GetWipField "DESC", wvalue2
MsgBox(wvalue2)
```

Running multiple instances of the WIP Edit plug-in

You can have multiple browser windows open, all running the WIP Edit plug-in. Note, however, that if the web application is not designed for multiple browser access by the same user, you will still experience problems.

You can get related debugging information when running multiple instances of the WIP Edit plug-in by setting this environment variable:

```
WIPCTLDEBUG=Y
```

The debugging information is placed in the wipctl.log file in the system's TMP directory or the directory specified by WIPEDITTMP.

NOTE: While this is not an issue for iDocumaker, some implementations in which iDocumaker is integrated with other web applications can be affected.

Using WIP Edit with SiteMinder®

You can use the WIP Edit plug-in with web sites protected by SiteMinder® and with web sites that use clustered web servers. SiteMinder stores security information in a cookie. The WIP Edit plug-in looks for this cookie and attaches the cookie information to requests for resources and the saving of documents.

CONTROLLING THE INTERFACE

The WIPCTL program (WIPCTL.DLL) contains the WIP Edit interface which lets custom web applications send messages to WIP Edit to do things like zoom in or zoom out, advance to the next page or form, and so on. This component must be registered with regsvr32.

NOTE: This information is intended for someone writing ASP or JSP scripts.

The WIPCTL.DLL also includes the CmdWithMessage method which lets someone writing a script receive a response from the WIP Edit plug-in in that script. The following table documents the WIPCTL methods and options:

To...	Use...
Anchor the data entry area at the top of your screen, instead of having it move as you move through the various fields.	FixedEdit(void)
Anchor the data entry area at the top of your screen, instead of having it move as you move through the various fields.	FixedPrompt(void)

To...	Use...
Change data within the form set. See cmdSetFormsetField on page 300 for more information.	cmdSetFormsetField (VARIANT fieldName, VARIANT newValue
Change the form being edited. The parameters include: • formname – Name of the form in the form set. • formno – Instance of the form. This is because a form set may have multiple forms with the same name. • pageno – Page within the form.	GotoForm(BSTR formname,int formno,int pageno);
Change the WIP index field for the current document. See Changing the WIP index field on page 295 for more information.	SetWipField
Check required fields. See BSTR getRequiredFieldName() on page 301 for additional information.	BSTR checkRequiredField()
Decrease the magnification of your form display.	ZoomOut(void)
Display additional information about the variable fields in an image.	Information(void)
Display how to perform specific functions using the various options, commands, and system tools.	HelpHowTo(void)
Display multiple form or image windows in layers. The system stacks the forms one behind another so you see the complete form set and the name or title of each form.	Cascade(void)
Display multiple form or image windows on your screen. If you tile a form set, each window displays the first image of each form in that form set.	Tile(void)
Display multiple form or image windows stacked on top of one another. You close each top layer display window to reveal underlying windows.	Stack(void)
Display retrieved form sets in Stack mode. Stack Only is the default display when you retrieve archived form sets.	StackOnly(void)
Display the entire image in the active window. You see the complete image on one window.	FitToWindow(void)
Display the full width of the image in the active window. You see a complete horizontal display.	FitToWidth(void)
Display window for product information.	ProductionInformation(void)
Document the shortcuts.	HelpShortcuts(void)

To...	Use...
Execute functions defined in WIPEDIT.RES. WIPEDIT.RES is the file that defines the menu for the WIP Edit plug-in. This is similar to the MEN.RES for AFEMAIN. Each function has a number that identifies the function. This is the number that should be used in cmd.	cmd(int cmd);
Find out whether a save command was successful from a Java script. See cmdGetResponse on page 301 for more information.	cmdGetResponse
Get an overview of the system.	HelpContents(void)
Have the system scroll the form as you move through the fields on the form. The current field always stays in view. With this option turned off, it is possible for the field your cursor is in to not appear on your screen.	AutoFocus(void)
Increase the magnification of your form display.	ZoomIn(void)
Learn how to use help.	UsingHelp(void)
Let someone writing the script receive a message back from the WIP Edit plug-in in the script. The WIPEDIT.RES function must provide a response to return. A MEN.RES function must be written to handle this situation. An example is RACCheckRequiredFields. RACCheckRequiredFields checks to see if a function has been installed that will set up a response to be returned in the rsptoasp. This method only works with Visual Basic scripts.	cmdWithMessage(int cmd)
Look up definitions of terms used throughout the system.	HelpGlossary(void)
Move to the next form.	FormNext(void)
Move to the next page.	PageNext(void)
Move to the previous form.	FormPrevious(void)
Move to the previous page.	PagePrevious(void)
Pass a parameter to a function defined in the wipedit.res file. See cmdGetResponseWithParm on page 300 for more information.	cmdGetResponseWithParm (LONG cmd, VARIANT FieldName)
Redraw or redisplay your form after making changes.	Refresh(void)
Retrieve the WIP index field for the current document. See Changing the WIP index field on page 295 for more information.	GetWipField

To...	Use...
Retrieve the WIP index field for the current document. Works just like the GetWipField method except it returns the value of the field instead of setting a parameter. See Changing the WIP index field on page 295 for more information.	GetWipIndex
Return the name of the field that needs data if the result of checkRequiredFields was False. See BSTR getRequiredFieldName() on page 301 for more information.	BSTR getRequiredFieldName()
Return your form to 100% display size.	ZoomNormal(void)
Return WIP Edit plug-in version information. See GetVersion on page 301 for more information.	GetVersion
Save the document. This sends the copy of the document back to the server but it does not close the document.	Save(void)
Select the form in a form set you want to view. This option is helpful when you are viewing a stacked form set.	FormSelect(void)
Select which page you want to view. This option is helpful when you are viewing a stacked form set.	SelectSection(void)
View the size and location of variable fields on a form.	FieldTemplat(void)

NOTE: WIPCTL also includes a Terminate method. Do not use this method. The Terminate method was only included to be consistent with the existing interface.

Example ASP code

Here is example ASP code for WIPCTL.

```
Function Save
    set aspobj = CreateObject("Wipctl.WipEd.1")
    aspobj.Save
    set aspobj = Nothing
End Function
```

Example Visual Basic script

Here is an example Visual Basic script:

```
Dim rspmsg
set aspobj = CreateObject("Wipctl.WipEd.1")
aspobj.cmdWithMessage 263,rspmsg
set aspobj = Nothing
MsgBox(rspmsg)
```

cmdGetResponseWith Parm

Use this method to pass a parameter to a function defined in the wipedit.res file.

```
cmdGetResponseWithParm(LONG cmd, VARIANT FieldName)
```

This is a generic method that is used with a wipedit.res function.

Parameter	Description
cmd	The command ID in the wipedit.res file.
Fieldname	The name of the field in the form set.

In this example the cmdGetReponseWithParm method is used to get the value of a form set field and return it to a Java script. First, in the wipedit.res file, add this line:

```
MENUITEM    "RACGetFormField" 263 "racw32->RACGetFieldData"
```

Here is an example:

```
{
var rsp;
aspobj = new ActiveXObject("Wipctl.WipEd.1");
rsp = aspobj.cmdGetResponseWithParm(263, "COMM PROP PREM");
alert(rsp);
}
```

cmdSetFormsetField

Use this method to change data within the form set.

```
cmdSetFormsetField(VARIANT fieldName, VARIANT newValue
```

Parameter	Description
fieldName	Then name of the field in the form set
newValue	The value to change the field to.

This method returns the previous value of the field.

NOTE: If there are multiple fields in the form set with the same name, the system changes all of the matching names in the form set.

Here is an example:

```
{
var rsp;
aspobj = new ActiveXObject("Wipctl.WipEd.1");
rsp = aspobj.SetFormsetField("COMM PROP PREM", "44");
alert(rsp);
}
```

GetVersion

Use this method to return the current WIP Edit plug-in version information in the following format (a null terminated string separated with semi-colons):

```
dap-patch;3rdparty patch;accumulated CRC;version
```

There are no parameters for this method. Here is an example:

```
<script language="JavaScript">
{

    aspobj = new ActiveXObject("Wipctl.WipEd.1");
    version = aspobj.GetVersion();
    alert(version);

}
</script>
```

cmdGetResponse

The WIP Edit plug-in can send a response back to Java script that indicates the success or failure of a save operation initiated from the web page. This method lets you find out whether a save command was successful from a Java script. Here is an example of the cmdGetResponse method:

```
function CheckRequiredFields() {
    aspobj = new ActiveXObject("Wipctl.WipEd.1");
    var rspmsg = "";
    rspmsg = aspobj.cmdGetResponse(262);
    alert(rspmsg);
    rspmsg = ""
}
```

BSTR getRequiredFieldName( )

If the results of checkRequiredFields was False, use this method to return the name of the field that needs data. Here is an example:

```
function CheckFields() {
    aspobj = new ActiveXObject("Wipctl.WipEd.1");
    var rspmsg = aspobj.checkRequiredField();
    if (rspmsg == "false")
    {

        rspmsg = aspobj.getRequiredFieldName();
        alert(rspmsg);
    }
    else
    {
        alert("all required fields have data");
    }

}
```

SETTING UP CUSTOM FUNCTIONS

Custom functions let you send a message back to IDS by selecting a menu item. To use a custom function, you must set them up consistently in several places:

- Decide the node name for the custom function. This is used to map the transaction from the customer INI file and the WIPEDIT.RES file. In this example, *CUSTFUNC* is the node name.
- Specify the node name in the INI2XML control group in the customer INI file, as shown here

```
< INI2XML >
    MakeNode = CUSTFUNC
```

- The information sent back to the IDS is defined as shown here:

```
< INI2XML:CUSTFUNC >
    ReqType             = WSTATUS
    NewWIP1.StatusCode = AP
    ReqType             = WSTATUS
    WIPS1.RecordID      = #RECNUM
    WIPS                = 1
    WIPS1.Status        = W
    Config              = #CONFIG
    GoChange            = Yes
    PutURL              = LOCALHOST
    EncryptedLogin      = #ENCRYPTEDLOGIN
    UserID              = #ENCRYPTEDLOGIN
    SaveDPWFile         = Yes
    Script              = /doc-prog/iwip/sampco/wipsave.asp
```

  You must define ReqType in the DOCSERV.INI file.

- Make sure the WIPEDIT.RES menu file for the program references the custom function. The last parameter must match the node name, as shown here:

```
MENUITEM "&CUSTOMFUNC" 9910 "racw32->RACtoIDS" CUSTOMFUNC"
[ReqType:WSTATUS]
function = atcw32->ATCLogTransaction
function = atcw32->ATCLoadAttachment
function = atcw32->ATCUnloadAttachment
function = dprw32->DPRSetConfig
function = dprw32->DPRDecryptLogin
function = dprw32->DPRDefaultLogin
function = dprw32->DPRCheckLogin
; function = Atcw32->ATCReceiveFile,RF_POSTFILE,RF_POSTFILE,*
function = dprw32->DPRUpdateWipRecords
```

Changing the User Associated with a Document

You can use the AFEAssignDpw API function to change the user associated with a document through the WIP Edit plug-in.

When the user selects the assign option, a list of the users that can be assigned to the document appears. Select the appropriate user and click Ok. The document is saved and the user ID is assigned.

Here's how to set up this API:

1. Modify the WIPEDIT.RES file, to include the following function. The number 261 can vary, just make sure the number you use is not taken by another line in the WIPEDIT.RES file:

```
MENUITEM    "&Asssign"     261 "AFEOS2->AFEAssignDpw" "Assign"
```

2. Modify the DOCSERV.INI file to include these request types:

```
< ReqType:WLGNINFO >
    function = atcw32->ATCLogTransaction
    function = atcw32->ATCLoadAttachment
    function = atcw32->ATCUnloadAttachment
    function = dprw32->DPRSetConfig
    function = dprw32->DPRDecryptLogin
    function = dprw32->DPRDefaultLogin
    function = dprw32->DPRCheckLogin
    function = dprw32->DPRGetNewLogin
```

3. Modify the configuration specific INI to include the following options. The values for these INI options vary, based on your implementation.

```
< INI2XML >
    ReLoginScript= doc-prog/iwip/sampco/wiprelogin.asp
< File2DPW >
    DBF         = D:\docserv1\userinfo\USERINFO.DBF
    MDX         = D:\docserv1\userinfo\userinfo.mdx
```

Sending Passwords

IDS can use the DPRIni2Xml rule to pass an encrypted password to the WIP Edit plug-in to provide authentication when saving data back to IDS.

```
< INI2XML >
    HTTPUserID     = encrypteduserID
    HTTPPassword   = encryptedpassword
```

You can also use the cryruw32 program to create an encrypted value that can be understood by the WIP Edit plug-in. This lets you avoid putting passwords in the INI file where they can easily be read. For instance, if you enter this from the command line:

```
cryruw32.exe password
```

you will see the output similar to the following:

```
Encrypted string (2XAUnkxUYlx7i5AnQ4m4E1m00)
```

REQUESTING A DICTIONARY

The WIP Edit plug-in can request a user spelling dictionary from IDS when running a spell check.

Use the DPRINI2XML rule to calculate a CRC (Cyclic Redundancy Check) that will be stored in the DPW file. This line will calculate the CRC of a spelling dictionary specified by the user ID:

```
< INI2XML >
    CalcCRC = d:\docserv1\spell\#USERID.tlx!TLX
```

To update the spelling dictionary if the WIP Edit plug-in has changed it, use the DPRPutResource rule:

```
[ ReqType:PUTRESOURCE]
    function = atcw32->ATCLogTransaction
    function = atcw32->ATCLoadAttachment
    function = atcw32->ATCUnloadAttachment
    function = dprw32->DPRSetConfig
    function = dprw32->DPRDecryptLogin
    function = dprw32->DPRDefaultLogin
    function = dprw32->DPRCheckLogin
    function = dprw32->DPRPutResource
```

Specifying the user dictionary

Use the UserDict option to specify the name of the dictionary file you want to use in the WIP Edit spell check process. If you omit this option, the spell dictionary file name is based on the user ID.

To begin, download an INI file to the WIP Edit plug-in. For this example, use USER.INI. Add the following to the configuration-specific INI file:

```
< File2DPW >
    INIToken = user.ini
< Spell >
    UserDict = dictionary.tlx
```

TRAPPING EVENTS

The options to control the trapping of events were implemented because web pages that use anchor tags cause WIP Edit to exit prematurely. If your web page contains anchor tags you may need these options.

These INI options are in the INI file downloaded to WIP Edit, usually named *WIPEDIT.INI*. The INI file is specified in the INIToken option, as shown below:

```
< INI2XML >
    INIToken           = wipedit.ini
< WIPEdit >
    DisableRightClick  =
    TrapEvents         =
    TrapOnlyQuitEvent  =
```

Option	Description
DisableRightClick	Enter Yes to turn off the right-click menu. The default is No.
TrapEvents	Enter No to turn off event trapping. This makes it easier to integrate with iPPS and iDocumaker. The default is Yes.
TrapOnlyQuitEvent	Enter Yes to tell the WIP Edit plug-in to ask the user to save the document when closing the browser, but not when navigating to another page. The default is No. If you set the TrapEvents option to Yes, the TrapOnlyQuitEvent option has no affect.

NOTE: Whether the document is saved or whether you are prompted to save the document depends on the following options in the WIPEDIT.INI file. If you set the OverridePrompt option to Yes, you are not prompted when the plug-in closes. The default is No.

```
< WIPSave >
    OverridePrompt =
```

If you want WIP Edit to automatically save the document. Set the OverridePrompt option to Yes and set the SaveOnExit option to Yes.

```
< WIPSave >
    SaveOnExit =
```

The default for the SaveOnExit option is No.

TRACKING SESSION INFORMATION

The WIP Edit plug-in will let a web application specify data that will be sent back to the web server when a document is saved. This lets iPPS or iDocumaker send session information to the web server/IDS when saving data or getting resources.

The DPRPrintDpw rule looks for groups of attachment variables to add information to the DPW file. This information is used by WIP Edit to add data to the GETRESOURCE and WIPSAVE request.

Use HTTPFORMDATA variables to add multi-form post data:

Variable	Description
HTTPFORMDATA	The number of variables to add multi-form post data
HTTPFORMDATA#.NAME	The name of the variable.
HTTPFORMDATA#.VALUE	The value of the variable.

Use HTTPQUERYSTRING variables to add the query string:

Variable	Description
HTTPQUERYSTRING	The number of variables to add to the query string
HTTPQUERYSTRING#.NAME	The name of the variable.
HTTPQUERYSTRING#.VALUE	The value of the variable.

Use the HTTPHEADER variables to add the HTTP header:

Variable	Description
HTTPHEADER	The number of variables to add to the HTTP header
HTTPHEADER#.NAME	The name of the variable.
HTTPHEADER#.VALUE	The value of the variable.

Use the HTTPCOOKIE variables to add the cookie header:

Variable	Description
HTTPCOOKIE	The number of variables to add to the cookie header
HTTPCOOKIE#.NAME	The name of the variable.
HTTPCOOKIE#.VALUE	The value of the variable.

Examples

Here are some examples:

To add multipart form data to the HTTP request the following attachment variables were added to the request that creates the DPW file:

```
HTTPFORMDATA = 1
HTTPFORMDATA1.NAME = nameformdata1
HTTPFORMDATA1.VALUE = valueformdata1
```

The resulting line in the HTTP request would look like this:

```
----------------------------7d32f01b1003de
Content-Disposition:form-data; name="nameformdata1"
```

```
valueformdata1
```

To add data to the query string for the HTTP request these attachment variables were added to the request that creates the DPW file.

```
HTTPQUERYSTRING 1
HTTPQUERYSTRING1.NAME = SESSIONID
HTTPQUERYSTRING1.VALUE = 8010e572-001b-43e3-98f4-e1b0e0116933
```

In the resulting line in the HTTP request, HTTPQUERYSTRING adds the following information to the URL. Here is an example:

```
/doc-prog/iwip/sampco/wipsave.asp?SESSIONID= 8010e572-001b-43e3-
98f4-e1b0e0116933 HTTP/1.1
```

To create a header for the HTTP request these attachment variables were added to the request that creates the DPW file:

```
HTTPHEADER = 1
HTTPHEADER1.NAME = someheader1
HTTPHEADER1.VALUE = someVALUE1
```

In the resulting line in the HTTP request, HTTPHEADER adds information to the HTTP header. The following example is from a save request:

```
someheader1:someVALUE1
```

To add data to the cookie header the HTTP request the following attachment variables were added to the request that creates the DPW file:

```
HTTPCOOKIE = "1"
HTTPCOOKIE1.NAME = "cookie"
HTTPCOOKIE1.VALUE = "Toolfloat=false; Toolbottom=false;
IX=%E9%C4%92%
08%C9%D3xo%D9%2D%AF%D3%A0%AC%26%15%7E%FA%23M%01%D9%FDt%23%A2%13%7E%
CAN%95%80%B2%
E5cC%0Enj%E7%1E%E4; ASPSESSIONIDACRTQSCA=EGPPLAECPNDMGOIIMANOAPPB"
```

The resulting cookie header in the HTTP request would look like this:

```
Cookie: Toolfloat=false; Toolbottom=false;
IX=%E9%C4%92%08%C9%D3xo%D9%2D%AF%D3%A0%AC%26%15%7E%FA%23M%01%D9%FDt
%23%A2%13%7E%CAN%95%80%B2%E5cC%0Enj%E7%1E%E4;
ASPSESSIONIDACRTQSCA=EGPPLAECPNDMGOIIMANOAPPB;
ASPSESSIONIDQSBCQTCA=JHCKEELAANHOGDGAPMABIHDL
```

SETTING UP PRINTERS

This topic tells you how to set up printers for the WIP Edit plug-in. The WIP Edit plug-in gets the fonts it needs from Docupresentment, using the GETRESOURCE request. This helps insure better fidelity of printed copies by using PCL or PostScript printers.

To use set up printers, make sure...

- The WIPEDIT.RES file includes the print option.
- The WIPEDIT.INI file has the print types set up in the same manner as those in Documaker Workstation or PPS.

NOTE: The INITOKEN option in the File2Dpw control group of the CONFIG.INI file must be set before options in the WIPEDIT.INI file can take effect.

- The GETRESOURCE type is in the docserv.xml or DOCSERV.INI file.
- The FontLib option in the MasterResource control group is set in the CONFIG.INI file.

Here is an example of the WIPEDIT.RES file:

```
POPUP "&Print" 1070 "Print"
BEGIN
MENUITEM "Pri&nt Formset..." 1065 "NULL" "NULL"
MENUITEM "&Form..." 1066 "NULL" "NULL"
MENUITEM "Pa&ge..." 1067 "NULL" "NULL"
END
```

Here are examples for the WIPEDIT.INI file:

```
:PostScript examples.
< PrtType:PST >
    DownloadFonts = Yes,Enabled
    Module = PSTW32
    PrintFunc = PSTPrint
    Resolution = 300
;   SendOverlays = Yes,Enabled
    SendOverlays = No,Disabled
< PrtType:PXL >
    DownloadFonts = Yes,Enabled
    Module = PXLW32
    PageNumbers = Yes
    PrintFunc = PXLPrint
    SendOverlays = No,Enabled
< PXL >
    Device = \\Atl1dc01\YEL_HP8000_A
    DownloadFonts = Yes
    Module = PXLW32
    PrintFunc = PXLPrint
< PrtType:PCL >
    DownloadFonts = Yes,Enabled
    Module = PCLW32
    MultipleCopies = Yes
    PrintFunc = PCLPrint
    SendOverlays = No,Enabled
```

Here is an example the docserv.xml file:

```
<section name="ReqType:GETRESOURCE">
<entry name="function">atcw32->ATCLogTransaction</entry>
<entry name="function">atcw32->ATCLoadAttachment</entry>
<entry name="function">atcw32->ATCUnloadAttachment</entry>
<entry name="function">dprw32->DPRSetConfig</entry>
<!-- entry name="function">dprw32->DPRDecryptLogin</entry -->
<!-- entry name="function">dprw32->DPRDefaultLogin</entry -->
<!-- entry name="function">dprw32->DPRCheckLogin</entry -->
```

```
<entry name="function">atcw32-
>ATCSendFile,RETURNFILE,RETURNFILE,Binary</entry>
<entry name="function"<dprw32->DPRGetResource,RETURNFILE</entry>
<!-- -->
</section>
```

Here is an example of the DOCSERV.INI file:

```
[ReqType:GETRESOURCE]
function = atcw32->ATCLogTransaction
function = atcw32->ATCLoadAttachment
function = atcw32->ATCUnloadAttachment
function = dprw32->DPRSetConfig
function = dprw32->DPRDecryptLogin
function = dprw32->DPRDefaultLogin
function = dprw32->DPRCheckLogin
function = atcw32->ATCSendFile,RETURNFILE,RETURNFILE,Binary
function = dprw32->DPRGetResource,RETURNFILE
```

Here is an example of how you would set the FontLib option:

```
< MasterResource >
    FontLib = mstrres\sampco\fmres
```

For more information on setting printing options, see the Documaker Server Reference Guide or the Documaker Workstation Supervisor Guide.

Chapter 7

Using the DP.DLL ActiveX Interface

DP.DLL is a COM object that can be used by ASP client applications to communicate with IDS via SOAP messages and the MQSeries message bus without an IDS client. It supports the same SOAP message format as IDS, including rowsets and file attachments.

Connection information can come from an MQSERVER environment variable, a Client Connection Definition Table (CCDT), or from properties in an XML configuration file.

You can use DP.DLL as a standalone client DLL to communicate with a remote IDS via MQSeries and SOAP attachments using the Microsoft IMessage interface. This is supported on Microsoft Windows 2000 and later platforms.

NOTE: The DP.DLL COM object is not distributed with IDS. To receive this additional feature, contact your sales representative.

This appendix includes information on the following topics:

- Requirements on page 312
- Setting Up the Configuration File on page 313
- Properties on page 315
- Methods on page 316
- Examples on page 325

REQUIREMENTS

To use the DP.DLL ActiveX Interface, you must have the following:

- SOAP Toolkit version 2.0 (MSSMO.dll)
- MSXML version 4.0
- CDO for Windows 2000 or later
- MQSeries. Client installation (version 5.3 or later is required for SSL connections)

You must have these default directories under the virtual directory:

- Cache (used to write all input and output files)
- Debug (used to write all debug files)

Setting Up the Configuration File

You must have an XML configuration file called INI.XML. Place this file under the virtual directory. It can contain these properties:

Property	Description
QUEUEMANAGER	The name of the queue manager.
RESULTQ	The name of the input queue.
REQUESTQ	The name of the output queue.
CHANNEL	The channel name, should correspond to the name of a server connection channel, used to create a matching client connection channel at run time.
CONNECTION	The IP address and port number of the box hosting the queue manager and server connection channel.
MSGLEN	The maximum message length of a message. Be sure to configure the queue manager and server connection to support the same message length. This property is optional. The default max message size is 4MB.
SSLCIPHERSPEC	The cipher specification (encryption and hashing algorithm) that should be used in SSL connections. Should correspond to the Cipherspec chosen for the server connection channel when SSL connections were enabled for it. This property is only required when establishing connections to a queue manager configured for SSL.
SSLPEERNAME	The DN (Distinguished Name) of the subject in the SSL certificate used by the queue manager. Used to verify the client application is connecting to the correct queue manager. This property is only required when establishing connections to a queue manager configured for SSL and when client applications desire to verify the DN of the certificate used by the queue manager - enabling this option forces the queue manager to send its certificate to the client for verification of the DN as part of the SSL handshake.
SSLCLIENTAUTH	This property is only required when establishing connections to a queue manager configured for SSL and when client applications desire to verify the certificate used by queue manager - enabling this option forces the queue manager to send its certificate to the client for verification of the DN as part of the SSL handshake.

Here is an example of a configuration file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<GROUPS>
   <GROUP NAME="MQSERIES">
<QUEUEMANAGER>queue_manager</QUEUEMANAGER>
<REQUESTQ>REQUESTQ</REQUESTQ>
<RESULTQ>RESULTQ</RESULTQ>
<CHANNEL>SSLCHANNEL1</CHANNEL>
<CONNECTION>X.X.X.X(1414)</CONNECTION>
<MSGLEN></MSGLEN>
    <SSLCIPHERSPEC>RC4_MD5_US</SSLCIPHERSPEC>
    <SSLPEERNAME>CN=ssl_qmgr, C=US, S=GA, L=Atlanta, O=Acme, Co.,
OU=PD</SSLPEERNAME>
    <SSLCLIENTAUTH>Y</SSLCLIENTAUTH>
   </GROUP>
</GROUPS>
```

NOTE: If the MQSERVER or MQCHLLIB and MQCHLTAB system environment variables are specified, the system uses them to derive the connection information instead of using the property values in the XML configuration file.

Keep in mind...

- SSL is only supported in MQSeries version 5.3 or later.
- When using SSL, be sure to first give the IIS account read permission to the key.sto file (the SSL key repository).
- The IIS account must have access to the following registry keys to send the client certificate to the queue manager when the following SSL options are enabled in the server connection channel. Otherwise, WebSphere MQ issues error code 2193 complaining the client application did not send the certificate to the queue manager for verification:

  Registry Keys:

  ```
  HKEY_USERS\.Default\Microsoft\Software\SystemCertificates\Root
  HKEY_USERS\.Default\Microsoft\Software\SystemCertificates\trust
  HKEY_USERS\.Default\Microsoft\Software\SystemCertificates\CA
  HKEY_USERS\.Default\Microsoft\Software\SystemCertificates\my
  ```

  The easiest thing to do is to configure IIS Out-Of-Process Applications under 'Component Services' mmc snap/in to run under an identity that has permissions to these keys and restart IIS. Alternatively, the two options specified below can be disabled in the server connection channel to avoid requiring client applications send their certificate for verification as part of the SSL handshake.

  Server Connection Channel Options:

  Only Accept Certificates with Distinguished Names matching these values.'
  Always Authenticate parties initiating connections to this channel definition.'

PROPERTIES

The DP.DLL ActiveX interface includes these properties:

Property	Description
Request	Type: Collection Contains request name/value pairs for a transaction.
Result	Type: Collection Contains result name/value pairs for a transaction
ErrMsg	Type: String Contains an error description of the last error encountered in the MQSeries APIs.
RC	Type: Integer Can return either zero (0) for success or one (1) for failure for the last MQSeries API Call in DP.DLL.
bDebug	Type: Boolean Can be set to one (1) or True or zero (0) or False. Used to write the inbound and outbound SOAP attachments and XML form sets to disk. Also used to enable tracing throughout DP.dll. Tracing output goes to trace.txt file located on the root context of the web application.
Expires	Type: Long Used to set the time in minutes a message will exist in the queue before it is removed by MQSeries.
TimeOut	Type: Long Used to set the time the MQSeries MQGet API will wait for a message when attempting to retrieve a message from the queue.
ShowAtt	Type: Boolean Can be one (1) or True or zero (0) or False. Used to display the request and result collections on an ASP page for debugging purposes.
GUID	Type: String Contains the unique message identifier for a SOAP message. Used in putMsg and getMsg calls in order to match a request to a response.
CleanUpInterval	Type: Long Contains the clean up interval of the cache. Always set to three or four times the session expiration time in order to avoid a conflict. Set the time in minutes.
OutputBuffer	Type: String Contains the outgoing SOAP attachment before calling putMsg.
InputBuffer	Type: String Contains the incoming SOAP attachment retrieved by getMsg call.

METHODS

The DP.DLL ActiveX Interface includes these methods:

- AddNameValuePair on page 317
- Bin2Unicode on page 317
- CleanCache on page 317
- GetMsg on page 318
- GetUniqueString on page 318
- Initialize on page 318
- InitializeDefaults on page 319
- ProcessTrn on page 319
- PutMsg on page 319
- ReadIniOptions on page 320
- RequestValue on page 320
- ResultValue on page 321
- SetGUID on page 321
- SOAPAddAttachment on page 321
- SOAPGetAttachment on page 322
- SOAPGetAttachmentAsBuffer on page 322
- SOAPLoadAttachment on page 322
- SOAPUnloadAttachment on page 323
- Terminate on page 323
- Trace on page 323
- Trace on page 323
- WriteBinFile on page 324
- WriteToLog on page 324

AddNameValuePair

Use this method to add name/value pairs from a Session, Form, or QueryString Collection to the request collection.

Syntax

```
AddNameValuePair(Name, Value)
```

Parameters

Parameter	Description
Name	The index name of the name/value pair.
Value	The value of the name/value pair.

See Example 1 on page 325 and Example 2 on page 327.

Bin2Unicode

Use this method to convert a binary string into a Unicode string.

Syntax

```
Bin2Unicode (sABSTR)
```

Parameters

Parameter	Description
sABSTR	A binary string.

CleanCache

Use this method to read every record in the random access file:

```
APPL_PHYSICAL_PATH & "log.db"
```

and compare its date and time stamp to the CleanUpInterval property.

Syntax

```
CleanCache
```

If the time difference exceeds the interval, the method deletes the record from the log, removes the file from the cache, and marks the record as deleted so the same record can be used again by the WriteToLog method.

FileExists

Use this method to see if a file exists.

Syntax

```
FileExists (FileName)
```

This method returns True if the file exists, otherwise False.

Parameters:

Parameter	Description
FileName	Enter the full file name of the file to check.

See Example 1 on page 325.

GetMsg

Use this method to retrieve a SOAP message from the result queue into the InputBuffer property.

Syntax

```
GetMsg
```

This method expects the TimeOut and GUID properties to be set. This method call is only necessary when processing a transaction by calling the individual methods instead of calling the ProcessTrn method. This method returns zero (0) for success or one (1) for failure.

See Example 2 on page 327.

GetUniqueString

Use this method to return a unique identifier string.

Syntax

```
GetUniqueString
```

See Example 2 on page 327.

Initialize

Use this method to connect to the queue manager and open the input and output queues.

Syntax

```
Initialize
```

Make sure the InitializeDefaults and ReadIniOptions methods are called first to set the default MQ objects and connection properties. This method call is only necessary when you are processing a transaction by calling the individual methods instead of calling the ProcessTrn method.

This method returns zero (0) for success or one (1) for failure.

See Example 2 on page 327.

INITIALIZEDEFAULTS

Use this method to initialize the MQSeries defaults.

Syntax

```
InitializeDefaults
```

Call this method before any other method calls. It is only required if processing a transaction by calling the individual methods instead of calling the ProcessTrn method.

See Example 2 on page 327.

PROCESSTRN

Use this method to:

- Initialize the MQSeries default settings.
- Read all connection properties from the INI.XML file.
- Initialize the MQSeries connection and open the queues for input and output.
- Generate a message ID to correlate a request with a response message.
- Generate the SOAP request message from the request collection.
- Put the SOAP request message in the request queue by message ID.
- Retrieve the result SOAP message from the result queue by message ID.
- Unload the result SOAP message into the result collection.
- Close the queues and disconnect the queue manager.

Syntax

```
ProcessTrn
```

See Example 1 on page 325.

PUTMSG

Use this method to place a SOAP message in the request queue.

Syntax

```
PutMsg
```

This method expects the GUID and OutputBuffer properties to be set. This method call is only necessary when you are processing a transaction by calling the individual methods instead of calling the ProcessTrn method. This method returns zero (0) for success or one (1) for failure.

See Example 2 on page 327.

READINIOPTIONS

Use this method to read these options from the INI.XML file located in the root context of the web application:

- QUEUEMANAGER
- RESULTQ REQUESTQ
- CHANNEL
- CONNECTION
- MSGLEN
- SSLCIPHERSPEC
- SSLPEERNAME
- SSLCLIENTAUTH

Syntax

```
ReadIniOptions
```

Always call this method immediately after InitializeDefaults method to set the connection properties before you call the Initialize method. This method call is only necessary when you are processing a transaction by calling the individual methods instead of calling the ProcessTrn method.

See Setting Up the Configuration File on page 313 for a description of each property. See Example 2 on page 327.

REQUESTVALUE

Use this method to return a value in the request collection name/value pair, found by NameIndex.

Syntax

```
RequestValue (NameIndex)
```

This method returns an empty string if the name/value pair is not found.

Parameters

Parameter	Description
NameIndex	The name index of the name/value pair in the request collection.

See Example 1 on page 325.

RESULTVALUE

Use this method to returns a value in the result collection name/value pair, found by the NameIndex parameter.

Syntax

```
ResultValue (NameIndex)
```

This method returns an empty string if the name/value pair is not found.

Parameters

Parameter	Description
NameIndex	The name index of the name/value pair in the result collection.

See Example 1 on page 325.

SETGUID

Use this method to set the message ID that should be used for a request/response transaction.

Syntax

```
SetGUID
```

This method call is only necessary when you are processing a transaction by calling the individual methods instead of calling the ProcessTrn method.

See Example 2 on page 327.

SOAPADDATTACHMENT

Use this method to add a file as a SOAP attachment to the request message.

Syntax

```
SOAPAddAttachment( FileName, ID, Type)
```

Parameters

Parameter	Description
FileName	The full file name of the file to add as an attachment.
ID	The unique identifier for the file attachment.
Type	The media type and transfer encoding type for the attachment. You can choose from TEXT or BINARY

Always call the SOAPAddAttachment method before calling the SOAPLoadAttachment method.

See Example 1 on page 325.

SOAPGetAttachment

Use this method to retrieve a SOAP attachment from the result message as a file written to disk.

Syntax

```
SOAPGetAttachment (FileName, ID)
```

Parameters

Parameter	Description
FileName	The full file name of the file that will be unloaded.
ID	The unique identifier for the file attachment in the SOAP message.

This method returns True if the attachment was found, otherwise, False.

See Example 1 on page 325.

SOAPGetAttachmentAsBuffer

Use this method to return a buffer containing the attachment or an empty string if the attachment was not found.

Syntax

```
SOAPGetAttachmentAsBuffer (ID)
```

The method returns a SOAP attachment as a string buffer.

Parameters

Parameter	Description
ID	The unique identifier for the file attachment in the SOAP message.

SOAPLoadAttachment

Use this method to convert the request collection into a SOAP message.

Syntax

```
SOAPLoadAttachment
```

This method expects the request collection to be set through AddNameValuePair method calls. The method expects file attachments to be set through the SOAPAddAttachment method calls. This method sets the OutputBuffer property.

This method call is only necessary when you are processing a transaction by calling the individual methods instead of calling the ProcessTrn method.

See Example 2 on page 327.

SOAPUnloadAttachment

Use this method to extract the SOAP message from the InputBuffer property and set the result collection.

Syntax

```
SOAPUnloadAttachment
```

This method call is only necessary when you are processing a transaction by calling the individual methods instead of calling the ProcessTrn method.

See Example 2 on page 327.

Terminate

Use this method to close the input and output queues and disconnect from the queue manager.

Syntax

```
Terminate
```

This method call is only necessary when processing a transaction by calling the individual method instead of calling the ProcessTrn method. This method returns zero (0) for success or one (1) for failure.

See Example 2 on page 327.

Trace

Use this method to write the date/time stamp, including milliseconds along with the contents of a buffer to a trace log location defined as:

```
ASP.Server.MapPath("trace.txt")
```

Syntax

```
Trace (Buffer)
```

Parameters

Parameter	Description
Buffer	A string buffer that contains the information you want written to the log.

Unicode2Bin

Use this method to convert a Unicode string into a binary string.

Syntax

```
Unicode2Bin (str)
```

Parameters

Parameter	Description
str	A Unicode string.

WRITEBINFILE

Use this method to write the contents of a file to a browser.

Syntax

```
WriteBinFile (FileName)
```

Parameters

Parameter	Description
FileName	The full file name of the binary file to write to the browser.

See Example 2 on page 327.

WRITETOLOG

Use this method to write entries into the cleanup log.

Syntax

```
WriteToLog (FileName)
```

Each entry contains the full path and file name of a file written to the cache directory. The path and name of the log file is:

```
APPL_PHYSICAL_PATH & "log.db"
```

Each entry contains the date and time stamp and a deleted flag initially set to False. The system uses the first record marked as deleted in the log as the record place for the new record to save space.

Parameters

Parameter	Description
FileName	Full file name of the file to write to the cleanup log.

See also the CleanCache method and CleanUpInterval property.

EXAMPLES

Here are some examples that show you how to use the DP.DLL ActiveX Interface methods.

Example 1

This example uses the ProcessTrn method to send and receive a request and reply to and from IDS:

1. This HTML page submits a request to an ASP page:

```
<html>
<head>
</head>
<body>
<form name=submitReq action="ProcessTrn_Example.asp" method=post>
<input name="GROUP1" value="GENERAL LIABILITY" type=hidden>
<input name="GROUP2" value="APPLICATION" type=hidden>
<input name="CONFIG" value="AMERGEN" type=hidden>
<input name="USERID" value="DOCUCORP" type=hidden>
<input name="PASSWORD" value="DOCUCORP" type="hidden">
<input name="PASSWORDENCRYPTED" type=hidden value="NO">
<input name="ARCEFFECTIVEDATE" value="20020819" type=hidden>
<input name="PRINTPATH" type=hidden value="Output\">
<input name="PRTTYPE" value="PDF"><br>
<input name="REQTYPE" value="FRMPBPRTTEST"><br>
<input type=submit name="btnSubmit" value="submit">
</form>
```

2. This ASP page calls the ProcessTrn method to send/receive a request/response to/from IDS:

```
 <%

Set DP = server.CreateObject("DP.IDSMessage")
DP.ShowAtt = 0
DP.bDebug = 1

For i=1 to Request.Form.Count
   DP.AddNameValuePair Request.Form.Key(i), Request.Form(i)
Next

VirtualPath  = Request.ServerVariables("APPL_PHYSICAL_PATH") &
"Cache\"
OutputFormset =  VirtualPath & "OutputFormset.xml"

DP.SOAPAddAttachment OutputFormset, "ATC_XMLFORMSET", "BINARY"

DP.ProcessTrn()

File = getFilename(DP.ResultValue("PRINTFILE"), "\")
                FullFileName = VirtualPath & File

If (DP.SOAPGetAttachment (FullFileName, "OUTFILE")) Then

    Set DP = Nothing
    Session("File") = FullFileName
    Response.Redirect "Print.asp"
Else
    Response.Write "Error encountered retrieving file!"
```

```
    Set DP = Nothing
End if

    function getFileName(sFile, delimiter)
        getFileName = Mid(sFile, InstrRev(sFile, delimiter, -1)+1,
len(sFile))
    end function
%>
```

3. Then, this ASP print page appears:

```
 <%

    File = Session("File")

    set DP = Server.CreateObject("DP.IDSMessage")

    DP.WriteBinFile(File)

    set DP = Nothing



%>
```

Example 2 This example shows how to use the individual APIs to send and receive requests and replies to and from IDS:

1 This HTML page sends a request to an ASP page:

```
<html>
<head>
</head>
<body>
<form name=submitReq action="APIs_Example.asp" method=post>
<input name="GROUP1" value="GENERAL LIABILITY" type=hidden>
<input name="GROUP2" value="APPLICATION" type=hidden>
<input name="CONFIG" value="AMERGEN" type=hidden>
<input name="USERID" value="DOCUCORP" type=hidden>
<input name="ARCEFFECTIVEDATE" value="20020819" type=hidden>
<input name="PRINTPATH" type=hidden value="Output\">
<input name="PRTTYPE" value="PDF"><br>
<input name="REQTYPE" value="FRMPBPRTTEST"><br>
<input type=submit name="btnSubmit" value="submit">
</form>
```

2 This ASP page calls the individual functions to send and receive requests and responses to and from IDS:

```
<%

    Set DP = Server.CreateObject("DP.IDSMessage")
    DP.ShowAtt = 0
    DP.bDebug = 1
    DP.Expires = 300
    DP.TimeOut = 60


    For i=1 to Request.Form.Count
    DP.AddNameValuePair Request.Form.Key(i), Request.Form(i)
    Next


    DP.InitializeDefaults

    DP.ReadINIOptions

    DP.Initialize


    if DP.RC <> 0 then
    Response.Write DP.ErrMsg
    end if

    GUID = DP.GetUniqueString
    DP.SetGUID(GUID)

    VirtualPath  = Request.ServerVariables("APPL_PHYSICAL_PATH") &
"Cache\"

    OutputFormset =  VirtualPath & "OutputFormset.xml"
```

```
    DP.SOAPAddAttachment OutputFormset, "ATC_XMLFORMSET", "BINARY"

    DP.SOAPLoadAttachment

    DP.PutMsg

    DP.GetMsg

    DP.SOAPUnloadAttachment

    DP.Terminate

    File = getFilename(DP.ResultValue("PRINTFILE"), "\")
    FullFileName = VirtualPath & File


    If (DP.SOAPGetAttachment (FullFileName, "OUTFILE")) Then
    Set DP = Nothing
    Session("File") = FullFileName
    Response.Redirect "Print.asp"
    Else
    Response.Write "Error encountered retrieving file!"
    Set DP = Nothing
    End if


    function getFileName(sFile, delimiter)
        getFileName = Mid(sFile, InstrRev(sFile, delimiter, -1)+1,
len(sFile))
    end function
%>
```

2.3Asp print page:

```
<%
File = Session("File")

set DP = Server.CreateObject("DP.IDSMessage")

DP.WriteBinFile(File)

set DP = Nothing

%>
```

Appendix A

System Files

The following pages list and explain the various files which comprise the Internet Document Server. These are the files installed on your computer when you install the Internet Document Server and its various bridges.

This includes information about the following:

- IDS Configuration Files on page 330
- Sample Output Files on page 333

There are two file types that are treated in a special manner by the web server package—CGI-compliant executables and HTML content pages. The Internet Document Server includes both types of files.

Typically, CGI-compliant executable files are stored in directories specified as CGI directories to the web server. In a similar manner, HTML content pages are stored in directories specified as content directories to the web server.

IDS Configuration Files

The Internet Document Server and its bridges use the following INI files:

File	Used for
fapcomp.ini	System tools, such as the Font Manager
docserv.xml	Internet Document Server settings
docclient.xml	IDS client settings
docclnt.ini	IDS CGI client application
dsi.ini	custom client programs written using VB, Java, and so on, which call DSI APIs.
dap.ini	the various bridges
(*resource*).ini	the various libraries

Since the server must start before a client can begin processing, the docserv.xml file is read first.

The same option can be defined in both the DAP.INI file and in the various INI files for your resources. When this happens, the settings in the resource INI files take precedence over those in the DAP.INI file.

Docserv.xml file format

Prior to IDS version 2.0, the configuration file was a simple INI file (docserv.ini). For IDS 2.0, the format changed to an XML file. This gives you more control over configuration options.

In the INI format, you could only have one level of control groups (sections), with entries under each group or section. Using the XML format, you can now have multiple levels of subsections under a section for better grouping. Options relevant to the passing of messages can be, for example, grouped under a messaging subsection.

The general format of the docserv.xml file is:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <section name="DocumentServer">
        <entry name="FileWatchTimeMillis">10001</entry>
        <entry name="FilePurgeTimeSeconds">3600</entry>
        <entry name="FilePurgeList">purgeme.properties</entry>
    </section> <!-- DocumentServer -->
    <section name="BusinessLogicProcessor">
        <section name="MultiThreadedRequests">
            <entry name="Request">ECH</entry>
        </section> <!-- MultiThreadedRequests section -->
        <section name="messaging">
            <section name="http">
                <entry name="port">49152</entry>
            </section> <!-- http section -->
            <section name="timed">
                <entry name="AutoRunIntervalSeconds">3600</entry>
                <section name="Timers">
                    <entry name="XYZ">Tue 3:27:01 PM</entry>
```

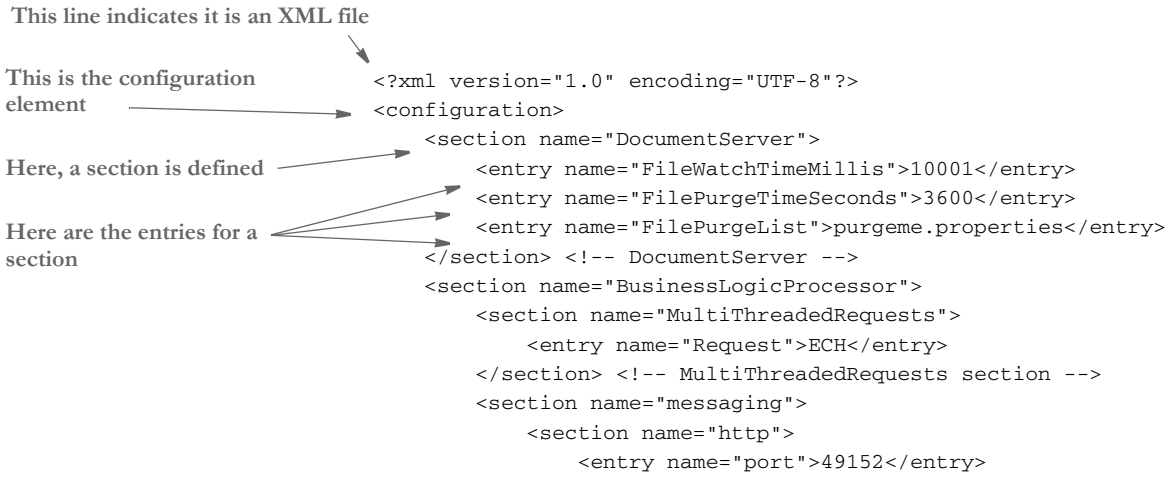
```
                </section>
            </section> <!-- timed section -->
            <section name="queue">
                <entry
name="queuefactory.class">com.docucorp.messaging.mqseries.DSIMQMess
ageQueueFactory</entry>
                <!-- Settings for MQSeries connection -->
                <entry name="mq.queue.manager">queue.manager</entry>
                <entry name="mq.inputqueue.name">requestq</entry>
                <entry name="mq.inputqueue.maxwaitseconds">5</entry>
                <entry name="mq.outputqueue.name">resultq</entry>
                <entry name="mq.tcpip.host">10.1.10.1</entry>
                <entry name="mq.queue.channel">queue_channel</entry>
                <entry name="mq.tcpip.port">1414</entry>
            </section> <!-- queue section -->
        </section> <!-- messaging section -->
    </section> <!-- BusinessLogicProcessor -->
    <section name="ReqType:INI">
        <entry name="function">irlw32->;IRLInit</entry>
        <entry name="function">dprw32->;DPRInit</entry>
        <!-- Following rule now initted in THREADINI -->
        <!-- entry name="function">DSICoRul->;Init</entry -->
        <!-- entry name="function">pobrs->;POWInit</entry -->
        <entry name="function">Tpdw32->;TPDInitRule</entry>
    </section>
    <section name="ReqType:THREADINI">
        <entry name="function">atcw32->;ATCLoadAttachment</entry>
        <entry name="function">atcw32->;ATCUnloadAttachment</entry>
        <entry name="function">DSICoRul->;Init</entry>
        <entry name="function">DSICoRul-
>;Invoke,DocuCorp_IDS_DPRCo.DPR->;DPRCoLoginInit</entry>
    </section>
    <section name="ReqType:ECH">
        <entry
name="function">java;com.docucorp.ids.rules.EchoTest;;transaction;e
cho;</entry>
    </section>
</configuration>
```

The file begins with the line indicating it's an XML file. Under that is the *configuration* element, the root element of the XML. Inside the configuration element are several *section* elements, each with a *name* attribute to identify the section. Some section names, such as *REQTYPE:INI* are the same as in IDS version 1.

A section may just have several *entry* elements inside it. Each entry has a *name* attribute to identify it, and the text in between the <entry> and </entry> tags is the value of the entry.

A section may also have other section elements inside of it, for example the BusinessLogicProcessor section. The BusinessLogicProcessor section has subsections pertaining to getting requests to process and sending results back to clients.

In this document any configuration settings will list the section, and optionally any subsections, that an entry belongs to.



Docclient.xml format

Similar to IDS 2.0, most IDS client programs now use an XML-based configuration file. The exception is docclnt, the CGI client program.

The general format of the docclient.xml file is:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <section name="DocumentClient">
    <section name="messaging">
      <section name="queue">
        <entry name="queuefactory.class">com.docucorp.messaging.
           mqseries.DSIMQMessageQueueFactory</entry>
        <!-- Settings for MQSeries connection -->
        <entry name="mq.queue.manager">queue.manager</entry>
        <entry name="mq.inputqueue.name">requestq</entry>
         <entry name="mq.inputqueue.maxwaitseconds">5</entry>
         <entry name="mq.outputqueue.name">resultq</entry>
        <entry name="mq.tcpip.host">10.1.10.1</entry>
        <entry name="mq.queue.channel">queue_channel</entry>
        <entry name="mq.tcpip.port">1414</entry>
      </section>  <!-- queue section -->
    </section>  <!-- messaging section -->
  </section> <!-- DocumentClient -->
</configuration>
```

The overall structure is similar to docserv.xml. The main difference is that the messaging parameters are under a "DocumentClient" section. This makes it possible for client applications and IDS use the same configuration file, with client settings under the "DocumentClient" section and IDS settings under the "DocumentServer" and "BusinessLogicProcessor" sections.

Sample Output Files

Here are printouts of the sample output files you should receive when you check your server installation.

The output comes from these functions:

- DSIEXW32.EXE
- DSICOTB.EXE, option ESS
- DSICOTB.EXE, option Roll Your Own
- DSICOTB.EXE, option RSS
- DSICOTB.EXE, option SSS

DSIEXW32.EXE

Here is the output you should see when you execute DSIEXW32.EXE. You will see similar results when you execute DSICoEx.

```
Name = ALLOCCOUNT Value = 3073
Name = ERRORCOUNT Value = 0
Name = FREECOUNT Value = 391
Name = LASTRESTART Value = Wed Aug 12 16:31:14 1998
Name = LIBRARIES Value = 11
Name = LIBRARIES1.DATE Value = Jun 30 1998
Name = LIBRARIES1.NAME Value = IRL
Name = LIBRARIES1.TIME Value = 11:31:06
Name = LIBRARIES1.VERSION Value = 100.013.001
Name = LIBRARIES10.DATE Value = Jun 30 1998
Name = LIBRARIES10.NAME Value = DPR
Name = LIBRARIES10.TIME Value = 11:48:16
Name = LIBRARIES10.VERSION Value = 400.098.001
Name = LIBRARIES11.DATE Value = Aug  5 1998
Name = LIBRARIES11.NAME Value = PDF
Name = LIBRARIES11.TIME Value = 16:02:25
Name = LIBRARIES11.VERSION Value = 400.098.010
Name = LIBRARIES2.DATE Value = Jun 26 1998
Name = LIBRARIES2.NAME Value = IRP
Name = LIBRARIES2.TIME Value = 18:10:35
Name = LIBRARIES2.VERSION Value = 100.013.001
Name = LIBRARIES3.DATE Value = Jun 26 1998
Name = LIBRARIES3.NAME Value = DQM
Name = LIBRARIES3.TIME Value = 18:11:31
Name = LIBRARIES3.VERSION Value = 100.013.001
Name = LIBRARIES4.DATE Value = Jun 26 1998
Name = LIBRARIES4.NAME Value = IBASE
Name = LIBRARIES4.TIME Value = 18:01:12
Name = LIBRARIES4.VERSION Value = 100.013.001
Name = LIBRARIES5.DATE Value = Jun 26 1998
Name = LIBRARIES5.NAME Value = DCB
Name = LIBRARIES5.TIME Value = 18:06:22
Name = LIBRARIES5.VERSION Value = 100.013.001
Name = LIBRARIES6.DATE Value = Jun 30 1998
Name = LIBRARIES6.NAME Value = ATC
Name = LIBRARIES6.TIME Value = 11:29:22
Name = LIBRARIES6.VERSION Value = 100.013.001
Name = LIBRARIES7.DATE Value = Jun 29 1998
Name = LIBRARIES7.NAME Value = DSIJ
```

```
Name = LIBRARIES7.TIME Value = 17:50:06
Name = LIBRARIES7.VERSION Value = 100.013.001
Name = LIBRARIES8.DATE Value = Jun 26 1998
Name = LIBRARIES8.NAME Value = WFX
Name = LIBRARIES8.TIME Value = 17:52:36
Name = LIBRARIES8.VERSION Value = 100.013.001
Name = LIBRARIES9.DATE Value = Jun 30 1998
Name = LIBRARIES9.NAME Value = DSI
Name = LIBRARIES9.TIME Value = 11:36:19
Name = LIBRARIES9.VERSION Value = 100.013.001
Name = RESTARTCOUNT Value = 0
Name = RESULTS Value = SUCCESS
Name = SUCCESSCOUNT Value = 1
Name = UPTIME Value = Wed Aug 12 16:31:14 1998
```

DSICoTB, option ESS

Here is the output you should see when you execute the Visual Basic program, DSICoTB.EXE, option ESS.

LOG

```
InitSession
Submit: ESS
GetQueueRec
Term
----------------------- Done -----------------------
```

OUTPUT

```
ALLOCCOUNT9477
ERRORCOUNT0
FREECOUNT6791
LASTRESTARTWed Aug 12 16:48:37 1998
LIBRARIES11
LIBRARIES1.DATEJun 30 1998
LIBRARIES1.NAMEIRL
LIBRARIES1.TIME11:31:06
LIBRARIES1.VERSION100.013.001
LIBRARIES10.DATEJun 30 1998
LIBRARIES10.NAMEDPR
LIBRARIES10.TIME11:48:16
LIBRARIES10.VERSION400.098.001
LIBRARIES11.DATEAug  5 1998
LIBRARIES11.NAMEPDF
LIBRARIES11.TIME16:02:25
LIBRARIES11.VERSION400.098.010
LIBRARIES2.DATEJun 26 1998
LIBRARIES2.NAMEIRP
LIBRARIES2.TIME18:10:35
LIBRARIES2.VERSION100.013.001
LIBRARIES3.DATEJun 26 1998
LIBRARIES3.NAMEDQM
LIBRARIES3.TIME18:11:31
LIBRARIES3.VERSION100.013.001
LIBRARIES4.DATEJun 26 1998
LIBRARIES4.NAMEIBASE
LIBRARIES4.TIME18:01:12
```

```
LIBRARIES4.VERSION100.013.001
LIBRARIES5.DATEJun 26 1998
LIBRARIES5.NAMEDCB
LIBRARIES5.TIME18:06:22
LIBRARIES5.VERSION100.013.001
LIBRARIES6.DATEJun 30 1998
LIBRARIES6.NAMEATC
LIBRARIES6.TIME11:29:22
LIBRARIES6.VERSION100.013.001
LIBRARIES7.DATEJun 29 1998
LIBRARIES7.NAMEDSIJ
LIBRARIES7.TIME17:50:06
LIBRARIES7.VERSION100.013.001
LIBRARIES8.DATEJun 26 1998
LIBRARIES8.NAMEWFX
LIBRARIES8.TIME17:52:36
LIBRARIES8.VERSION100.013.001
LIBRARIES9.DATEJun 30 1998
LIBRARIES9.NAMEDSI
LIBRARIES9.TIME11:36:19
LIBRARIES9.VERSION100.013.001
MESSAGESMSG0002
RESTARTCOUNT1
RESULTSSUCCESS
SUCCESSCOUNT7
UPTIMEWed Aug 12 16:44:15 1998
```

DSICoTB, option Roll Your Own

Here is the output you should see when you execute the Visual Basic program, DSICoTB.EXE, option Roll Your Own.

LOG (left hand side)

```
InitSession
Submit
USERID  USERID
PASSWORDPASSWORD
CONFIG  INSURE
GetQueueRecord
GetAttachmentAll
------------------------ Done ------------------------
```

OUTPUT (right hand side)

```
CONFIGINSURE
PASSWORDPASSWORD
REPORTTOFORMAKER
RESULTSSUCCESS
RIGHTS9
SECURITY
USERIDUSERID
USRMESSAGE
```

DSICoTB, option RSS

Here is the output you should see when you execute the Visual Basic program, DSICoTB.EXE, option RSS.

LOG (left side of window)

```
InitSession
Submit: RSS
GetQueueRec
Term
----------------------- Done -----------------------
```

OUTPUT (right side of window)

```
ALLOCCOUNT12542
ERRORCOUNT0
FREECOUNT9867
LASTRESTARTWed Aug 12 16:48:37 1998
LIBRARIES11
LIBRARIES1.DATEJun 30 1998
LIBRARIES1.NAMEIRL
LIBRARIES1.TIME11:31:06
LIBRARIES1.VERSION100.013.001
LIBRARIES10.DATEJun 30 1998
LIBRARIES10.NAMEDPR
LIBRARIES10.TIME11:48:16
LIBRARIES10.VERSION400.098.001
LIBRARIES11.DATEAug  5 1998
LIBRARIES11.NAMEPDF
LIBRARIES11.TIME16:02:25
LIBRARIES11.VERSION400.098.010
LIBRARIES2.DATEJun 26 1998
LIBRARIES2.NAMEIRP
LIBRARIES2.TIME18:10:35
LIBRARIES2.VERSION100.013.001
LIBRARIES3.DATEJun 26 1998
LIBRARIES3.NAMEDQM
LIBRARIES3.TIME18:11:31
LIBRARIES3.VERSION100.013.001
LIBRARIES4.DATEJun 26 1998
LIBRARIES4.NAMEIBASE
LIBRARIES4.TIME18:01:12
LIBRARIES4.VERSION100.013.001
LIBRARIES5.DATEJun 26 1998
LIBRARIES5.NAMEDCB
LIBRARIES5.TIME18:06:22
LIBRARIES5.VERSION100.013.001
LIBRARIES6.DATEJun 30 1998
LIBRARIES6.NAMEATC
LIBRARIES6.TIME11:29:22
LIBRARIES6.VERSION100.013.001
LIBRARIES7.DATEJun 29 1998
LIBRARIES7.NAMEDSIJ
LIBRARIES7.TIME17:50:06
LIBRARIES7.VERSION100.013.001
```

```
LIBRARIES8.DATEJun 26 1998
LIBRARIES8.NAMEWFX
LIBRARIES8.TIME17:52:36
LIBRARIES8.VERSION100.013.001
LIBRARIES9.DATEJun 30 1998
LIBRARIES9.NAMEDSI
LIBRARIES9.TIME11:36:19
LIBRARIES9.VERSION100.013.001
MESSAGESMSG0001
RESTARTCOUNT2
RESULTSSUCCESS
SUCCESSCOUNT7
UPTIMEWed Aug 12 16:44:15 1998
```

DSICoTB, option SSS

Here is the output you should see when you execute the Visual Basic program, DSICoTB.EXE, option SSS.

LOG (left side of window)

```
InitSession
Submit: SSS
GetQueueRec
Term
----------------------- Done -----------------------
```

OUTPUT (right side of window)

```
ALLOCCOUNT9397
ERRORCOUNT0
FREECOUNT6720
LASTRESTARTWed Aug 12 16:48:37 1998
LIBRARIES11
LIBRARIES1.DATEJun 30 1998
LIBRARIES1.NAMEIRL
LIBRARIES1.TIME11:31:06
LIBRARIES1.VERSION100.013.001
LIBRARIES10.DATEJun 30 1998
LIBRARIES10.NAMEDPR
LIBRARIES10.TIME11:48:16
LIBRARIES10.VERSION400.098.001
LIBRARIES11.DATEAug  5 1998
LIBRARIES11.NAMEPDF
LIBRARIES11.TIME16:02:25
LIBRARIES11.VERSION400.098.010
LIBRARIES2.DATEJun 26 1998
LIBRARIES2.NAMEIRP
LIBRARIES2.TIME18:10:35
LIBRARIES2.VERSION100.013.001
LIBRARIES3.DATEJun 26 1998
LIBRARIES3.NAMEDQM
LIBRARIES3.TIME18:11:31
LIBRARIES3.VERSION100.013.001
LIBRARIES4.DATEJun 26 1998
LIBRARIES4.NAMEIBASE
```

```
LIBRARIES4.TIME18:01:12
LIBRARIES4.VERSION100.013.001
LIBRARIES5.DATEJun 26 1998
LIBRARIES5.NAMEDCB
LIBRARIES5.TIME18:06:22
LIBRARIES5.VERSION100.013.001
LIBRARIES6.DATEJun 30 1998
LIBRARIES6.NAMEATC
LIBRARIES6.TIME11:29:22
LIBRARIES6.VERSION100.013.001
LIBRARIES7.DATEJun 29 1998
LIBRARIES7.NAMEDSIJ
LIBRARIES7.TIME17:50:06
LIBRARIES7.VERSION100.013.001
LIBRARIES8.DATEJun 26 1998
LIBRARIES8.NAMEWFX
LIBRARIES8.TIME17:52:36
LIBRARIES8.VERSION100.013.001
LIBRARIES9.DATEJun 30 1998
LIBRARIES9.NAMEDSI
LIBRARIES9.TIME11:36:19
LIBRARIES9.VERSION100.013.001
RESTARTCOUNT1
RESULTSSUCCESS
SUCCESSCOUNT6
UPTIMEWed Aug 12 16:44:15 1998
Visited: http://
```

Appendix B

Error Messages

This appendix describes how you can customize the error messages you may receive while using IDS. For more information, see Displaying Error Messages on page 340.

This appendix also lists and explains error messages you may receive while using the Internet Document Server and any of the various bridges.

The messages are grouped in these categories:

- Internet Document Server Error Messages on page 344
- Documaker Bridge Error Messages on page 346
- Printstream Bridge Error Messages on page 351

Displaying Error Messages

The system includes an XML file which provides a template for all server and base rule error messages. It is up to you to customize this file if you use custom rules or if you want to modify the description of the problem. The file includes US English error descriptions, possible causes, and remedies.

You can find examples of how to use this XML file in the Docupresentment samples for ASP and JSP pages.

NOTE: CGI clients cannot use this file and must use HTML templates instead.

Here is an example of the XML file:

XML layout

```
<?xml version="1.0" encoding="UTF-8"?>
<ERRORCODES>
<CODE VALUE="ATC0001">
<PARAMETERS>
<VARIABLE/>
</PARAMETERS>
<SEVERITY>Error</SEVERITY>
<CATEGORY>Server configuration</CATEGORY>
<DESCRIPTION>Can not add variable <PARAMETER NAME="VARIABLE">
//ROWSET[@NAME="ATC0001"]//VAR[@NAME="VARIABLE"]</PARAMETER>
to the attachment</DESCRIPTION>
<CAUSE>Attachment size is larger than supported by queuing system
<REMEDY>Reduce the size of the attachment. Example, if search request
returns too many matches redefine search criteria so number of
matches is reduced.
</REMEDY>
</CAUSE>
<CAUSE>Server is running low on memory
<REMEDY>Restart server. If problem persists, report it to tech
support</REMEDY>
</CAUSE>
<CAUSE>Memory was corrupted by ill-behaved rule
<REMEDY>If problem persists, report it to tech support</REMEDY>
</CAUSE>
</CODE>
<CODE VALUE="ATC0002">
<PARAMETERS>
<APINAME/>
</PARAMETERS>
<SEVERITY>Error</SEVERITY>
<DESCRIPTION>The virtual memory management API <PARAMETER
NAME="APINAME">
//ROWSET[@NAME="ATC0002"]//VAR[@NAME="APINAME"]</PARAMETER>
failed</DESCRIPTION>
<CAUSE>Memory corruption on the server due to ill-behaved rules.
<REMEDY>If problem persists, report it to tech support</REMEDY>
</CAUSE>
</CODE>
<CODE VALUE="DPR0009">
<PARAMETERS/>
<SEVERITY>Warning</SEVERITY>
<CATEGORY>User error</CATEGORY>
```

```
<DESCRIPTION>No matches were found for the specified search
criteria</DESCRIPTION>
<CAUSE>Search criteria specified by the user resulted in no matches
found
<REMEDY>Specify different search criteria</REMEDY>
</CAUSE>
</CODE>
</ERRORCODES>
```

Keep in mind...

- All error codes are attributes of the CODE children of the ERRORCODES root element.
- The PARAMETERS child of the CODE element defines the parameters in the error message. In example above, the ATC0001 error code parameter passed with the error message is named VARIABLE.

  In the attachment variable name sense, the following attachment variables will be present with ATC0001 error: ATC0001 and ATC00011.VARIABLE. The second number one indicates a row set. You must insert the value of the VARIABLE into the placeholder specified in XML file.

  The ATC0001 is a row set in DSI SOAP XML message, so it can also be accessed as an element on the ROWSET XML tree. The placeholder is indicated with the XML element PARAMETER and the text of this element is an XPath you can use to pull the parameter value from the IDS XML SOAP message. See the following sample IDS message.
- The SEVERITY element defines the severity level such as: error, warning or info.
- The CATEGORY element defines where the error was generated and the most likely cause, such as: server configuration, bridge configuration, or user error.
- The DESCRIPTION element defines the information displayed to the end user with the parameter placeholders replaced by actual parameter values.
- The CAUSE element defines the probable cause of this error. Since it is possible to have multiple causes, the application should be able to display multiples CAUSE elements.
- The REMEDY element, which is a child of the CAUSE element, provides an explanation how you can correct the problem.

Client error handling

Usually the client submits a request to IDS and gets results. One of the attachment variables returned is RESULTS. This value contains the value SUCCESS or the error code.

Since you can have multiple errors, be sure to check RESULTS for the value SUCCESS. If it is not SUCCESS, the client code should examine the returned results for all error messages, not just the code provided in RESULTS attachment variable. In other words, the attachment variable RESULTS should be considered as a binary indication of successful transaction, it was either success or not.

Here is a sample IDS message with an error. This layout shows row set changes, available in IDS 1.8 and higher. Do not compare this layout with messages created by the older versions of IDS.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
envelope">
<SOAP-ENV:Body>
<DSIMSG VERSION="100.018.0">
<CTLBLOCK>
<UNIQUE_ID>9F2AE2BB6609450887779A836D90D390</UNIQUE_ID>
<REQTYPE>RPD</REQTYPE>
<USERID>USERNAME</USERID>
</CTLBLOCK>
<MSGVARS>
<ROWSET NAME="ERRORS">
<ROW NUM="1">
<VAR NAME="CODE">ATC0001</VAR>
</ROW>
</ROWSET>
<ROWSET NAME="ATC0001">
<ROW NUM="1">
<VAR NAME="VARIABLE">FILENAME</VAR>
</ROW>
</ROWSET>
<VAR NAME="RESULTS">ATC0001</VAR>
</MSGVARS>
</DSIMSG>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Error reporting for C exceptions

The reporting of C exceptions while running a rule has includes the following information:

- Before the restart, IDS reports the type of C Exception and the rule that triggered the exception. This is reported through IDS's regular logging techniques. Here is an example:

```
ERROR [BLP-0]: 2005-06-30 14:54:23,703 BusinessLogicProcessor The
thread tried to read from or write to a virtual address for which it
does not have the appropriate access.
ERROR [BLP-0]: 2005-06-30 14:54:23,718 RequestDescription Rule
'tstw32.dll->TSTTestBlowUP' had an exception in Run Forward message.
```

- After IDS restarts, it sends a message to the client program through the queues to report there was a problem. This is reported via the usual rowset/error message way of reporting errors. Here is an example:

```
Result returned back from server

Message variables
RESULTS=SRV0004

Rowsets
ERRORS
Row 1
CODE=SRV0004
SRV0004
Row 1
CURMSG=Server failed to process the request <TSTLIB> due to a fatal
error.
```

Internet Document Server Error Messages

Here is a list of the error messages you may receive.

Code	Message Text
ATC0001	Cannot add variable #VARIABLE,# to the attachment list.
ATC0002	The virtual memory management API #APINAME,%s# failed.
ATC0003	The attachment variable #VARNAME,%s# could not be located.
IRL0001	The required attachment variable #VARIABLE,%s# could not be located.
IRL0002	No search criteria were specified. The attachment variable FIELDS in empty.
IRL0003	The user information database, #FILENAME,%s#, could not be opened.
IRL0004	The user ID #USERID,%s# is invalid.
IRL0005	The password specified for user #USERID,%s# is incorrect.
IRL0006	The virtual memory management API #APINAME,%s# failed.
IRL0007	The INI option #INIOPTION,%s# could not be located in the group #INIGROUP,%s#.
IRL0008	The database API #APINAME,%s# failed accessing the table #TABLENAME,%s#.
IRL0009	No matches were found for the search criteria.
IRL0010	The system encountered an internal error of unknown type. The call by #LOCATION,%s# to #APINAME,%s# failed.
IRL0011	The attachment field #VARIABLE,%s# does not contain valid data.
IRL0012	The queue management API #APINAME,%s# failed.
IRL0013	The initialization file, #FILENAME,%s# could not be loaded.
IRL0014	Platform error. Area: #AREA,# Code: #CODE,# Code2: #CODE2,# Message: #MESSAGE,#
IRL0015	The parameter #PARAMETER,# is invalid.
IRL0016	The value #VALUE,# was not found for option #OPTION,# in group #GROUP,#.
IRL0017	Cannot locate variable #VARIABLE,#.
IRL0022	Cannot add variable #VARIABLE,# to the attachment list.
IRL0023	Cannot save changes to the file #FILE,#.
IRL0025	Cannot add variable #VARIABLE,# to the attachment record #RECORD,#.
RL0026	Cannot find the global variable #VARIABLE,#. Make sure the IRLInitFTP rule is registered on INI request.
IRL0027	Rule parameters for rule #RULENAME,# are incorrect. The rule is disabled.

Code	Message Text
IRL0028	TP server is not specified in the attachment or in the INI file. The FTP rule is disabled.
IRL0029	FTP connection cannot be established. Make sure the FTP rule is configured correctly.
IRL0030	Cannot find variable #VARIABLE,# in the attachment. FTP operation #OPERATION,# will be skipped.
IRL0031	Error #ERRORCODE,# #ERRORDESCRIPTION,# on FTP operation #OPERATION,#.
SRV0001	The Rule Processor failed while processing the messages #CURMSG,#.
SRV0002	The server is not configured for request type: #REQTYPE,#.
SRV0003	The master server could not send the message to IDS.
SRV0004	Server failed to process the request due to a fatal error.
SRV0005	Server failed to process the request due to a time-out. You must enable the long transaction detection feature for the SRV0005 error to work. To enable this message, set the client time-out longer than the Watchdog timer value on IDS side.

DOCUMAKER BRIDGE ERROR MESSAGES

Here is a list of the error messages you may receive.

Code	Message Text
ATC0001	Cannot add variable #VARIABLE,# to the attachment list.
ATC0002	The virtual memory management API #APINAME,# failed.
ATC0003	The attachment variable #VARNAME,# could not be located.
DPR0001	Cannot locate variable #VARIABLE,# in the attachment list.
DPR0002	No search criteria was specified. The attachment variable FIELDS is empty.
DPR0003	The user information database, #FILENAME,#, could not be opened.
DPR0004	The user ID #USERID,# is invalid.
DPR0005	The password specified for user #USERID,# is incorrect.
DPR0006	The virtual memory management API #APINAME,# failed.
DPR0007	The INI option #INIOPTION,# could not be located in the group #INIGROUP,#.
DPR0008	The database API #APINAME,# failed accessing the table #TABLENAME,#.
DPR0009	No matches were found for the search criteria.
DPR0010	The system encountered an internal error of unknown type. The call by #LOCATION,# to #APINAME,# failed.
DPR0011	The attachment field #VARIABLE,# does not contain valid data.
DPR0013	The initialization file, #FILENAME,# could not be loaded.
DPR0012	The database API #APINAME,# cannot locate the table #TABLENAME,#.
DPR0014	Platform error. Area: #AREA,# Code: #CODE,# Code2: #CODE2,# Message: #MESSAGE,#
DPR0015	FAP Version is not in sync. #LOCATION,#.
DPR0016	Failed to unload template file #FILE,#.
DPR0017	Cannot locate variable #VARIABLE,#.
DPR0018	Cannot load font cross reference file #PATH,##FILE,##EXTENSION,#
DPR0019	Cannot retrieve data into the #FILE,# file. ARCRetrieveDoc API failed. CATALOGKEY =#CATALOGKEY,# CARID=#CARID,#. #PATH,##FILE,##EXTENSION,#
DPR0020	DSLoadFormList API failed on file #FILE,#
DPR0021	DSLoadNAFormset API failed on file #FILE,#
DPR0022	Cannot add variable #VARIABLE,# to the attachment list.
DPR0024	Cannot create DSI variable #VARIABLE,#.

Code	Message Text
DPR0025	Cannot add variable #VARIABLE,# to the attachment record #RECORD,#.
DPR0026	Cannot load the import file #FILE,#.
DPR0027	Cannot load the form set definition file #FILE,#.
DPR0028	Cannot load the FAP file for image #IMAGE,#.
DPR0029	Loading of the provided import file resulted in empty form set.
DPR0030	The combination of group names (#GROUPNAME1,#) and (#GROUPNAME2,#) and the form name (#FORMNAME,#) does not exist in form definition file. Invalid import data.
DPR0031	Cannot open import file #FILE,#.
DPR0032	No form name provide in the import file.
DPR0033	Cannot parse import file. Line: #LINEDATA,#
DPR0034	The combination of group names (#GROUPNAME1,#) and (#GROUPNAME2,#) and the form name (#FORMNAME,#) and image name (#IMAGENAME,#) does not exist in form definition file. Invalid import data.
DPR0035	Cannot open the export file #FILENAME,#. Error reported by OS #ERRORNO,#
DPR0036	The DSI variable #VARIABLE,# does not contain a valid FAP form set.
DPR0037	The attachment variable #VARIABLE,# with value #VALUE,# is not a valid encrypted string.
DPR0038	The rule parameters #PARAMETERS,# for the rule #RULE,# are not correct or empty.
DPR0039	The call by #LOCATION,# to API #APINAME,# failed.
DPR0040	DSI variable #VARIABLE,# does not contain valid data.
DPR0041	The call by #LOCATION,# to API #APINAME,# failed. The status code has been changed by another user.
DPR0042	Failed to unload File #FILE,# in #LOCATION,#.
DPR0043	Failed to DBQueryFormatInfo from #FILE,#.
DPR0044	Failed to DBInitializeFile #FILE,#.
DPR0045	Failed to DBOpen #FILE,#.
DPR0046	Failed to UTLLockARC #FILE,#.
DPR0047	Failed to ArcInit #FILE,#.
DPR0048	Failed to ArcArchiveDataFile #FILE,#.
DPR0049	Failed to create the XML document in #LOCATION,#.
DPR0050	Failed to export the form set to XML in #LOCATION,#.
DPR0051	Failed to unload the XML file #FILE,# in #LOCATION,#.

Code	Message Text
DPR0052	Failed to decrypt the attachment variable #VARIABLE,# in the wild card search.
DPR0053	Unable to get a random seed value in #LOCATION,#.
DPR0054	Invalid login.
DPR0055	Unable to DSIGlobalDataCreate in #LOCATION,#. Most likely the global data setup is incorrect.
IRL0001	The required attachment variable #VARIABLE,%s# could not be located.
IRL0002	No search criteria were specified. The attachment variable FIELDS in empty.
IRL0003	The user information database, #FILENAME,%s#, could not be opened.
IRL0004	The user ID #USERID,%s# is invalid.
IRL0005	The password specified for user #USERID,%s# is incorrect.
IRL0006	The virtual memory management API #APINAME,%s# failed.
IRL0007	The INI option #INIOPTION,%s# could not be located in the group #INIGROUP,%s#.
IRL0008	The database API #APINAME,%s# failed accessing the table #TABLENAME,%s#.
IRL0009	No matches were found for the search criteria.
IRL0010	The system encountered an internal error of unknown type. The call by #LOCATION,%s# to #APINAME,%s# failed.
IRL0011	The attachment field #VARIABLE,%s# does not contain valid data.
IRL0012	The queue management API #APINAME,%s# failed.
IRL0013	The initialization file, #FILENAME,%s# could not be loaded.
IRL0014	Platform error. Area: #AREA,# Code: #CODE,# Code2: #CODE2,# Message: #MESSAGE,#
IRL0015	The parameter #PARAMETER,# is invalid.
IRL0016	The value #VALUE,# was not found for option #OPTION,# in group #GROUP,#.
IRL0017	Cannot locate variable #VARIABLE,#.
IRL0022	Cannot add variable #VARIABLE,# to the attachment list.
IRL0023	Cannot save changes to the file #FILE,#.
IRL0025	Cannot add variable #VARIABLE,# to the attachment record #RECORD,#.
RL0026	Cannot find the global variable #VARIABLE,#. Make sure the IRLInitFTP rule is registered on INI request.
IRL0027	Rule parameters for rule #RULENAME,# are incorrect. The rule is disabled.

Code	Message Text
IRL0028	TP server is not specified in the attachment or in the INI file. The FTP rule is disabled.
IRL0029	FTP connection cannot be established. Make sure the FTP rule is configured correctly.
IRL0030	Cannot find variable #VARIABLE,# in the attachment. FTP operation #OPERATION,# will be skipped.
IRL0031	Error #ERRORCODE,# #ERRORDESCRIPTION,# on FTP operation #OPERATION,#.
MTC0001	The attachment variable #VARIABLE,# could not be located.
MTC0010	The system encountered an internal error of unknown type. The call by #LOCATION,# to #APINAME,# failed.
MTC0011	The attachment field #VARIABLE,# does not contain valid data.
MTC0014	Platform error. Area: #AREA,# Code: #CODE,# Code2: #CODE2,# Message: #MESSAGE,#
MTC0017	Cannot locate variable #VARIABLE,#.
MTC0018	Cannot load font cross reference file #PATH,##FILE,##EXTENSION,#
MTC0022	Cannot add variable #VARIABLE,# to the attachment list.
SRV0001	The Rule Processor failed while processing the messages #CURMSG,#.
SRV0002	The server is not configured for request type: #REQTYPE,#.
SRV0003	The master server could not send the message to IDS
TPD0001	Cannot DSILocateValue #NAME,#. Make sure TPDInit rule was executed.
TPD0002	Call to API #APINAME,# failed.
TPD0003	File #TIFFNAME,# cannot be loaded.
TPD0004	Stem variable #NAME,# cannot be located.
TPD0005	Stem variable #NAME,# has invalid value (0).

Java Error Messages

Here is a list of the error messages you may receive.

Code	Message Text
JAV0001	JNIAPI #APINAME,# Failed.
JAV0002	Cannot locate the global variable #VARIABLE,#.
JAV0003	Cannot create the global variable #VARIABLE,#.
JAV0004	Cannot parse the rule PARM #VARIABLE,#.
JAVARULE001	Cannot find the argument file #ARGUMENTS,#.
JAVARULE002	Problem using the argument file #ARGUMENTS,#. #MESSAGE,#
JAVARULE003	Cannot locate the attachment variable #ATTACHVAR,# for parsing arguments.
JAVADOM0001	A DSI exception was returned from Java. The error message is #ERRORMESSAGE,#
JAVADOM0002	Cannot find the attachment variable #ATTACHVAR,#. This variable contains the name of the XML file.
JAVADOM0004	Error changing document. Error code returned is #ERRORCODE,#. Error message is #ERRORMESSAGE,#
JAVADOM0005	Java class #DOMCLASS,# specified in "DOMCLASS" not an instance of Java interface DOMTransformer.
JAVADOM0006	An exception was returned from Java. The error message is #ERRORMESSAGE,#
JAVADOM0007	The Java class "#CLASS,#" was not found. Check the UserClassPath setting.
JAVAXSL0001	Error during run forward message.
JAVAXSL0002	Error getting XSL filters. Error code returned is #ERRORCODE,#. Error message is #ERRORMESSAGE,#
JAVAXSL0003	A DSI exception was returned from Java. The error message is #ERRORMESSAGE,#
JAVAXSL0004	An exception was returned from Java. The error message is #ERRORMESSAGE,#
JAVAXSL0005	Java class #PROVIDERCLASS,# specified in "#PROVIDERCLASS" not an instance of Java interface XSLProfilesProvider.
JAVAXSL0006	An exception was returned from Java during initialization. The error message is #ERRORMESSAGE,#
JAVAXSL0007	The Java class "#CLASS,#" was not found. Check the UserClassPath setting.

Printstream Bridge Error Messages

Here is a list of the error messages you may receive.

Code	Message Text
MTC0001	The attachment variable #VARNAME,%s# could not be located.
MTC0010	The system encountered an internal error of unknown type. The call by #LOCATION,%s# to #APINAME,%s# failed.
MTC0011	The attachment field #VARIABLE,%s# does not contain valid data.
MTC0014	Platform error. Area: #AREA,# Code: #CODE,# Code2: #CODE2,# Message: #MESSAGE,#
MTC0017	Cannot locate variable #VARIABLE,#.
MTC0018	Cannot load font cross reference file #PATH,##FILE,##EXTENSION,#
MTC0022	Cannot add variable #VARIABLE,# to the attachment list.

AFP Error Messages

The following information describes how to handle error messages you may encounter while using the Printstream Bridge and AFP print streams.

Character set xxxxxxxx not found…

If you receive this error message, the AFP print stream uses a character set and code page file name instead of coded font file name to specify an AFP font to be used. In this case, you will need to create an additional file called IBMXREF.TBL to provide additional AFP font information. IBMXREF.TBL is a text file that contains pairs of coded font names and character set names. This file should be placed in the directory specified by the FORMLIB INI setting.

What you are doing is specifying the coded font file name to use when a reference to the character set file is encountered in the AFP print stream. The system searches in the FXR file for the coded font file name to determine font information it needs during the PDF conversion.

When entering the coded font and character set names in IBMXREF.TBL, do not use the first two letters (X0, X1, C0, C1, and so on). The coded font and character set names need to be written in *UPPER CASE* and separated by at least one space. Each pair of coded font and character set names should be written on separate lines. For example, if you receive an error stating…

```
Character set C0AR111 not found…
```

Add a line of coded font and character set names to the IBMXREF.TBL. If a coded font file named *X0AR11P* contained a reference to the character set file *C0AR111*, you would add the following line to IBMXREF.TBL:

```
AR11P AR111
```

Notice the first two letters of *X0AR11P* and *C0AR111* are omitted from the line added to IBMXREF.TBL. You should have inserted the coded font file, *X0AR11P*, into the FXR file previously.

If you have character set files but do not have any coded font files, you can insert a character file into the FXR. However, you must edit the font inserted into the FXR and specify a coded font file name on the Printers page. In this case, use the character set name as the coded font name and change the first letter from *C* to *X*. In this case, the pair of names stored in the IBMXREF.TBL will be the same.

If you have a character set file that is used by more than one code page file, you can map each character set/code page file combination to a coded font file named in the FXR. To do this, add a third column to the IBMXREF.TBL. The third column contains the name of code page file. For example, to map the coded font file, *X0AR11P*, to the character set and code page files, *C0AR111*, and *T1ISI121*, you would add this line to IBMXREF.TBL:

```
AR11P AR111 T1ISI121
```

Notice the first two letters of *X0AR11P* and *C0AR111* are omitted from the line added to IBMXREF.TBL but the full name of the code page file, *T1ISI121*, is used.

Error opening overlay: xxxxxxxx

If you receive this error message, the AFP print stream uses an overlay that the system could not find. Notice the path and file extension of the overlay specified in the error message. Make sure your AFP overlay is stored in the proper directory and contains the expected file extension.

Error opening page segment: xxxxxxxx

If you receive this error message, the AFP print stream uses a page segment that the system could not find. Notice the path and file extension of the page segment specified in the error message. Make sure your AFP page segment is stored in the proper directory and contains the expected file extension.

Error opening logo: xxxxxxxx

If you receive this error message, it is likely that the AFP print stream uses a page segment that the system could not find. If so, you would have received an error message for the missing page segment as well. Correct the problem with the missing page segment and this error should disappear as well.

APPENDIX C

Choosing a Paper Size

The system supports a wide variety of paper sizes including US and international sizes. The following tables show the paper sizes you can choose from:

- US Standard Sizes on page 356
- ISO Sizes on page 357
- Japanese Standard Sizes on page 360

You can also find the following related information in this topic:

- Printer Support for Paper Sizes on page 361
- Paper Sizes for AFP Printers on page 365

US Standard Sizes

These paper sizes are commonly used in the United States and Canada. The height and width are in FAP units (2400 per inch), millimeters, and inches. The inch dimensions are approximate.

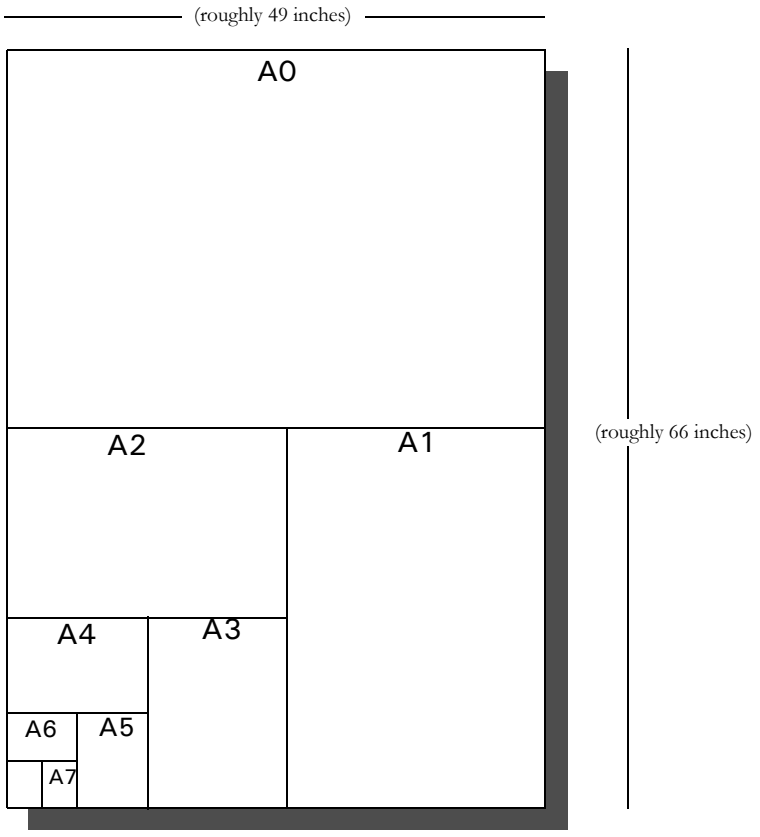
		Width x Height		
Name	Code	FAP units	Millimeters	Inches (approximate)
US letter	0	20400 x 26400	216 × 279	8½ x 11
US legal	1	20400 x 33600	216 × 356	8½ x 14
US executive	3	17400 x 25200	190 × 254	7¼ 10½
US ledger	4	40800 x 26400	432 x 279	17 x 11
US tabloid	5	26400 x 40800	279 × 432	11 x 17
US statement	6	13200 x 20400	140 x 216	5½ x 8½
US folio	7	20400 x 31200	216 x 330	8½ x 13
US fanfold	8	35700 x 26400	378 x 279	14⅞ x 11
Custom	98	any x any	any x any	any x any

ISO Sizes

The International Organization for Standardization (ISO) paper sizes, which are based on the earlier Deutsche Industrie Norm (DIN) sizes, are used throughout the world except in Canada, the United States, and Japan. There are three main series of paper sizes: A, B, and C.

ISO A sizes

The A series of sizes are typically used for correspondence, books, brochures, and other printed materials. This diagram shows most of the various A sizes. The height and width are in FAP units (2400 per inch), millimeters, and inches. The inch dimensions are approximate.



Width x Height

Name	Code	FAP units	Millimeters	Inches (approximate)
ISO A0	20	79464 x 112345	841 x 1189	33 1/8 x 46 1/4
ISO A1	21	56125 x 79464	594 x 841	23 3/8 x 33 1/8
ISO A2	22	39685 x 56125	420 x 594	16 1/2 x 23 3/8
ISO A3	23	28063 x 39685	297 x 420	11 3/4 x 16 1/2
ISO A4	2	19842 x 28063	210 x 297	8 1/4 x 11 3/4

		Width x Height		
Name	Code	FAP units	Millimeters	Inches (approximate)
ISO A5	25	13984 x 19842	148 x 210	5 7/8 x 8 1/4
ISO A6	26	9921 x 13984	105 x 148	4 1/8 x 5 7/8
ISO A7	27	6992 x 9921	74 x 105	2 7/8 x 4 1/8
ISO A8	28	4913 x 6992	52 x 74	2 x 2 7/8
ISO A9	29	3496 x 4913	37 x 52	1 1/2 x 2
ISO A10	30	2457 x 3496	26 x 37	1 x 1 1/2
ISO 2A	32	112345 x 158927	1189 x 1682	46 3/4 x 66 1/4
ISO 4A	34	158927 x 224690	1682 x 2378	66 1/4 x 93 5/8

ISO B sizes

The B series of sizes are designed primarily for posters, wall charts, and similar items where the difference between each A size represents too large a jump. The height and width are in FAP units (2400 per inch), millimeters, and inches. The inch dimensions are approximate.

		Width x Height		
Name	Code	FAP units	Millimeters	Inches (approximate)
ISO B0	40	94487 x 133605	1000 x 1414	39 1/8 x 55 1/8
ISO B1	41	66802 x 94487	707 x 1000	27 7/8 x 39 1/8
ISO B2	42	47244 x 66802	500 x 707	19 5/8 x 27 7/8
ISO B3	43	33354 x 47244	353 x 500	13 7/8 x 19 5/8
ISO B4	44	23622 x 33354	250 x 353	9 7/8 x 13 7/8
ISO B5	45	16630 x 23622	176 x 250	7 x 9 7/8
ISO B6	46	11811 x 16630	125 x 176	5 x 7
ISO B7	47	8315 x 11811	88 x 125	3 1/2 x 5
ISO B8	48	5858 x 8315	62 x 88	2 1/2 x 3 1/2
ISO B9	49	4157 x 5858	44 x 62	1 3/4 x 2 1/2
ISO B10	50	2929 x 4157	31 x 44	1 1/4 x 1 3/4
ISO 2B	52	133605 x 188974	1414 x 2000	55 3/4 x 78 3/4
ISO 4B	54	188974 x 267209	2000 x 2828	78 3/4 x 111 1/4

ISO C sizes

The C series of sizes are designed for making envelopes and folders to take the A series of sizes. The height and width are in FAP units (2400 per inch), millimeters, and inches. The inch dimensions are approximate.

		Width x Height		
Name	**Code**	**FAP units**	**Millimeters**	**Inches (approximate)**
ISO C0	60	86645 x 122550	917 x 1297	36 1/8 x 51
ISO C1	61	61228 x 86645	648 x 917	25½ x 36
ISO C2	62	43275 x 61228	458 x 648	18 x 25½
ISO C3	63	30614 x 43275	324 x 458	12¾ x 18
ISO C4	64	21638 x 30614	229 x 324	9 x 12¾
ISO C5	65	15307 x 21638	162 x 229	6 3/8 x 9
ISO C6	66	10772 x 15307	114 x 162	4½ x 6 3/8
ISO C7	67	7653 x 10772	81 x 114	3¼ x 4½
ISO C8	68	5386 x 7653	57 x 81	2¼ x 3¼
ISO C9	69	3779 x 5386	40 x 57	1 5/8 x 2¼
ISO C10	70	2646 x 3779	28 x 40	1 1/8 x 1 5/8
ISO DL	71	10394 x 20787	110 × 220	4 1/3 x 8 2/3

The DL size is for a sheet 1/3 of the A4 size. This is the most common size of envelope.

Japanese Standard Sizes

Japan has its own standard paper sizes, called the Japan Industrial Standard (JIS). The JIS A series is identical in size to the ISO A series. The JIS B series, however, does not match the ISO B series. There is no equivalent to the ISO C series. This table shows the JIS paper sizes. The height and width are in FAP units (2400 per inch), millimeters, and inches. The inch dimensions are approximate.

		Width x Height		
Name	Code	FAP units	Millimeters	Inches (approximate)
JIS B0	80	97322 x 137573	1030 x 1456	40½ x 57¼
JIS B1	81	68787 x 97322	728 x 1030	28¾ x 40½
JIS B2	82	48661 x 68787	515 x 728	20¼ x 28¾
JIS B3	83	34393 x 48661	364 x 515	14¼ x 20¼
JIS B4	84	24283 x 34393	257 x 364	10⅛ x 14¼
JIS B5	85	17197 x 24283	182 x 257	7¼ x 10⅛
JIS B6	86	12094 x 17197	128 x 182	5 x 7¼
JIS B7	87	8598 x 12094	91 x 128	3½ x 5
JIS B8	88	6047 x 8598	64 x 91	2½ x 3½
JIS B	89	4252 x 6047	45 x 64	1¾ x 2½
JIS B10	90	3024 x 4252	32 x 45	1¼ x 1¾

Printer Support for Paper Sizes

This table outlines the various paper sizes supported by the different print drivers. The table includes information for the PDF, RTF, HTML, Metacode, PCL 5, PCL 6, GDI, PostScript, and AFP print drivers. The PDF, RTF, HTML, and Metacode print drivers support all paper sizes.

Paper size	PDF, RTF, HTML, and Metacode	PXL[1]	PCL[2]	GDI[2]	PST[3]	AFP[4]
US letter	X	X	X	X	X	X
US Legal	X	X	X	X	X	X
US executive	X	X	X	X	X	X
US ledger	X	X	X	X	X	X
US tabloid	X	Y	US letter	X	X	X
US statement	X	JIS B5	US executive	X	X	X
US folio	X	US legal	US legal	X	X	X
US fanfold	X	US ledger	US ledger	X	X	X
ISO 4A	X	Y	US letter	US letter	US letter	C
ISO 2A	X	Y	US letter	US letter	US letter	C
ISO A0	X	Y	US letter	US letter	X	C
ISO A1	X	Y	US letter	US letter	X	C
ISO A2	X	Y	US letter	US letter	X	C
ISO A3	X	X	X	X	X	X
ISO A4	X	X	X	X	X	X

Sizes marked with an *X* are fully supported by the corresponding driver.

Sizes marked with a *Y* are supported by sending the paper dimensions in millimeters to the printer.

Sizes that refer to another size substitute the referred size when *paper size matching* is turned on. If paper size matching is not turned on, the behavior depends upon the specific driver. To turn on paper size matching, use this INI option:

```
< PrtType:XXX >
    PaperSizeMatching = Yes
```

[1] When paper size matching is not turned on, the PCL 6 (PXL) driver sends the paper dimensions in millimeters to the printer.

[2] When paper size matching is not turned on, these drivers substitute US letter.

[3] This driver does not use paper size matching. US letter is substituted for the unsupported paper sizes

[4] Sizes marked with a C are supported, but are commented out of the AFP formdef source file called F1FMMST.DAT, See Paper Sizes for AFP Printers on page 365 for more information.

Paper size	PDF, RTF, HTML, and Metacode	PXL[1]	PCL[2]	GDI[2]	PST[3]	AFP[4]
ISO A5	X	X	X	X	X	X
ISO A6	X	X	X	X	X	X
ISO A7	X	ISO A6	ISO C5	ISO A6	X	C
ISO A8	X	ISO A6	ISO C5	ISO A6	X	C
ISO A9	X	ISO A6	ISO C5	ISO A6	X	C
ISO A10	X	ISO A6	ISO C5	ISO A6	X	C
ISO 4B	X	Y	US letter	US letter	US letter	C
ISO 2B	X	Y	US letter	US letter	US letter	C
ISO B0	X	Y	US letter	US letter	X	C
ISO B1	X	Y	US letter	US letter	X	C
ISO B2	X	Y	US letter	US letter	X	C
ISO B3	X	Y	US letter	US letter	X	C
ISO B4	X	JIS B4	US ledger	X	X	X
ISO B5	X	JIS B5	X	X	X	X
ISO B6	X	JIS B6	ISO C5	X	X	X
ISO B7	X	ISO A6	ISO C5	ISO A6	X	C
ISO B8	X	ISO A6	ISO C5	ISO A6	X	C
ISO B9	X	ISO A6	ISO C5	ISO A6	X	C

Sizes marked with an *X* are fully supported by the corresponding driver.

Sizes marked with a *Y* are supported by sending the paper dimensions in millimeters to the printer.

Sizes that refer to another size substitute the referred size when *paper size matching* is turned on. If paper size matching is not turned on, the behavior depends upon the specific driver. To turn on paper size matching, use this INI option:

```
< PrtType:XXX >
    PaperSizeMatching = Yes
```

[1] When paper size matching is not turned on, the PCL 6 (PXL) driver sends the paper dimensions in millimeters to the printer.

[2] When paper size matching is not turned on, these drivers substitute US letter.

[3] This driver does not use paper size matching. US letter is substituted for the unsupported paper sizes

[4] Sizes marked with a C are supported, but are commented out of the AFP formdef source file called F1FMMST.DAT, See Paper Sizes for AFP Printers on page 365 for more information.

Paper size	PDF, RTF, HTML, and Metacode	PXL[1]	PCL[2]	GDI[2]	PST[3]	AFP[4]
ISO B10	X	ISO A6	ISO C5	ISO A6	X	C
ISO C0	X	Y	US letter	US letter	X	C
ISO C1	X	Y	US letter	US letter	X	C
ISO C2	X	Y	US letter	US letter	X	C
ISO C3	X	Y	US letter	X	X	C
ISO C4	X	JIS B4	US ledger	X	X	C
ISO C5	X	X	X	X	X	C
ISO C6	X	JIS B6	ISO C5	X	X	C
ISO C7	X	ISO A6	ISO C5	ISO A6	X	C
ISO C8	X	ISO A6	ISO C5	ISO A6	US letter	C
ISO C9	X	ISO A6	ISO C5	ISO A6	US letter	C
ISO C10	X	ISO A6	ISO C5	ISO A6	US letter	C
ISO DL	X	X	X	X	X	X
JIS B0	X	Y	US letter	US letter	X	C
JIS B1	X	Y	US letter	US letter	X	C
JIS B2	X	Y	US letter	US letter	X	C
JIS B3	X	Y	US letter	US letter	X	C
JIS B4	X	X	X	US fanfold	X	X

Sizes marked with an *X* are fully supported by the corresponding driver.

Sizes marked with a *Y* are supported by sending the paper dimensions in millimeters to the printer.

Sizes that refer to another size substitute the referred size when *paper size matching* is turned on. If paper size matching is not turned on, the behavior depends upon the specific driver. To turn on paper size matching, use this INI option:

```
< PrtType:XXX >
    PaperSizeMatching = Yes
```

[1] When paper size matching is not turned on, the PCL 6 (PXL) driver sends the paper dimensions in millimeters to the printer.

[2] When paper size matching is not turned on, these drivers substitute US letter.

[3] This driver does not use paper size matching. US letter is substituted for the unsupported paper sizes

[4] Sizes marked with a C are supported, but are commented out of the AFP formdef source file called F1FMMST.DAT, See Paper Sizes for AFP Printers on page 365 for more information.

Paper size	PDF, RTF, HTML, and Metacode	PXL[1]	PCL[2]	GDI[2]	PST[3]	AFP[4]
JIS B5	X	X	X	X	X	X
JIS B6	X	X	X	X	X	X
JIS B7	X	ISO A6	ISO C5	ISO A6	X	C
JIS B8	X	ISO A6	ISO C5	ISO A6	X	C
JIS B9	X	ISO A6	ISO C5	ISO A6	X	C
JIS B10	X	ISO A6	ISO C5	ISO A6	X	C

Sizes marked with an *X* are fully supported by the corresponding driver.

Sizes marked with a *Y* are supported by sending the paper dimensions in millimeters to the printer.

Sizes that refer to another size substitute the referred size when *paper size matching* is turned on. If paper size matching is not turned on, the behavior depends upon the specific driver. To turn on paper size matching, use this INI option:

```
< PrtType:XXX >
    PaperSizeMatching = Yes
```

[1] When paper size matching is not turned on, the PCL 6 (PXL) driver sends the paper dimensions in millimeters to the printer.

[2] When paper size matching is not turned on, these drivers substitute US letter.

[3] This driver does not use paper size matching. US letter is substituted for the unsupported paper sizes

[4] Sizes marked with a C are supported, but are commented out of the AFP formdef source file called F1FMMST.DAT, See Paper Sizes for AFP Printers on page 365 for more information.

Paper Sizes for AFP Printers

The AFP formdef source file (F1FMMST.DAT) contains support for the following paper sizes, but since this file contains support for so many paper sizes, its size could affect printer performance. To limit the effect, some of the paper sizes are commented out, as shown in this table:

Size	Commented out?
Letter	No
Legal	No
Executive	No
Ledger	Yes
Tabloid	Yes
Statement	Yes
Folio	Yes
Fanfold	Yes
ISO A3	Yes
ISO A4	No
ISO A5	Yes
ISO A6	Yes
ISO B4	Yes
ISO B5	Yes
ISO B6	Yes
ISO DL	Yes
JIS B4	Yes
JIS B5	Yes
JIS B6	Yes

NOTE: The F1FMMST.DAT and F1FMMST.FDF files can be found in the FMRES master resource library (MRL).

The commented source line begins with an asterisk (*). To add support for another paper size, open the F1FMMST.DAT file and delete the asterisk at the beginning of each line that references a paper size you want to add.

Because the AFP formdef is composed on medium map names that specify page orientation, paper size, tray selection, and duplex settings, there are 31 groups of medium map settings. Each of these groups contains the 57 possible paper sizes. So, for each paper size you add, there are 31 sources lines you must *uncomment* to fully support a paper size for all orientations, trays, and duplex settings.

After you uncomment the lines that reference the paper size you want to add, run the AFPFMDEF utility to rebuild your AFP formdef file with the new information. For more information on this utility, see the Docutoolbox Reference.

Index

A

A4 page size
    PaperSize option 211
AbsolutePage property 167
Acrobat
    downloading Acrobat Reader 4
Acrobat Reader
    base fonts 215
    embedded fonts 223
    fonts 226
Active Server Pages 12, 25, 168
ActiveX Data Objects 166
AddNameValuePair method 317
AddReq method 18
AFEAssignDpw API 303
AFGJOB.JDT file 151
AFP
    error messages 352
AGFA fonts
    embedding 215
    international characters 229
AllowColorSheetLink option 231
AllowInput option 181, 231
AltFrom option 137, 138
ANSI code page 229
AppIdx INI option
    multiple bridges 130
ArcRet control group
    multiple bridges 130

ASP
IDSASP object 17
Print Preview 245
sample files 25
samples 25
showing PDF files 22
ATCReceiveFile
referencing attachment variables 106
ATCSendFile
referencing attachment variables 106
attachment fields
sending and receiving 19
attachment variables
referencing 106
attachments
distributing email 137, 138
REPLYTO 138
ATTCHDFD.DFD file 136, 139, 140
authentication
for Print Preview 250

B

Barcode option 180
base fonts
Acrobat Reader 216
defined 226
BaseErrors option 149
batch active flag 220
batch requests
submitting 132
BatchPrint control group 222
Bin2Unicode method 317
Bitmap option 180
bitmaps
color 214
BmSub option 233
BmSubChar option 233
BOF property 168
Bookmark option
custom bookmarks 227
PDF printers 209
bookmarks
creating custom 227
DisplayMode option 210
Box option 180
bridges 2
using multiple 130
built-in functions
REPLYTO 138

C

callback function 220
Certificate Authority 124
CGI
query strings 19
character widths
SplitPercent option 219
CheckNextRecip INI option 221
Class option
PDF printers 210
CleanCache method 317
ClearMsgFile option 149
ClearReq method 18
ClearRes method 18
Client Connection Definition Table 124
cmdGetResponseWithParm method 300
cmdSetFormsetField method 300
CmdWithMessage method 296
code pages
support for PDF files 229
CollapsePage option 181, 231
ColorSheet option 231
CommandTimeout property 167
Comp Pack 214
Comp TIFF 214
Compression attachment variable 134
Compression option 212

compression ratios 211
CONFIG attachment variable
  multiple bridges 130
CONFIG.INI file
  referencing attachment variables 106
correlation IDs 119
CreateScriptFile option 232
custom page sizes
  PaperSize option 211

D

DAP.INI file 145
  distributing email 136
  multiple bridges 130
  PDF compression option 212
  referencing attachment variables 106
DCLTW32 program 16, 19
Debug control group 146
Debug option 294
  MailType control group 137
DefaultTimeoutSeconds attribute 199
Device option 231
DFD VARIABLE option 139
dialogs
  customizing 192
DirLinks option 181, 231
DisableRightClick option 305
DisplayMode option 209, 210
distributed documents 2
DOCCLNT.INI files
  request types 131
DOCSERV.INI file
  referencing attachment variables 106
  request types 131
docserv.xml file 145, 330
Documaker
  running via IDS 144
Documaker Bridge
  error messages 346
Documerge 2
DownloadDPWFonts option 294
DownloadFAP option 221
DownloadFonts option
  embedding fonts 215, 224
  overview 231
  PDF printers 209
DP.DLL 311
DPP files 283
DPRAdd2Attachment rule
  distributing email 137, 139, 140
DPRCreateEMailAttachment rule
  distributing email 138, 140
DPRDecryptValue rule 196
DPRFap2Html rule 235
DPRFindTemplate rule
  distributing email 136, 138, 139, 140
DPRLog rule
  distributing mail 141
DPRMail rule
  and the DPRLog rule 141
  distributing email 137, 139, 140
DPRParseRecord rule
  and the DPRLog rule 141
  distributing email 136, 139
DPRPrint rule 151
DPRSetConfig rule
  multiple bridges 130
DPRTblLookUp rule 235
DPRWIP rules
  and Print Preview 237
DPRWipTableParms rule 248
DPW files 152
DRLGetConfig 202
DSICoEx
  sample output 333
DSICoTB
  sample output 334, 335, 336, 337
DSIEncr COM object 195
DSIEXW32 program
  sample output 333
DSIGetSOAPMessage 164

DSIGetSOAPMessageSize 164
DSIJWP.DLL file 134
DSILIB
    default time-outs 199
DSIMessage class 76
DSIRowset2XML rule 194
DSIRowset2XMLSize rule 194
DSIServer control group 151
DSITEST utility 103
DSN property 167
DumpScript option 232
duplex 134

E

email
    distributing with IDS 135
    message bus 141
Email2IDS control group 136, 139
EmailAdd option 136
EmailAdd2Attachment control group 137, 140
EmailDFD control group 136, 139
embedding fonts
    compressing 211
    how to 224
    PDF Print Driver 215
encrypting
    URLs 195
Enterprise Web Processing Services (EWPS) 205
EntryBackColor option 232
EntryFontColor option 232
EOF property 168
eplyToQueueManagerName property 125
error messages 273, 339
    AFP 352
    displaying 340
    Documaker Bridge 346
    Internet Document Server 344
    Java 350
    Printstream Bridge 351
Errors property 168
Execute method 168
executive page size
    PaperSize option 211
extra info 227

F

FAP files
    converting into HTML 179
    converting to XML 197
FAP2HTML utility 179
FAP2XML utility 197
FAPGetExtraInfo function 228
FAPPutExtraInfo function 228
favorites list 274
FD2HTW32 utility 177
Field option 180
FieldErrors option 149
FieldFontFudge option 181
Fields property 168
File option
    EmailDFD control group 136
FILE2IDS utility 132
FileExists method 318
FileExt option 151
files
    sample output 333
    system 329
    testing the transfer of 103
firewalls
    using 195
font cross-reference files
    and the PDF Print Driver 223
    optimizing 213
Font File field 225
Font File Name field 217
font IDs
    PDF Print Driver 223
    removing 213

Font Index field 217, 225
Font Manager
  embedding fonts 224
font mapping
  PDF Print Driver 229
Font Name field 216
FontCompression option 211
fonts
  AFP error messages 352
  changing 217
  compressing embedded fonts 211
  embedding PostScript fonts 224
  FieldFontFudge option 181
  XML 183
ForceColorBitmaps option 211
FORM.DAT file
  publishing forms 177
forms
  publishing on the web 177
frequently used forms 274
From option 137, 138
FSISYS.INI file 149
  and the PDF Print Driver 209
  callback function 220
FSIUSER.INI file 149
FTP
  and firewalls 195
full-screen mode
  DisplayMode option 210
FXRVALID utility
  embedding fonts 215, 217
  optimizing PDF files 214

G

GenData
  creating HTML files 233
GenDataStopOn control group 149
GenPrint
  and the PDF Print Driver 209
  callback support 220
  CheckNextRecip option 221
  MultiFilePrint option 221
  SendOverlays option 221
GENSemaphoreName option 146, 150
GetAttach 106
GetMsg method 318
GetUniqueString method 318
GetVersion method 299, 301

H

HiddenFieldScript option 232
hInstance property 17
HR option 181, 231
HTML
  and JSP 13
  print driver 230
  supported language features 8
  vs. PDF 4
HTML files
  converting FAP files 179

I

i_GetMRLResource rule 274
iDocumaker
  favorites list 274
IDS
  pausing 172
  running Documaker 144
IDSASP
  creating front-end solutions 16
  illustrated 17
IDSINSTANCE variable 70
IDSJSP bean 14
IDSJSP.jar 13

IDSServer control group 145, 146, 151
IDSSQL.ADO 167
IDSSQL.DLL 166
IDSSQL.IDSRC 168
IDSSQLRULE.DLL 166
IECOLOR.CSS file 231
Image Only PDF files 229
ImageErrors option 149
ImageExt option 181
ImagePath option 181, 231
ImagePathCreate option 231
IMG_ZIndex option 233
INI files
  customizing Print Preview 248
  FSISYS.INI and the PDF Print Driver 209
  list of 330
INIFile option
  multiple bridges 130
Initialize method 318
InitializeDefaults method 319
INIToken option 305
instance numbers 70
instances
  Watchdog 66
internal message format 192
Internet access 6
Internet Document Server
  and Document Management Solutions 2
  error messages 344
  illustration 12
  overview 11
  using the 9
intranet 2
IRLFileFTP rule 195
IRLInitFTP rule 195

J

J2EE-compliant application servers 113
Java
  error messages 350
  Java Management Extensions (JMX) 94
  Java Message Service (JMS) 113
  Java Naming and Directory Interface (JNDI) 113
  server pages 13
  threads test utility 57
  WebSphere MQ 116
JavaScript option 231
JPEG files 214
JSON 205

L

LDAP 198
legal page size
  PaperSize option 211
letter page size
  PaperSize option 211
limitations
  PDF Print Driver 229
load balancing 74
LoadFAPBitmap option 221
log files
  DPRLog rule 141
  logging categories 96
LogConfConvert.xsl template 95
LogFile option 149
LogFileType option 149, 150
LOGIN.ASP 245
LOGIN.ASP file 25
logos
  error messages 353
  optimizing PDF files 213

M

Mail control group 137
MailFunc option 137

MailType control group 137
MailType option 137
Management Information Base (MIB) file 64
marshaller class 76
MaxTimeoutSeconds attribute 199
message format
    internal 192
MinTimeoutSeconds attribute 199
Module option 137, 231
    PDF printers 210
MoveFirst method 168
MoveLast method 168
MoveNext method 168
MovePrevious method 168
MQSeries 116
    DP.DLL COM object 311
MultiFileLog option 221
MultiFilePrint callback function 220, 221
MultiFilePrint option 149, 150
multi-line text fields
    in XML files 183
MultiPage option 231
multiple bridges 130
multiple servers
    measurements 52
    using 50
MVS
    ODBC connections 51

N

Name option 137
NoBatchSupport option 222

O

ODBC
    connections to MVS 51
oDSI property 17
OnEndPage method 18
OnStartPage method 18
Option field 217
Options field 217, 225
OTH record 217
Other tab 217
Other tab (Font Properties window) 225
OutputFunc option 134
OutputMod option 134
overlays
    and the PDF Print Driver 221
    error messages 352
OverridePrompt option 305

P

page segments
    error messages 353
PageBreaks option 181, 231
PageNumbers option 210, 231
PageSize property 167
PaperSize option 211
Password property 168
passwords
    and firewalls 195
Path option 139
pausing IDS 172
PCL printing
    mixing simplex and duplex 134
PDF Converter 3
PDF files
    creating separate files 222
    embedded fonts 223, 226
    initial display 210
    optimizing 213
    showing 22
    types of 229
    vs. HTML files 4

PDF Print Driver
changing fonts 217
fonts 229
limitations 229
overview 208
setting up 209
support for the GenPrint program 209
PDFW32.DLL 209
performance
measurements with multiple servers 51
PDF Print Driver 221
using multiple servers 50
personal forms lists 274
point sizes 223
Port option 137
Portable Document Format 208
PostGenDataExecutable option 155
PostGenPrintExecutable option 155
PostGenTrnExecutable option 155
PostScript
embedded fonts 223
PostScript Font File Name field 217
PostScript fonts
compressing 211
embedding 215, 217
PreLoadRequired option 222
Print Preview
ASP files 245
compressed PCL files 134
customizing 247
illustration 238
overview 237
using 239
PrintFormset rule 220
PrintFunc option 210, 231
PRINTOUT.ASP 25
PRINTPATH attachment variable 151
PrintPath option 151
Printstream Bridge
error messages 351
PrintViewOnly option
PDF printers 210
processing
documents using the internet 1
ProcessQ method 18, 19
ProcessRq method 18, 20
ProcessTrn method 319, 325
PRTLIB
and the PDF Print Driver 220
PrtType control group
PDF compression option 212
PDF Print Driver 209, 224
PrtView_WIPTable control group 248
PRTZCompressOutPutFunc function 134
publishing forms on the web 177
PullCode option 136
PutMsg method 319

Q

queues
~GetAttach variable 62, 106
client connection definition tables 124
default message queue handler 109
DESTINATION parameter 61
HTTP queues 110
IDSClientRule 60
logging categories 84
message queues 108
messaging systems 75
pausing IDS 172
pooling 122
ReplyToQueueName property 125
security exits 123
SOAP 158
SSL connections 124
transforming XML messages 188
using HTTP 126
using Java message service 113
using multiple 111
using WebSphere MQ 116
WaitTime property 18

R

ReadBinFile method 18, 24
ReadIniOptions method 320
ReceiveByCorrelationID API 119
RECIPS.ASP 25
RECORD.ASP 25
RecordCount property 168
RepeatInterval attribute 201
ReplyTo option 137, 138
replyToQueueName property 125
REQTYPE
    multiple servers 51
request
    submitting batch requests 132
    types 131
Request property 17
requests
    monitoring 65
    timed 201
RequestValue method 320
required
    components 6
Response.Redirect method 22
Result property 17
ResultValue method 321
Rotated Fonts field 213
RPDCheckRPRun rule 146
RPDCreateJob rule 146
RPDJobTicket variable 146
RPDProcessJob rule 146
RPDRunProcess variable 146
RPDRunRP control group 146
RPDSemaphoreName option 146, 150
RPDStopRPRun rule 146
RPEX1.INI file 145
RULServerJobProc option 150
RULStandardBaseProc rule 151
RUNMQSC tool 116
RunOnPrimaryInstanceOnly attribute 201

S

samples
    output files 333
SaveOnExit option 305
ScriptPath option 232
ScriptPathCreate option 232
SEARCH.ASP file 25
security
    URLs 195
security issues 195
semaphores 146
SendColor option 231
    PDF printers 210
SendOverlays option
    PDF printers 210
Server option 137
Server.CreateObject method 196
ServerBaseProc rule 151
servers
    performance measurements 52
    setting up additional 53
    using multiple 50
SetGUID method 321
setting up
    a Windows NT Service 54
    PDF compression options 212
    the PDF Print Driver 209
Setup Data field 216
ShowAtt property 17
simplex 134
single step mode 220
SleepingTime option 150
SNMP server programs 64
SOAP 205
    DP.DLL COM object 311
    message format 158
SOAP standards 192
SOAPAddAttachment method 321
SOAPGetAttachment method 322
SOAPGetAttachmentAsBuffer method 322

SOAPLoadAttachment method 322
SOAPUnloadAttachment method 323
SplitPercent option 210, 214, 219
SplitText option 180, 210, 214, 219, 232
SQL
    connecting to 166
SQLCommand property 168
SSL connections 124
symbol fonts 225
system files 329

T

Table option 181
TemplateFields option 231
Terminate method 323
TerSub paragraphs 235
Text option 180
TextMerge option 180
Thin Client Forms Publisher 171
thin clients 2
thumbnails
    DisplayMode option 210
TIFF files
    converting to PDF 229
timed requests 201
TimeOut property 18
time-outs
    DSILIB client applications 199
TPDInitRule rule
    multiple bridges 131
Trace method 323
TransactionErrors option 149
TrapEvents option 305
TrapOnlyQuitEvent option 305
TrueType fonts
    compressing 211
    embedding 215, 217
Type 1 fonts 225
TypeFace field 216

U

UDDI compliance 158
Unicode2Bin method 323
URL
    requests 127
URLs
    encrypting 195
user IDs
    and firewalls 195
User property 168
USERINFO database 250
USERLIST attachment variable 250
using
    multiple bridges 130
    the Internet Document Server 9
    the PDF Print Driver 208

W

WaitForStart option 149
WaitTime property 18
Watchdog 70
Watchdog process 66
Watchdog timer value 345
web servers
    and firewalls 195
WebLogic 113
WebSphere 113
    CCDT files 124
    correlation IDs 119
    overview 116
    security exits 123
    setting up 117
    SSL connections 124
Windows NT
    setting up an NT Service 54
WindowsRawPrinter.jar file 134

WIP Edit plug-in
changing user assignments 303
cmdGetResponseWithParm method 298, 300
cmdSetFormsetField method 297, 300
DPW files 152
GetVersion method 299, 301
WIPAPLST.ASP 245
WIPAPPR.ASP 245
WIPCTL program 296
WIPENTRY.ASP 246
WIPFIND.ASP 246
WIPFMSET.ASP 246
WIPFOOT.ASP 247
WIPHEAD.ASP 247
WIPLIST.ASP 245
WIPRECIP.ASP 246
WIPSTAT.ASP 246
WriteBinFile method 324
WriteToLog method 324
WSCOLOR.CSS file 231

X

XML
error message template 340
formatting text 183
internal message format 192
message format 158
XML files
converting FAP files 197
XML2Attach control group 137, 139, 140
XML2Body control group 136, 139, 140
XRFToken option 294

Z

Security exits 123