Oracle® Documaker

Internet Document Server SDK Reference

version 2.2

Part number: E16256-01

March 2011
Copyright © 2009, 2010, Oracle and/or its affiliates. All rights reserved.
The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.
The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

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

Oracle, JD Edwards, and PeopleSoft are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.
THIRD PARTY SOFTWARE NOTICES

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

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

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

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

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

Copyright © 2009, Berkeley Software Distribution (BSD)

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

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE JDOM AUTHORS OR THE PROJECT CONTRIBUTORS BE LIABLE FOR ANY 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) 2000-2004 Jason Hunter & Brett McLaughlin. All rights reserved.

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

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

Copyright © 2009 MIT

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

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

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

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

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

This product includes software developed by W3C.

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

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

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

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

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

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

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

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

Chapter 1, Using the Internet Document Server SDK

4  Queues
5  Finding the Information You Need
7  Using the DSI APIs with C
10 Using Unicode in Attachment Variables
   10  Sample Program-DSIEX
12 Writing Processing Rules in C
   12  How the System Processes Rules
   13  Creating Rules
   15  Creating, Accessing, and Destroying Variables
   15  Accessing the Attachment
16 Accessing the Queue
16 Using Utility Functions
16 Creating Rules for Reserved Request Types
19 Using the Java Libraries
   20  Using the MsgClient Sample Program
21 Writing Processing Rules in Java
   21  How the System Processes Rules
   21  Developing and Deploying Java Rules
23 Java Rules vs. C Rules
23 Function Signature for Java Rules
25 Using the IDSWebdav Servlet Client APIs and DPRLIB Rules
   26 DPRLbyPropFind
   28 DPRLbyGet
   30 DPRLbyPut
   31 DPRLbyLock
   32 DPRLbyUnlock
   33 DPRLbyDelete
   34 DPRLbyOptions
   35 DPRLbyCopy
Chapter 2, DSI C APIs

C API Functions

83 DSIAddAttachRec
84 DSIAddAttachVar
85 DSIAddAttachVarEx
86 DSIAddToAttachRec
87  DSIAddToAttachRecEx
89  DSIAddToQueue
90  DSIAttachCursorFirst
92  DSIAttachCursorFirstEx
94  DSIAttachCursorLast
96  DSIAttachCursorLastEx
98  DSIAttachCursorName
100  DSIAttachCursorNext
102  DSIAttachCursorNextEx
104  DSIAttachCursorPrev
106  DSIAttachCursorPrevEx
108  DSIAttachCursorValue
110  DSIAttachCursorValueEx
112  DSIAttachVarLength
113  DSIAttachVarLengthEx
115  DSICacheFile
116  DSICloseAttachCursor
117  DSICopyAttachVars
118  DSICopyQRecord
119  DSICreateValue
120  DSIDeleteAttachVar
121  DSIDestroyValue
122  DSIEncryptValue
123  DSIEncryptValueEx
124  DSIErrorMessage
125  DSIErrorMsg
126  DSIFindInQueue
127  DSIGetFirstFromQueue
128  DSIGetSOAPMessage
129  DSIGetSOAPMessageSize
130  DSIGetQError
131  DSIGetQField
133  DSIGetQFieldLength
134  DSIGetQueueRec
135  DSIGetUniqueString
136  DSIInit
Chapter 3, DSI Java APIs

172 Using JavaBean Components
176 Returning a RecordSet Object
178 Using IDSJSP in a JSP Container
178 DSI Bean APIs
179 Using the DSI Java Messaging Library for Client Applications
180 Passing JVM Options to DSILIB
Chapter 4, DSI Processing Rules

184 Server Rules
186 FTPRule
187 Putting and Getting Multiple Files
189 IRLCleanDirectory
191 IRLClearLog
192 IRLCopyAttachment
193 IRLDecryptValue
194 IRLInit
195 IRLFileFTP
202 IRLInitFTP
203 IRLLog
204 IRLPurgeCache
205 IRLSearch
206 IRLSendVersion
208 IRLStatistics

225 Client Rules
226 ATCAppend2Attachment
227 ATCLoadAttachment
228 ATCLogTransaction
230 ATCReceiveFile
233 ATCSendFile
235 ATCSendMultipleFiles
236 ATCUnloadAttachment
238 IRCInit
239 IRCPrint
240 IRCRequest
241 IRCResult
242 IRCSendVersion
244 IRCUnloadPage
Chapter 5, DSI Visual Basic APIs

246   Using the Prototypes and Examples
246   Handling Errors
247   Using the Web Services Example
248   Visual Basic Methods
       AddAttachRec 251
       AddAttachVar 253
       AddToAttachRec 254
       AddToQueue 256
       AttachCursorFirst 257
       AttachCursorLast 258
       AttachCursorName 260
       AttachCursorNext 261
       AttachCursorPrev 262
       AttachCursorValue 264
       AttachList 265
       CacheFile 266
       CloseAttachCursor 267
       CopyAttachVars 268
       CopyQRecord 269
       CreateValue 270
       CreateValueObj 272
       DeleteAttachVar 274
       DestroyValue 275
       DestroyValueObj 277
       DumpDebugInfo 279
       ErrorMessage 280
       FindInQueue 281
       GetAttachment 282
       GetAttachmentAll 284
       GetAttachRecSet 286
       GetAttachVarSet 288
       GetPriority 290
       GetQField 291
Property Instance 334
Property Signature 335
Property TraceEnable 336
Property TracePath 337
Chapter 1
Using the Internet Document Server SDK

This chapter tells you how to use the Document Server Interface (DSI) APIs for creating rules and applications to interface with Oracle Insurance’s Internet Document Server. The various API functions and processing rules are described in detail in this manual.

You can use the API C functions, Java methods, Visual Basic methods, and processing rules to build either a proprietary client interface or a custom set of rules which work with the Internet Document Server.

The APIs provide a number of services, including...

- Interprocess communication
- Persistent variables
- Accessible across function calls
- Error reporting

Several general purpose functions are also available.

The DSI API includes interfaces (APIs), for C, Java, and Visual Basic so you can use these languages to build custom rules and applications. You will also find sample clients which you can use as a reference. For more information, see...

- Finding the Information You Need on page 5
- Using the DSI APIs with C on page 7
- Writing Processing Rules in C on page 12
- Using the Java Libraries on page 19
- Using the IDSWebdav Servlet Client APIs and DPRLIB Rules on page 25
- Writing Processing Rules in Visual Basic on page 61

The illustration on the following page shows how data flows within the system and its overall architecture.
Chapter 1
Using the Internet Document Server SDK

**Distributed Clients**

- World Wide Web or Intranet Client
- Local or Batch Client

**Front-End (Client) Components**

- JSP or servlet based Web Application
- Java-based Custom Client Module
- ActiveX-based Custom Client Module

**Internet Document Server**

**DSI API**

- Request Queue (input)
- Request Queue (output)

**Document Processing Server**

**Back-End Components**

- Bridges
- Processing Rules
- Data, Document Sets or Archives

Front-end components talk to IDS via the DSI API. These components provide communications and an interface which gather client request, translate those requests for the Internet Document Server, and then translate the results for the client's use.

Back-end components include bridges to other applications, the rules which process the data, the data or archives being processed, and document sets. These components communicate with IDS via the DSI API.
Two-tier and three-tier models are supported. In the three-tier model, the remote client can take a variety of forms and paths. The remote can be a web browser using CGI, a web browser using Java, or stand-alone, \textit{fat or thin}, Java or C clients. Notice that there are two paths from the remote client, one through a front-end component, such as CGI, JSP or servlet, and the other through a Java client. The two paths merge at the DSI API, one for C the other for Java.

The system includes a CGI client, which supports rules. Because you can write your own front-end client, the term \textit{front-end client} applies to both. Discussions about rule processing in the front-end client, however, refer to a CGI client.

Similarly, the two-tier model can be supported by writing local applications, such as those that do not use remote communications. You can write these local applications in either Java or C. These local applications use their own APIs. The DSIEX.C sample program, discussed in the topic \textit{Sample Program-DSIEX on page 10}, is an example of a local application.

Aside from the languages there are these key differences:

- The front-end CGI client supports rules and relies on HTML scripts
- The Java browser applet has a persistent connection with the Java server console application.
- The CGI script runs on a front-end client on the HTML server; the Java applet processing is split between the remote web browser and the server.

The general structure of a DSI session depends upon whether you are writing an executable program or a custom set of rules in C or an applet and application in Java. An executable program requires additional calls to initialize and terminate the Internet Document Server and its database access subsystems. To keep things from getting too confusing, the markers below indicate the steps unique to \textit{CGI} or \textit{Java}:

<table>
<thead>
<tr>
<th></th>
<th>Java</th>
<th>CGI</th>
<th>Java</th>
<th>CGI</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The browser makes a request to a web application (JSP or servlet)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>The browser loads an HTML page with a reference to a CGI script</td>
<td>The web application accepts user input, creates a request and adds the request to the server’s request queue</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>The client executable on the server (CGI or Java) receives user input.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Based upon data supplied by the user, the rules create an attachment and a queue record</td>
<td>The data compiled by the rules is added to the server’s request queue.</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>The server retrieves the request from its queue, and, based upon the request, executes its own set of rules</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>The rules read the attachment record and use the supplied information to create a new attachment and queue record</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>The data compiled by the server rules is posted to the server’s result queue</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td>The client retrieves the results and executes yet another set of rules</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGI 11</td>
<td>The rules read the attachment created by the server and use this information to format output to be provided to the user</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Java 12</td>
<td>The information is passed to the web application, which formats a reply and passes the reply to the browser</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGI 13</td>
<td>An HTML page is formatted and passed to the browser</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** An attachment is a block of information accessed in the form of name/value pairs. Attachments are used to pass information between the client and the server rules, as well as the API.

This sequence is greatly simplified, ignoring the details of how rules compile data and determine what information needs to be provided at each stage of the process. These details may include database accesses, requests from the user for additional information, the creation of files, and other tasks.

**Queues**

Typically, you will have more than one browser active at a time so input and output to the Internet Document Server is organized around queues. These queues serialize the requests and process them on a first in, first out basis. The DSI queues also let you prevent conflicts as several clients perform several tasks at a time.
Depending on how you implement the system, you may not need to install or use all of the components. Below is a table which shows the order in which you should read the chapters and appendices in this manual and in the other Internet Document Server related guides and briefly describes these chapters or appendixes.

<table>
<thead>
<tr>
<th>To...</th>
<th>Read...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find an overview of the Internet Document Server</td>
<td>Chapter 1 of the Internet Document Server Guide.</td>
</tr>
<tr>
<td>Create PDF, HTML, or XML output</td>
<td>Chapter 3 of the Internet Document Server Guide.</td>
</tr>
</tbody>
</table>

Once you install the Internet Document Server, you will typically use one of the following bridges:

<table>
<thead>
<tr>
<th>Bridge</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documaker Bridge</td>
<td>This bridge lets you retrieve and display form sets stored in Documaker’s archive module. For more information, see Using the Documaker Bridge.</td>
</tr>
<tr>
<td>Documanage Bridge</td>
<td>This bridge lets you web-enable Documanage to search, retrieve, and manipulate information from Documanage. See the Documanage manuals entitled, General Reference for the Documanage Bridge and Rules Reference for the Documanage Bridge for more information.</td>
</tr>
<tr>
<td>Printstream Bridge</td>
<td>This bridge lets you convert Metacode and AFP output created by the Documerge system into PDF files used by the Internet Document Server. For more information, see Using the Printstream Bridge.</td>
</tr>
<tr>
<td>Docuflex Bridge</td>
<td>This bridge lets the Internet Document Server use Docuflex as a composition engine. For more information, see Using the Docuflex Bridge.</td>
</tr>
</tbody>
</table>

If you plan to customize the Internet Document Server, either by building custom client modules or by adding processing rules, install the Internet Server SDK and refer to the appropriate chapters of this manual for additional information.

<table>
<thead>
<tr>
<th>To...</th>
<th>Read...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Install and learn about the Internet Document Server SDK</td>
<td>Chapter 1, Using the Internet Document Server SDK, beginning on page 1</td>
</tr>
<tr>
<td>Use C to customize the Internet Document Server</td>
<td>Chapter 2, DSI C APIs on page 79.</td>
</tr>
<tr>
<td>Use Java to customize the Internet Document Server</td>
<td>Chapter 3, DSI Java APIs on page 171</td>
</tr>
<tr>
<td>Have the Internet Document Server run specific processing rules</td>
<td>Chapter 4, DSI Processing Rules on page 183</td>
</tr>
</tbody>
</table>
Keep in mind that XML standards, as defined by the W3C, require you to substitute text characters that are not in XML tags (for example, between `<entry>` and `</entry>` tags) as escape sequences. The characters that require substitution are listed in the following table. If you cut and paste an XML example from this or other Docupresentation documentation into an XML configuration file, you will have to manually make these substitutions.

<table>
<thead>
<tr>
<th>For this character</th>
<th>Use this escape sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;</code> (less than)</td>
<td><code>&lt;</code> (less than)</td>
</tr>
<tr>
<td><code>&gt;</code> (greater than)</td>
<td><code>&gt;</code> (greater than)</td>
</tr>
<tr>
<td><code>&amp;</code> (ampersand)</td>
<td><code>&amp;</code> (ampersand)</td>
</tr>
<tr>
<td><code>'</code> (apostrophe)</td>
<td><code>'</code> (apostrophe)</td>
</tr>
<tr>
<td><code>&quot;</code> (quotation mark)</td>
<td><code>&quot;</code> (quotation mark)</td>
</tr>
</tbody>
</table>
A front-end client has a number of convenient and powerful features for access to the Internet Document Server using the DSI C API. Note that access to all of the client functionality is not provided through the DSI C API.

You must handle memory management, rule processing, HTML formatting, and other calls to the operating system. The DSI API does, however, handle communication with the server. You can find prototypes for all of the DSI C API functions in DSILIB.H. For executable programs, access to the DSIW32.DLL file must be explicitly included in your link by including the implib DSIW32.lib.

In addition, a number of functions are available expressly for use in custom front-end clients. If you are writing an executable program, note that the client must call the DSIInit and DSIInitInstance functions before it calls any of the other DSI functions.

**NOTE:** You cannot call the DSIInit and DSIInitInstance functions more than once without an intervening call to the DSITerm and DSITermInstance functions.

The DSIInit function returns a process-level handle used for calls to the DSInitInstance function, which in turn returns a thread-level handle. The instance handle is used for all subsequent calls to DSI functions.

```c
/* for .EXE only */
hApp = DSIInit();
hInstance = DSIInitInstance( hApp );
```

If you are writing rules and not an executable program, the opposite is true. You should not call the DSIInit and/or DSIInitInstance functions because the program running the rules has already made those calls. As you'll see in the topic Writing Processing Rules in C on page 12, you will be passed the instance handle every time the rule is called.

**NOTE:** The functions DSIInit, DSIInitInstance, DSITermInstance, and DSITerm functions are required for EXEs only. Do not use them when writing rules.

If you are using the queue APIs, the next task is to call DSIInitQueue once for each of the input and output queues. These calls initialize the communication channels between a front-end client and server and create the attachment lists.

```c
DSIInitQueue( hInstance, DSI_INPUTQUEUE, "RESULTQ" )
```

Once the queues have been initialized, you can implement your design. The queue fields required by the server are:

- the request type (see the table on page 16.) DSIQSET_REQTYPE
- your user ID (your choice) DSIQSET_USERID
- a globally unique identifier, DSIQSET_UNIQUE_ID

Once the rule processing has been completed and the attachment list filled, a front-end client must fill the appropriate queue fields and add the record to the queue for retrieval by the server. Additionally, if a front-end client provides attachment data to the Internet Document Server, you must set the DSIQSET_ATTACHMENT field.
**NOTE:** You set the DSIQSET_ATTACHMENT field to add a single attachment buffer that the caller maintains. For other situations, you would use the DSIAddAttachVar and DSIStoreAttachment functions.

Since your process or thread likely will not be the only user of the server, the DSIQSET_UNIQUE_ID field, which you will use to locate the response, should be unique to a given request. The easiest way to do this is to use the DSIGetUniqueString function, as shown here:

```c
/* set the request type */
DSISetQField(   hInstance,
               DSI_OUTPUTQUEUE,
               DSIQSET_REQTYPE,
               "SSS",
               sizeof( "SSS" ) );

/* set the user id */
DSISetQField(   hInstance,
               DSI_OUTPUTQUEUE,
               DSIQSET_USERID,
               "MyID",
               sizeof( "MyID" ) );

/* set the unique id */
first the field length */
DSIGetQFieldLength(    hInstance,
                      DSI_OUTPUTQUEUE,
                      DSIQSET_UNIQUE_ID )

/* next get a unique identifier from DSI */
DSIGetUniqueString( hInstance, szUnique, cbUnique );

/* put unique id into the queue record */
DSISetQField(   hInstance,
               DSI_OUTPUTQUEUE,
               DSIQSET_UNIQUE_ID,
               szUnique,
               cbUnique );
```

Once the above fields have been filled, call the DSIAddToQueue function to post the message to the server.

```c
DSIAddToQueue( hInstance, DSI_OUTPUTQUEUE );
```

To use a proprietary attachment format, retrieve each attachment variable in turn, copying them all into a single buffer in the format desired, and pass the result to the DSISetQField function. The length of this buffer cannot exceed 64K.

To retrieve results from the Internet Document Server, call the DSIFindInQueue or DSIGetQueueRec function with the pszId parameter set to the value used for the DSIQSET_UNIQUE_ID (we recommend that you use the DSIGetUniqueString function to generate this value).
You can then retrieve the attachment from the result record using the DSIGetQField function and parse it into individual attachment variables. Alternatively, you can use the DSIParseAttachment function to produce a list of name/value pairs that can be retrieved using the DSIAAttachCursorFirst, DSIAAttachCursorNext, DSIAAttachCursorPrev, DSIAAttachCursorLast functions, as shown below:

```c
DSIGetQueueRec(hInstance,
              DSI_INPUTQUEUE,
              szUnique,
              1000L,
              10000L);
DSIParseAttachment(hInstance, DSI_INPUTQUEUE);
DSIOpenAttachCursor(hInstance, DSI_INPUTQUEUE);
DSIAAttachCursorFirst(hCursor,
                        szName,
                        sizeof(szName),
                        szValue,
                        sizeof(szValue));
DSIAAttachCursorNext(hCursor,
                      szName,
                      sizeof(szName),
                      szValue,
                      sizeof(szValue));
DSICloseAttachCursor(hCursor);

/* for .EXE only*/
if (hInstance != DSINULLHANDLE) {
    DSITermQueue(hInstance, DSI_INPUTQUEUE);
    DSITermQueue(hInstance, DSI_OUTPUTQUEUE);
    DSITermInstance(hInstance);
}
if (hApp != DSINULLHANDLE) {
    DSITerm(hApp);
}
```
IDS now supports Unicode, via UTF-8 encoding, in the setting and retrieving of values from attachment variables. The support is implemented via new functions and defined constants in the DSILIB library. The new functions are:

- DSIAddAttachVarEx
- DSIAddAttachRecEx
- 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 a rule writer can either use

- DSIAddAttachVar(hdsi, DSI_OUTPUTQUEUE, "FIELD", szValue);
- DSIAddAttachVarEx(hdsi, DSI_OUTPUTQUEUE, "FIELD", szValue, DSIENCODING_SINGLE_BYTE);
- 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 code page 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.

**Sample Program - DSIEX**

As an aid, the Internet Document Server includes a sample program named DSIEX.C and its executable DSIEXW32.EXE. It is a simple, single-threaded console application, which opens an input and output queue, requests the server status, and dumps the results to sysout. It also checks the installation and setup.

To run DSIEXW32.EXE, follow these steps:

1. Start the Internet Document Server in the \DOCSERV directory.
2 Run DSIEXW32.EXE.

The DSIEX program will run for a few seconds and stop after producing 30+ lines of output. If you want to look more closely at the output, which includes a listing of all the libraries used by Internet Document Server, redirect the output to a file.

Take a look at DSIEX.C and you will see it includes all the steps outlined above, especially those required for an executable program, such as the calls to the DSInit, DSInitInstance, DSITermInstance, and DSITerm functions.
A rule is an entry point in a DLL that follows a standard parameter set or convention. You can use rules to customize how your system operates. The processing rules run either in a front-end client, such as the CGI client, or in the Internet Document Server. Please refer to Chapter 3 in the Internet Document Server Guide, for a discussion on configuring the rules in the configuration file. The standard rules you can use are explained in the topic Server Rules on page 184.

The rules run by the front-end CGI client are contained in DLLs, which the system loads when it receives a request that requires the use of a rule. Because rules run within the process address space of the executable program, memory violations within a rule are memory violations within the server. This is not a result you want to occur so take steps to prevent them.

The same may be said of memory leaks and performance bottlenecks. For this reason, you should carefully write and test the rule before you place it in service. There are some good tools available to help you look for bugs, memory leaks and performance bottlenecks, such as Bounds Checker and Heap Agent. The results are well worth the effort. It is assumed that you are familiar with the C programming language.

**HOW THE SYSTEM PROCESSES RULES**

To process the various rules, the system loops through a list of rules and calls each in turn with this set of messages:

- DSI_MSGINIT
- DSI_MSGRUNF
- DSI_MSGRUNR
- DSI_MSGTERM

**DSI_MSGINIT message**

The DSI_MSGINIT message lets a rule initialize lists and other data structures that will be used during processing of the following messages or by other rules.

**NOTE:** This rule list is run in forward order.

**DSI_MSGRUNF and DSI_MSGRUNR messages**

The DSI_MSGRUNF and DSI_MSGRUNR are the actual processing messages. Two processing messages are provided so rules have a chance to provide additional processing after other rules have done their work. The rule list is run in forward order during the processing of the DSI_MSGRUNF message and in reverse order while processing the DSI_MSGRUNR message.

**DSI_MSGTERM message**

Finally, the DSI_MSGTERM message allows rules to release any resources that were allocated during the previous three stages.

**NOTE:** This rule list is run in reverse order.
The rules processing engine provides no means to abort this processing loop. It is your responsibility to check at each stage to make sure that prior rules completed successfully, that necessary data has been provided, and react accordingly.

Used with a front-end CGI client and Internet Document Server, most transactions involve three runs of the rules processing engine. The first run, by the front-end CGI client, transforms user input into data usable by the server. The second run of the rules processing engine by the Internet Document Server performs the actual work of the transaction. The final run of the rules processing engine is again done by a front-end CGI client and transforms the server's results into user output.

During each run of the engine, a different set of data is available for use by the rules. Entering the first run, a front-end client has read and parsed the request, such as a URL provided by the web browser to the CGI client, as well as the environment variables. In the CGI client, each element of the URL and each environment variable are added to the output attachment list to make them available for use by rules.

To provide a front-end client with access to the attachment, be sure the ATCUnloadAttachment rule is present in the client's rule list. The ATCUnloadAttachment rule performs its processing during the DSI_MSGRUNR message. Keep this in mind when you order the rule list. Make sure all necessary attachment variables are created before the attachment is unloaded.

When the Internet Document Server rules run, certain fields in the Request queue record are accessible. To make sure the attachment variables provided by a front-end client are also accessible, include the ATCLoadAttachment rule in the rule list before any rules that require attachment data.

To provide the result processing loop of the client with access to the attachment variables created by the server, make sure the ATCUnloadAttachment rule is in the server's rule list. The ATCUnloadAttachment rule performs its processing during the DSI_MSGRUNR message. Keep this in mind when ordering the rule list so that all necessary attachment variables are created before the attachment is unloaded.

**NOTE:** See also Chapter 3 of the Internet Document Server Guide for more information.

When a front-end client begins to process results, certain fields of the result queue record are again available. As with the server run, any necessary attachment data must be made available with a call to ATCLoadAttachment in the rule list before attempting to access that data.

**CREATING RULES**

The rules you write in C for the client or server must follow this prototype:

```c
#include <dsistd.h>

_DSIXPORT long _DSIAPI MyRule(   DSIHANDLE hInstance,
    char *pszParms,
    unsigned long ulMsg,
    unsigned long ulOptions);
```

- `hInstance` is created by a call to the DSIInit function
• `pszParms` contains the rule parameters, as specified in the configuration file
• `ulMsg` is the current message, as discussed above
• `ulOptions` is reserved for future use

**NOTE:** Rules written for use with the front-end CGI client or server must not call the DSIInit or DSIInitInstance functions. These calls are handled elsewhere.

Each rule will generally have a switch statement with cases for each of the defined messages. Inside the rule, you can do just about anything you want. Remember, though, that allocated memory must be freed, and that performance bottlenecks in a rule create performance bottlenecks for the server.

**Rule template**
Here’s a template for a rule that will help you get started.

```c
_DSIXPORT long _DSIEXPORT MyRule (DSIHANDLE hInstance, char * pszParms, unsigned long ulMsg, unsigned long ulOptions )
{
    switch ( ulMsg )
    {
        /* ---------------------------
        * Initialization Message
        * Add data initialization here
        */
        case DSI_MSGINIT:
            break;

        /* ---------------------------
        * Run Rule Forward Message
        * Do desired processing
        */
        case DSI_MSGRUNF:
            break;

        /* ---------------------------
        * Run Rule Reverse Message
        * Do desired processing
        */
        case DSI_MSGRUNR:
            break;

        /* ---------------------------
        * Termination Message
        * Clear data, free any memory allocated
        */
        case DSI_MSGTERM:
            break;
    }
    return DSIERR_SUCCESS;
}
```
CREATING, ACCESSING, AND DESTROYING VARIABLES

The DSICreateValue, DSIQueryValueSize, DSILocateValue, and DSIDestroyValue functions provide easy access to persistent variables you can access from any rule.

NOTE: Variable names are case sensitive and must be unique.

Accessing the Attachment

The attachment is attached to the queue record passed between the client and server. Attachment variables are similar to those created by the DSICreateValue function, except attachment variables are passed between processes. If a value does not need to find its way from the client to the server or vice versa, use the DSICreateValue rule to create the variable.

The functions you can use to access the attachment are...

- DSIAddAttachVar
- DSILocateAttachVar
- DSIDeleteAttachVar
- DSIOpenAttachCursor
- DSICloseAttachCursor
- DSIAttachCursorFirst
- DSIAttachCursorNext
- DSIAttachCursorPrev
- DSIAttachCursorLast

In addition to these rules, there are several additional functions and rules you can use to access the attachment:

- The DSIAAAAddAttachRec and DSIAAAAddToAttachRec functions let you create stem variables, similar to the C language struct type. These stems allow for multiple records each with members having the same name.
- The HTML formatting rule, IRCUnloadPage, replaces special tags in an HTML template with the values in these variables. See Chapter 3 of the Internet Document Server Guide for more information.
- The DSICopyAttachVars function lets you copy an entire attachment from one queue to another.
- The DSIDSErrorMessage function lets you send formatted error messages to the user. The DSIDSErrorMessage function uses the stem variable capabilities of attachments and the HTML formatting support of the IRCUnloadPage rule so you can precisely report errors.
ACCESSING THE QUEUE

As a general rule, you should not have to access the queue record, as opposed to the attachment, from within custom rules. There may be times, however, when you want to change the request type or priority, or to use a proprietary attachment format. To query and set queue fields, use the DSIGetQField and DSISetQField functions.

There are several field identifiers you can use with these functions. As queue field lengths can change, call the DSIGetQFieldLength function before you retrieve the field. Be very careful when you modify fields, particularly when you use the provided client and server programs, because these programs rely on certain fields.

It is practically inevitable that a queue error will occur at some point. To get information regarding the nature of the error, use the DSIGetQError function.

There are additional queue APIs that should only be used when creating an executable. These APIs will be discussed shortly.

NOTE: The queue names DSI-INPUTQUEUE and DSI-OUTPUTQUEUE are relative, depending on your perspective. For example, the input queue in a rule is the output queue in a client.

USING UTILITY FUNCTIONS

At times, you may need to create and later delete temporary files. The DSI SDK includes two APIs you can use to perform these tasks:

<table>
<thead>
<tr>
<th>To...</th>
<th>Use this function...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate unique file names and avoid naming conflicts</td>
<td>DSIGetUniqueString</td>
</tr>
<tr>
<td>Remove temporary files after a specified time period</td>
<td>DSICacheFile with the IRLPurgeCache rule</td>
</tr>
</tbody>
</table>

CREATING RULES FOR RESERVED REQUEST TYPES

Several request types are reserved for use within the server and/or client. You cannot use these request types for transactions. While a default set of rules is provided for these reserved request types, in some cases you may want to change these defaults.

Here is a list of the reserved requests and a description of each. These requests may or may not be in use at any given time, and the default processing for these requests is subject to change.

<table>
<thead>
<tr>
<th>Request type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM</td>
<td>Reserved</td>
</tr>
<tr>
<td>CAD</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
To extend the existing rules for one of these request types, construct the rule as discussed. Then insert a call to the rule in the appropriate place in the configuration file (refer to Using the Documaker Bridge for more information).

For instance, to add MyPeriodicCleanupFunction in the MYDLL.DLL library after the IRLPurge rule has completed, modify the ReqType:SAR control group as shown here:

```
< section name="ReqType:SAR" >
    <entry name="function">irlw32->IRLPurge</entry>
    <entry name="function">mydll->MyPeriodicCleanupFunction</entry>
</section>
```

If you are replacing the functionality provided for one of the reserved request types, make sure the replacement rule provides adequate functionality. Then, simply remove (or comment) the existing rules and insert the replacements.

**NOTE:** The system does not check the status of rules. Processing continues even if your rule fails. You must make sure the previous steps of the process were completed without error.
A front-end client has convenient Java libraries available from Oracle Insurance for accessing IDS. The Internet Document Server Java Libraries handle communication with the server, the bundling of data and formatting the data for sending to the server, in addition to useful utility functions.

The libraries are available in the DocuCorpUtil.jar and DocucorpMsg.jar files. These files must be part of the CLASSPATH of the Java client program.

You will also need files for the parsing and writing of XML files, xerces.jar and xalan.jar. If you are running Java version 1.3 these files will need to be included in your CLASSPATH. These files are part of the Java runtime version 1.4 and later.

The Internet Document Server Java Libraries provide support for setting up queues for communicating with IDS. This is done through a queue factory, which creates input and output queues. The queue factory can be created using the getQueueFactory method of the class com.docucorp.messaging.DocucorpMsgUtil. Configuration parameters for the queue factory are passed in using a java.util.Properties object. The queue factory can then create the needed queues.

```java
DSIMessageQueueFactory queueFactory = DocucorpMsgUtil.getQueueFactory(props);
DSIMessageQueue inputQueue = queueFactory.createMessageQueue(DSIMessageQueueFactory.INPUTQUEUE);
DSIMessageQueue outputQueue = queueFactory.createMessageQueue(DSIMessageQueueFactory.OUTPUTQUEUE);
```

Requests sent and results retrieved from the server are held in instances of the com.docucorp.messaging.DSIMessage class. This class has methods for storing name/value pairs called message variables and strings or binary data in attachments. There are also methods for setting the request type and unique ID of the request.

```java
DSIMessage requestDSIMessage = new DSIMessage();
requestDSIMessage.setRequestType("SSS");
requestDSIMessage.setMsgVar("USERID", "USER");
requestDSIMessage.setMsgVar("PASSWORD", "PASS");
```

Before the request can be sent the data in the DSIMessage object must be changed to a format that can be sent through the queues. This process is called marshalling. A Java object that marshals a DSIMessage can be created using the getMarshaller method of the class com.docucorp.messaging.DocucorpMsgUtil. The marshaller will read the information in the DSIMessage and create an object that can be sent through the queues.

```java
DSIMessageMarshaller marshaller = DocucorpMsgUtil.getMarshaller(props);
Object request = marshaller.marshall(requestDSIMessage);
```

Since more than one client application can be communicating with the server through the queues, each message should be sent with a unique identifying string so the client application can get the correct result record back from the result queue. The Java class com.docucorp.util.UniqueStringGenerator can be used to make a unique string.

```java
UniqueStringGenerator usg = new UniqueStringGenerator();
String uniqueID = generateUniqueString();
```

With the marshaled request and unique ID, IDS can send the request to the server.

```java
outputQueue.putMessage(uniqueID, request);
```
The client application now waits for the server to process the request and make a result that will go in the client's input queue. The result is marked with the unique ID string sent with the request.

```java
Object result = inputQueue.getMessage(uniqueID, 1000, 3);
```

The result is in the same format that the marshaller used to send the request. To get the data in a usable format, the system uses the same kind of marshaller to unmarshall the result object into a DSIMessage.

```java
DSIMessage resultDSIMessage = new DSIMessage();
marshaller.unmarshall(result, resultDSIMessage);
```

You can now use DSIMessage methods to retrieve message variables and any attachments that the server may have sent back.

```java
Map messageVariables = resultDSIMessage.getAllMsgVars();
Map attachments = resultDSIMessage.getAllAttachments();
```

### Using the MsgClient Sample Program

As an aid, IDS includes a sample program named MsgClient.java and its compiled form MsgClient.class. It is a single-threaded console Java program that will fill in a DSIMessage from a data file, open an output and input queue, send a request, get the result back and display the result on the screen. For this example, assume...

- IDS is running
- The Docucorp Java Libraries, supporting files, and the MsgClient.class file is in a subdirectory called `lib`
- The client configuration file (dsimsgclient.properties) is in the current directory
- The data file (ssstest.txt) is in the current directory

Then you run MsgClient under Windows using this command:

```bash
java -cp lib;lib\DocucorpMsg.jar MsgClient ssstest.txt
```

The MsgClient sample has all the steps outlined above.
Writing Processing Rules in Java

A rule is a method in a Java class that follows a standard parameter set or convention. The method may be an instance method or a static (class) method. You can use rules to customize how IDS operates.

How the System Processes Rules

Each request sent to IDS corresponds to a list of rules. Each rule in the list is called with a set of messages (from the Java class com.docucorp.ids.data.IDSConstants):

- IDSConstants.MSG_INIT
- IDSConstants.MSG_RUNF
- IDSConstants.MSG_RUNR
- IDSConstants.MSG_TERM

MSG_INIT message

The MSG_INIT message lets a rule initialize any data that will be used by itself or other rules during the processing the other messages.

MSG_INIT is run in forward order, starting with the first rule in the request’s list of rules and proceeding to the last.

MSG_RUNF and MSG_RUNR messages

These messages are intended for the main data processing the rules have to do. Two messages are provided so every rule has a chance to run after the rules have been run once.

MSG_RUNF is run in forward order, starting with the first rule in the request’s list of rules and proceeding to the last. MSG_RUNR is run in reverse order, starting with the last rule in the request’s list of rules and proceeding to the first.

MSG_TERM message

The MSG_TERM message lets the rules release any non-memory related resources allocated during the run of the other messages.

Developing and Deploying Java Rules

Java rules are methods in Java classes. The Java class should include a no-argument constructor (unless you are using a static method) and a method that has the rule function signature, described below.

Java rules are deployed by placing the Java executable code in the rules subdirectory of the main IDS directory; there is no need to modify the CLASSPATH of IDS to run the rule. If the executable code is in a .jar file it can be put directly in the rules directory. If the executable code is separate .class files then it needs to have a directory structure that matches the package structure of the Java class.

For example, if the Java rule is CustomRule and its package is com.sampco, then the CustomRule.class file would need to be in the rules/com/sampco directory under the main IDS directory.

In addition to custom rules, any third party Java libraries needed to run the custom rules should be put in the rules subdirectory, such as database drivers, communications code, and so on. Java rules deployed also have access to Java code that is part of IDS. This code is in the lib subdirectory under the main IDS directory.
Every time IDS is restarted the rules subdirectory is checked for rules code. It is not necessary to shut down IDS and start it again to deploy new or updated Java rules.

To run a Java rule in a request, add a line to the request as follows:

```xml
<entry
  name="function">java;classname;objectname;scope;method;arguments</entry>
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>classname</td>
<td>Name of your Java class, in full package form. For example, if you have class CustomRule in the com.sampco package, the classname would be com.sampco.CustomRule</td>
</tr>
<tr>
<td>objectname</td>
<td>Name used to refer to the object. Required if using global scope. Multiple Java rules in different requests with global scope and the same object name would refer to the same Java object.</td>
</tr>
</tbody>
</table>
| scope | Scope can be one of the following values.  
  global - The object will remain until IDS is restarted.  
  transaction - The object will be created during the MSG_INIT message and will remain until the request has processed all the MSG_INIT, MSG_RUNF, MSG_RUNR and MSG_TERM messages.  
  local - The object is created and destroyed for every message run during the request.  
  static - No object is created; the method is a static method of the class and will be run as such. |
| method | Name of the method in the Java class to run as the rule. |
| arguments | Any additional arguments from the configuration line. |

Java rules were also implemented in IDS version 1.x but the function signature was closer to C rules, including the use of a DSI Handle. Although new Java rules should use the new function signature, mentioned below, version 1.x Java rules will run as-is in IDS 2.x.

To run an IDS 1.x Java rule in a request, add a line to the request as follows:

```xml
<entry
  name="function">dsijrule-
  >JavaRunRule,;classname;objectname;scope;method;arguments</entry>
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>classname</td>
<td>Name of your Java class, in full package form, using JNI formatting. For example, if you have class CustomRule in the com.sampco package, the classname would be com/sampco/CustomRule. This makes for easier conversion of IDS 1.x rule lines to IDS 2.</td>
</tr>
<tr>
<td>objectname</td>
<td>Name used to refer to the object. Required if using global scope. Multiple Java rules in different requests with global scope and the same object name would refer to the same Java object.</td>
</tr>
</tbody>
</table>
JAVA RULES VS. C RULES

C rules are functions with no data associated with them. This means that if a C rule needs data to operate it usually needs to allocate data structures in the DSI_MSGINIT message, use the data in DSI_MSGRUNF and DSI_MSGRUNR, and free it in DSI_MSGTERM.

Since the setup of Java rules can include the creation of Java objects from classes, data can automatically be associated with the Java rule. For example a Java rule run under transaction scope can allocate data structures it needs in the object’s member variables at object construction or during the run of the MSG_INIT message. If the resources allocated by the Java object are only memory resources, the memory will be de-allocated during garbage collection some time after the object goes out of scope. If the rule allocates non-memory resources (files, database connections, etc.) then it should follow the usual convention of allocating resources during MSG_INIT and freeing resources during MSG_TERM.

FUNCTION SIGNATURE FOR JAVA RULES

The methods for Java rules must follow this function signature:

```java
public int ruleMethod(RequestState requestState, String arg, int msg)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestState</td>
<td>the object that holds the current running state of the request at this point of execution. This includes a DSIMessage with the input message variables and attachments, a DSIMessage with the output message variables and attachments being built, configuration information to read, and so on.</td>
</tr>
<tr>
<td>arg</td>
<td>the arguments from the rule line of the configuration file.</td>
</tr>
<tr>
<td>msg</td>
<td>the message that is currently being run, either MSG_INIT, MSG_RUNF, MSG_RUNR or MSG_TERM.</td>
</tr>
</tbody>
</table>

The return code should be either IDSConstants.RET_SUCCESS if the rule ran successfully, or IDSConstants.RET_FAIL if not.
Example

Here is an example of a Java class that can be used as a starting point for rule writing:

```java
import com.docucorp.ids.data.*;

public class SampleRule {

    public SampleRule() {
        /*
         * You may want to do some data setup here.
         */
    }

    public int runRule(RequestState requestState,
                        String arg,
                        int msg) {

        try {
            switch (msg) {
                case IDSConstants.MSG_INIT:
                    /*
                     * Do any non-memory related setup here.
                     */
                    break;
                case IDSConstants.MSG_RUNF:
                    /*
                     * Do main processing here.
                     */
                    break;
                case IDSConstants.MSG_RUNR:
                    /*
                     * Do main processing here.
                     */
                    break;
                case IDSConstants.MSG_TERM:
                    /*
                     * Do any non-memory related cleanup here.
                     */
                    break;
            }
            return IDSConstants.RET_SUCCESS;
        } catch (Exception ex) {
            return IDSConstants.RET_FAIL;
        }
    }

    }
```
The IDSWebdavServlet client APIs and server side rules let you update libraries or file systems using these WebDav client commands:

<table>
<thead>
<tr>
<th>options</th>
<th>ls</th>
<th>cd</th>
</tr>
</thead>
<tbody>
<tr>
<td>propgetall</td>
<td>propfind</td>
<td>propget</td>
</tr>
<tr>
<td>get</td>
<td>put</td>
<td>lock</td>
</tr>
<tr>
<td>unlock</td>
<td>delete</td>
<td>copy</td>
</tr>
<tr>
<td>move</td>
<td>proppatch</td>
<td>mkcol</td>
</tr>
</tbody>
</table>

**Library management rules**

You can use these DPRLIB rules to update libraries maintained by Library Manager using WebDav commands.

- DPRLbyGet on page 28
- DPRLbyPut on page 30
- DPRLbyLock on page 31
- DPRLbyUnlock on page 32
- DPRLbyDelete on page 33
- DPRLbyOptions on page 34
- DPRLbyCopy on page 35
- DPRLbyPropPatch on page 36
- DPRLbyMKCol on page 37

**File system rules**

You can also use the following file system rules:

- propFind on page 42
- get on page 44
- put on page 45
- lock on page 46
- unlock on page 47
- delete on page 48
- options on page 49
- copy on page 49
- move on page 50
- propPatch on page 51
- mkCol on page 52
**DPRLbyPropFind**

Use this rule to return:

- The properties for a file if the resource you specify is a file
- A list of files and their properties if the resource you specify is a collection or file type (FAP, LOG, DDT, DAL, FOR, GRP, BDF)
- A list of collections or file types if the resource you specify is root (/).

This rule supports these WebDav commands by querying Library Manager for the configuration specified:

<table>
<thead>
<tr>
<th>Use this command</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>ls [path]</td>
<td>List the contents of a collection.</td>
</tr>
<tr>
<td>cd [path]</td>
<td>Change directories.</td>
</tr>
<tr>
<td>propget [path] [property]</td>
<td>Get a property.</td>
</tr>
<tr>
<td>propfind [path] [property]</td>
<td>Find a property.</td>
</tr>
<tr>
<td>propgetall [path]</td>
<td>List all properties for a resource.</td>
</tr>
</tbody>
</table>

**Input attachments**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| RESOURCEURI      | A resource URI specifying a user ID, config, file type, and resource. Here are some examples of resource URIs:  
|                  | /userid/config/filetype/resource/    |
|                  | /userid/config/filetype/           |
|                  | /userid/config/                    |
|                  | /userid/                          |
| DEPTH            | Enter a depth of one (1) for collections or file types in Library Manager. Enter a depth of zero (0) for file resources. |
### Output attachments

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| **PROPERTIES** | A rowset of rows that match each of the file resources available for a particular collection/file type. If DEPTH is one (1) and RESOURCEURI specifies a collection or file type in Library Manager, the PROPERTIES rowset returns a row for each resource available in the collection/file type.  
If DEPTH is zero (0) and RESOURCEURI specifies a file resource, the PROPERTIES rowset returns a single row with the properties for the resource you specified.  
Each row in the PROPERTIES rowset contains the following properties for a file resource:  
- **supportedlock** - If locking is allowed, this XML string appears:  
  ```xml
  <lockentry>
    <lockscope>
      <exclusive/>
    </lockscope>
    <locktype>
      <write/>
    </locktype>
  </lockentry>
  ```
- **getcontentlanguage** - currently returns `en_US`.  
- **resourcetype** - blank if the resource is a file, otherwise `collection` if the resource is a file type/directory.  
- **displayname** - the display name of the resource.  
- **HREF** - the resource URL for this resource  
- **getlastmodified** - the date and time indicating when the resource was last modified. This is a long value that contains the number of milliseconds since January 1, 1970.  
- **getcontentlength** - currently zero (0) because there is no support for retrieving the file size of a document stored in Library Manager (reserved for future use).  
If a resource is locked these additional properties are returned:  
- **lockowner** - The user ID that set the lock.  
- **lockscope** - The scope of the lock (exclusive).  
- **locksubject** - The name of the resource locked.  
- **lockdepth** - The depth of the resource locked (0).  
- **locktype** - The type of lock (write).  
- **locktimeout** - The time-out value after which the lock will expire (infinity).  
- **locktoken** - A unique ID that identifies the resource locked.  
This rowset is only present if RESULTS contains SUCCESS. |

| **RESULTS** | SUCCESS or ERROR |

---

Using the IDSWebdav Servlet Client APIs and DPRLIB Rules
INI options

Use these options in the DAP.INI file to see a listing of the configurations that support Library Manager.

```
< LbyConfigs >
  Config = RPEX1
  Config = RPEX2
</LbyConfigs>
```

DPRLbyGet

Use this rule to get or check out a resource file from Library Manager. This rule can retrieve a resource file by version and revision or by name, in which case it retrieves the latest version and revision for the resource specified. This rule supports these WebDav commands:

<table>
<thead>
<tr>
<th>Use this command</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>get [path] file</td>
<td>Get a resource.</td>
</tr>
<tr>
<td>head [path] file</td>
<td>Get header info for a resource. (currently works same as get)</td>
</tr>
</tbody>
</table>

Input attachments

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| RESOURCEURI | The resource URI of the resource you want to retrieve from Library Manager. Here is an example of the format for the resource URI: `/userid/config/filetype/resource`  
Here are some examples:  
`/cjr/rpex1/ddd/master.dtt`  
`/jdoe/RPEX1/DDT/MASTER_0000100001_20030707.DDT`  
If the resource file name does not contain version, revision, and archive effective date information, the DPRLbyGet rule retrieves the last version and revision for the resource specified. Use the DPRLbyGet rule to get or check out a resource from Library Manager. |
| USERID     | (Optional) The user ID you want to use for the get operation. If you include this attachment variable, it overrides the user ID provided as part of the resource URI.  
If the user ID is missing as an attachment variable and in the resource URI, the rule will fail. |

WEBDAVERRORCODE

This attachment variable is only present if RESULTS equals ERROR. It can contain one of these values:

- 404 - (WebDav 'not found' error code) - The RESOURCEURI cannot be found.
- 409 - (WebDav 'conflict' error code) - The RESOURCEURI specified is invalid.
- 420 - (WebDav 'method error' error code) - An internal API error or memory error occurred.

Input attachments

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| USERID     | (Optional) The user ID you want to use for the get operation. If you include this attachment variable, it overrides the user ID provided as part of the resource URI.  
If the user ID is missing as an attachment variable and in the resource URI, the rule will fail. |
Input rule arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECKOUT</td>
<td>If you include this rule argument and set its value to Yes, the rule tries to check out (get and lock) the resource specified. This is useful for configuring this rule for a check-out or get request type.</td>
</tr>
</tbody>
</table>

Output attachments

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| PROPERTIES     | A rowset with a row for the resource specified in RESOURCEURI. The row contains the following properties for a file resource:  
- supportedlock - If locking is allowed, this XML string appears:  
  
  property:  
  <lockentry>  
  <lockscope>  
  <exclusive/>  
  </lockscope>  
  <locktype>  
  <write/>  
  </locktype>  
  </lockentry>  
- getContentLanguage - currently returns en_US.  
- resourcetype - blank if the resource is a file, otherwise collection if the resource is a file type/directory.  
- displayname - the display name of the resource.  
- HREF - the resource URL for this resource.  
- getlastmodified - a date and time indicating when the resource was last modified. This is a long value that contains the number of milliseconds since January 1, 1970.  
- getContentLength - currently zero (0) because there is no support for retrieving the file size of a document stored in Library Manager.  
- If a resource is locked these additional properties are returned:  
  - LOCKOWNER - The user ID that set the lock.  
  - LOCKSCOPE - The scope of the lock (exclusive).  
  - LOCKSUBJECT - The name of the resource locked.  
  - LOCKDEPTH - The depth of the resource locked (0).  
  - LOCKTYPE - The type of lock (write).  
  - LOCKTIMEOUT - The time-out value after which the lock will expire (infinity).  
  - LOCKTOKEN - A unique ID that identifies the resource locked.  
- This rowset is only present if RESULTS contains SUCCESS. |
| RESULTS        | SUCCESS or ERROR |
Using the Internet Document Server SDK

**DPRLbyPut**

Use this rule to add a new resource or to check in (unlock and put) an existing resource into Library Manager. You can add a new resource or put an existing resource into Library Manager.

If the resource is new, its version and revision will be 00001. If the resource is an existing one and it is locked by the same user ID performing the put operation, the resource will be put into Library Manager with a new version and revision.

This rule supports the following WebDav commands:

<table>
<thead>
<tr>
<th>Use this command</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>put [path]</td>
<td>Put a file into Library Manager.</td>
</tr>
</tbody>
</table>

Keep in mind that if a put operation is attempted on an existing resource and the version and revision specified is not the latest one, the put operation will fail. The system only supports put operations for new documents or for the last existing version and revision which must be locked prior to the put call.

**Input attachments**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESOURCEREF</td>
<td>A resource URI specifying the resource you want to place into Library Manager. Here is an example of the format of the URI: /userid/config/filetype/resource/ Here are some examples: /cjr/rpex1/ddt/master.ddt /jdoe/RPEX1/DDT/MASTER_0000100001_20030707.DDT Keep in mind that if the resource file name in RESOURCEREF does not contain version, revision, and archive effective date information, the DPRLbyPut rule tries to put the last version and revision of the file resource you specified.</td>
</tr>
</tbody>
</table>
Using the IDSWebdav Servlet Client APIs and DPRLib Rules

Output attachments

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USERID</td>
<td>(Optional) The user ID you want to use for the put operation. If this attachment variable is present, it overrides the user ID provided in the resource URI. If the user ID is missing from the attachment variable and from the resource URI, the rule will fail. For put operations with an existing resource, the user ID must match that of the locked record or the put operation will fail.</td>
</tr>
<tr>
<td>ARCEFFECTIVEDATE</td>
<td>(Optional) An archive effective date. Here is the format for this attachment variable: MM/DD/YYYY If this variable is present, its value is used as the archive effective date for the put operation. If it is missing, the rule uses the current date as the archive effective date.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESULTS</td>
<td>SUCCESS or ERROR.</td>
</tr>
<tr>
<td>WEBDAVERRORCODE</td>
<td>This attachment variable only exists if RESULTS equals ERROR. It can contain one of these values: 404 - (WebDav ‘not found’ error code) - The RESOURCEURI cannot be found. 409 - (WebDav ‘conflict’ error code) - The RESOURCEURI specified is invalid. 420 - (WebDav ‘method error’ error code) - An internal API error or memory error occurred. 423 - (WebDav ‘locked’ error code) - The resource is locked under a different user ID.</td>
</tr>
</tbody>
</table>

DPRLibLock

Use this rule to lock a resource in Library Manager. This rule supports the following WebDav commands:

<table>
<thead>
<tr>
<th>Use this command</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock [path] file</td>
<td>Locks a resource.</td>
</tr>
</tbody>
</table>
Chapter 1
Using the Internet Document Server SDK

Input attachments

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESOURCEURI</td>
<td>The resource URI of the resource you want to lock in Library Manager. Here is an example of the format for a resource URI: <code>/userid/config/filetype/resource</code> Here are some examples: <code>/cjr/rpex1/ddt/master.ddt</code> <code>/jdoe/RPEX1/DDT/MASTER_0000100001_20030707.DDT</code> If the resource file name in RESOURCEURI does not contain version, revision, and archive effective date information, the DPRLbyLock rule tries to lock the last version and revision of the file resource you specified.</td>
</tr>
<tr>
<td>USERID</td>
<td>(Optional) The user ID you want to use for the lock operation. If this attachment variable is present, it overrides the user ID provided as part of the resource URI. If the user ID is omitted from the attachment variable and from the resource URI, the rule will fail.</td>
</tr>
</tbody>
</table>

Output attachments

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCKOWNER</td>
<td>The user ID that owns the lock.</td>
</tr>
<tr>
<td>LOCKSCOPE</td>
<td>The scope of the lock (exclusive).</td>
</tr>
<tr>
<td>LOCKSUBJECT</td>
<td>The name of the resource locked.</td>
</tr>
<tr>
<td>LOCKDEPTH</td>
<td>The depth of the resource locked (0).</td>
</tr>
<tr>
<td>LOCKTYPE</td>
<td>The type of lock (write).</td>
</tr>
<tr>
<td>LOCKTIMEOUT</td>
<td>The time-out value after which the lock will expire (infinity).</td>
</tr>
<tr>
<td>LOCKTOKEN</td>
<td>A unique ID that identifies the resource locked.</td>
</tr>
<tr>
<td>RESULTS</td>
<td>SUCCESS or ERROR.</td>
</tr>
<tr>
<td>WEBAVERRORCODE</td>
<td>This attachment variable only exists if RESULTS equals ERROR. It can contain one of these values: 404 - (WebDav ‘not found’ error code) - The RESOURCEURI cannot be found. 409 - (WebDav ‘conflict’ error code) - The RESOURCEURI specified is invalid. 420 - (WebDav ‘method error’ error code) - An internal API error or memory error occurred. 423 - (WebDav ‘locked’ error code) - The resource is already locked.</td>
</tr>
</tbody>
</table>

DPRLbyUnlock

Use this rule to unlock a resource file in a library maintained by Library Manager. This rule supports the following WebDav commands:
Using the IDSWebdav Servlet Client APIs and DPRLIB Rules

### DPRLbyUnlock

Use this rule to unlock a resource in Library Manager. This rule can be used to unlock resource files by version and revision or by name, in which case the rule removes the latest version and revision for the resource file specified.

If the resource you specify is a collection (file type), all resource files for the collection will be unlocked, provided none are locked. This rule supports the following WebDav commands:

#### Input attachments

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESOURSEURI</td>
<td>The resource URI of the resource you want to unlock in Library Manager. Here is an example of the format for a resource URI: <code>/userid/config/filetype/resource</code> Here are some examples: <code>/cjr/rpex1/ddt/master.ddt</code> <code>/jdoe/RPEX1/DDT/MASTER_0000100001_20030707.DDT</code> If the resource file name in RESOURSEURI does not contain version, revision, and archive effective date information, the DPRLbyUnlock rule tries to unlock the last version and revision of the file resource specified.</td>
</tr>
<tr>
<td>USERID</td>
<td>(Optional) The user ID you want to use for the unlock operation. If this attachment variable is present, it overrides the user ID provided in the resource URI. If the user ID is omitted from the attachment variable and from the resource URI, the rule fails. If the user ID does not match the one for the locked record, the rule fails.</td>
</tr>
</tbody>
</table>

#### Output attachments

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESULTS</td>
<td>SUCCESS or ERROR.</td>
</tr>
<tr>
<td>WEBDAVERRORCODE</td>
<td>This attachment variable only exists if RESULTS equals ERROR. It can contain one of these values: 404 - (WebDav 'not found' error code) - The RESOURSEURI cannot be found. 409 - (WebDav 'conflict' error code) - The RESOURSEURI specified is invalid. 420 - (WebDav 'method error' error code) - An internal API error or memory error occurred. 423 - (WebDav 'locked' error code) - The resource is locked by another user.</td>
</tr>
</tbody>
</table>

### DPRLbyDelete

Use this rule to remove a resource or collection from Library Manager. This rule can remove a resource file by version and revision or by name, in which case the rule removes the latest version and revision for the resource file you specified.

If the resource you specify is a collection (file type), all resources for the collection will be removed, provided none are locked. This rule supports these WebDav commands:

<table>
<thead>
<tr>
<th>Use this command</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>delete [path] file</td>
<td>Delete a resource.</td>
</tr>
</tbody>
</table>
DPRLbyOptions

Use this rule to display the WebDav commands supported by Library Manager. This rule supports these WebDav commands:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESOURCEURI</td>
<td>The resource URI of the resource you want to delete from Library Manager. Here is an example of the format you should use:</td>
</tr>
<tr>
<td></td>
<td>/userid/config/filetype/resource</td>
</tr>
<tr>
<td></td>
<td>Here are some examples:</td>
</tr>
<tr>
<td></td>
<td>/cjr/rpex1/ddt/master.ddt</td>
</tr>
<tr>
<td></td>
<td>/jdoe/RPEX1/DDT/MASTER_000100001_20030707.DDT</td>
</tr>
<tr>
<td></td>
<td>If the resource file name in RESOURCEURI does not contain version, revision, and archive effective date information, the DPRLbyDelete rule tries to delete the last version and revision of the file resource you specified.</td>
</tr>
<tr>
<td>RESULTS</td>
<td>(Optional) This variable is only generated by the DPRLby rules running prior to this rule in the same request type, such as the DPRLbyGet and DPRLbyCopy rules running in the WEBDAVMOVE request type. If this variable exists and is set to ERROR — indicating either the DPRLbyGet or DPRLbyCopy rule failed — this rule will not execute.</td>
</tr>
<tr>
<td>WEBDAVERRCODE</td>
<td>(Optional) This variable is only generated by DPRLby rules running prior to this rule in the same request type, such as the DPRLbyGet and DPRLbyCopy rules running in the WEBDAVMOVE request type. If this variable exists — indicating that either the DPRLbyGet or DPRLbyCopy rule failed — this rule will not execute.</td>
</tr>
</tbody>
</table>

Output attachments

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESULTS</td>
<td>SUCCESS or ERROR.</td>
</tr>
<tr>
<td>WEBDAVERRCODE</td>
<td>This attachment variable is only present if RESULTS equals ERROR. It can contain one of these values:</td>
</tr>
<tr>
<td></td>
<td>404 - (WebDav 'not found' error code) - The RESOURCEURI cannot be found.</td>
</tr>
<tr>
<td></td>
<td>409 - (WebDav 'conflict' error code) - The RESOURCEURI specified is invalid.</td>
</tr>
<tr>
<td></td>
<td>420 - (WebDav 'method error' error code) - An internal API error or memory error occurred.</td>
</tr>
<tr>
<td></td>
<td>423 - (WebDav 'locked' error code) - The resource is locked.</td>
</tr>
</tbody>
</table>
Using the IDSWebdav Servlet Client APIs and DPRLIB Rules

This rule displays the following WebDav commands that are supported by Library Manager:

<table>
<thead>
<tr>
<th>options</th>
<th>get</th>
<th>head</th>
</tr>
</thead>
<tbody>
<tr>
<td>propfind</td>
<td>propgetall</td>
<td>lock</td>
</tr>
<tr>
<td>unlock</td>
<td>delete</td>
<td>copy</td>
</tr>
<tr>
<td>move</td>
<td>proppatch</td>
<td>mkcol</td>
</tr>
</tbody>
</table>

**Input attachments**
- None

**Output attachments**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTIONS</td>
<td>A comma-delimited string of WebDav commands supported by Library Manager.</td>
</tr>
<tr>
<td>RESULTS</td>
<td>Success.</td>
</tr>
</tbody>
</table>

**DPRLbyCopy**

Use this rule to copy a resource from one location to another, such as from one library to another. Keep in mind...

- The resource and destination file names *must match*.
- The config value for the resource *must differ* from the config value for the destination.

If the resource you are copying does not exist in the destination library, it will be added as a new resource with a version and revision of 00001. If the resource being copied exists in the destination, it will be added as a new version and revision; this is true regardless of what version and revision was specified for the resource or destination file names.

The DPRLbyCopy rule supports these WebDav commands:

<table>
<thead>
<tr>
<th>Use this command</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy [source] [destination]</td>
<td>Copies a resource from one location to another.</td>
</tr>
</tbody>
</table>

**Input attachments**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBYFILE</td>
<td>The resource you want to use for the copy operation. A full path and file name generated by DPRLbyGet rule, which should be run before this rule in the WEBDAVCOPY request type.</td>
</tr>
</tbody>
</table>
Using the Internet Document Server SDK

DPRLbyPropPatch

Use this rule to set or remove properties defined on the resource identified by the RESOURCEURI. This rule supports these WebDav commands:

<table>
<thead>
<tr>
<th>Use this command</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>proppatch</td>
<td>Not supported by Library Manager.</td>
</tr>
</tbody>
</table>
The proppatch command is not supported by Library Manager. You cannot modify the properties for records in Library Manager. This rule always returns RESULTS set to ERROR and WEBDAVERROCODE set to method not allowed.

**Input attachments** None

**Output attachments**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESULTS</td>
<td>ERROR.</td>
</tr>
<tr>
<td>WEBDAVERROCODE</td>
<td>This attachment variable only exists if RESULTS contains ERROR, which in this case is always true. It will contain this value: 405 - (WebDav 'method not allowed' error code) - The server does not allow or support this method.</td>
</tr>
</tbody>
</table>

**DPRLbyMKCol**

Use this rule to create a collection in Library Manager. This rule supports these WebDav commands:

<table>
<thead>
<tr>
<th>Use this command</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>mkcol</td>
<td>Not supported by Library Manager.</td>
</tr>
</tbody>
</table>

Keep in mind the mkcol command is not supported by Library Manager. You cannot make new collections (file types) in Library Manager without first adding a resource of that type.

This rule always returns RESULTS set to ERROR and WEBDAVERROCODE set to unsupported media type.

**Input attachments** None

**Output attachments**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESULTS</td>
<td>ERROR.</td>
</tr>
<tr>
<td>WEBDAVERROCODE</td>
<td>This attachment variable only exists if RESULTS equals ERROR, which in this case is always true. It contains this value: 415 - (WebDav 'unsupported media type' error code) - The server does not support or understand the mkcol request type.</td>
</tr>
</tbody>
</table>
WebDav Request Types for Library Manager

You should use the following request types with Library Manager:

```xml
<section name="ReqType:WEBDAVOPTIONS">
  <entry name="function">atcw32->ATCLoadAttachment</entry>
  <entry name="function">atcw32->ATCUnloadAttachment</entry>
  <entry name="function">dprw32->DPRSetConfig</entry>
  <entry name="function">dprw32->DPRInitLby</entry>
  <entry name="function">dprw32->DPRLbyPropFind</entry>
</section>

<section name="ReqType:WEBDAVPROPFIND">
  <entry name="function">atcw32->ATCLoadAttachment</entry>
  <entry name="function">atcw32->ATCUnloadAttachment</entry>
  <entry name="function">dprw32->DPRSetConfig</entry>
  <entry name="function">dprw32->DPRInitLby</entry>
  <entry name="function">dprw32->DPRLbyPropFind</entry>
</section>

<section name="ReqType:WEBDAVGET">
  <entry name="function">atcw32->ATCLoadAttachment</entry>
  <entry name="function">atcw32->ATCUnloadAttachment</entry>
  <entry name="function">dprw32->DPRSetConfig</entry>
  <entry name="function">dprw32->DPRInitLby</entry>
  <entry name="function">dprw32->DPRLbyGet</entry>
  <entry name="function">atcw32->ATCSendFile,RESOURCE,LBYFILE,BINARY</entry>
</section>

<section name="ReqType:WEBDAVHEAD">
  <entry name="function">atcw32->ATCLoadAttachment</entry>
  <entry name="function">atcw32->ATCUnloadAttachment</entry>
  <entry name="function">dprw32->DPRSetConfig</entry>
  <entry name="function">dprw32->DPRInitLby</entry>
  <entry name="function">dprw32->DPRLbyGet</entry>
  <entry name="function">atcw32->ATCSendFile,RESOURCE,LBYFILE,BINARY</entry>
</section>

<section name="ReqType:WEBDAVPUT">
  <entry name="function">atcw32->ATCLoadAttachment</entry>
  <entry name="function">atcw32->ATCUnloadAttachment</entry>
  <entry name="function">dprw32->DPRSetConfig</entry>
  <entry name="function">dprw32->DPRInitLby</entry>
  <entry name="function">dprw32->DPRLbyPut</entry>
</section>

<section name="ReqType:WEBDAVCHECKOUT">
  <entry name="function">atcw32->ATCLoadAttachment</entry>
  <entry name="function">atcw32->ATCUnloadAttachment</entry>
  <entry name="function">dprw32->DPRSetConfig</entry>
  <entry name="function">dprw32->DPRInitLby</entry>
  <entry name="function">dprw32->DPRLbyGet,CheckOut=Yes</entry>
  <entry name="function">atcw32->ATCSendFile,RESOURCE,LBYFILE,BINARY</entry>
</section>

<section name="ReqType:WEBDAVCHECKIN">
  <entry name="function">atcw32->ATCLoadAttachment</entry>
  <entry name="function">atcw32->ATCUnloadAttachment</entry>
  <entry name="function">dprw32->DPRSetConfig</entry>
  <entry name="function">dprw32->DPRInitLby</entry>
  <entry name="function">dprw32->DPRLbyPut</entry>
</section>
```
<section name="ReqType:WEBDAVLOCK">
    <entry name="function">atcw32->ATCLoadAttachment</entry>
    <entry name="function">dprw32->DPRSetConfig</entry>
    <entry name="function">dprw32->DPRInitLby</entry>
    <entry name="function">dprw32->DPRLbyLock</entry>
</section>

<section name="ReqType:WEBDAVUNLOCK">
    <entry name="function">atcw32->ATCLoadAttachment</entry>
    <entry name="function">dprw32->DPRSetConfig</entry>
    <entry name="function">dprw32->DPRInitLby</entry>
    <entry name="function">dprw32->DPRLbyUnlock</entry>
</section>

<section name="ReqType:WEBDAVDELETE">
    <entry name="function">atcw32->ATCLoadAttachment</entry>
    <entry name="function">dprw32->DPRSetConfig</entry>
    <entry name="function">dprw32->DPRInitLby</entry>
    <entry name="function">dprw32->DPRLbyDelete</entry>
</section>

<section name="ReqType:WEBDAVCOPY">
    <entry name="function">atcw32->ATCLoadAttachment</entry>
    <entry name="function">dprw32->DPRSetConfig</entry>
    <entry name="function">dprw32->DPRInitLby</entry>
    <entry name="function">dprw32->DPRLbyGet</entry>
    <entry name="function">dprw32->DPRLbyCopy</entry>
</section>

<section name="ReqType:WEBDAVMOVE">
    <entry name="function">atcw32->ATCLoadAttachment</entry>
    <entry name="function">dprw32->DPRSetConfig</entry>
    <entry name="function">dprw32->DPRInitLby</entry>
    <entry name="function">dprw32->DPRLbyGet</entry>
    <entry name="function">dprw32->DPRLbyCopy</entry>
    <entry name="function">dprw32->DPRLbyDelete</entry>
</section>

<section name="ReqType:WEBDAVPROPPATCH">
    <entry name="function">atcw32->ATCLoadAttachment</entry>
    <entry name="function">dprw32->DPRLbyPropPatch</entry>
</section>

<section name="ReqType:WEBDAVMKCOL">
    <entry name="function">atcw32->ATCLoadAttachment</entry>
    <entry name="function">dprw32->DPRLbyMKCol</entry>
</section>
Using File System Rules

In addition to the DPRLIB Library Manager rules for WebDav support, version 2.0 also comes with a set of Java rules you can use to perform file system updates on the server side via WebDav commands submitted by the IDSWebdavServlet client component.

The file system rules include:

- propFind on page 42
- get on page 44
- put on page 45
- lock on page 46
- unlock on page 47
- delete on page 48
- options on page 49
- copy on page 49
- move on page 50
- propPatch on page 51
- mkCol on page 52

File system request types

To use the file system rules, replace Library Manager request types with the following file system request types:

```xml
<!-- ***Begin WebDav rules for a file system. -->
<section name="ReqType:WEBDAVOPTIONS">
  <entry name="function">java;com.docucorp.ids.rules.WebdavFileSystemRule;;static;options;FILE,webdavfilesystem.properties</entry>
</section>
<section name="ReqType:WEBDAVPROPFIND">
  <entry name="function">java;com.docucorp.ids.rules.WebdavFileSystemRule;;transaction;propFind;FILE,webdavfilesystem.properties</entry>
</section>
<section name="ReqType:WEBDAVGET">
  <entry name="function">java;com.docucorp.ids.rules.WebdavFileSystemRule;;transaction;get;FILE,webdavfilesystem.properties</entry>
</section>
<section name="ReqType:WEBDAVPUT">
  <entry name="function">java;com.docucorp.ids.rules.WebdavFileSystemRule;;transaction;put;FILE,webdavfilesystem.properties</entry>
</section>
<section name="ReqType:WEBDAVHEAD">
  <entry name="function">java;com.docucorp.ids.rules.WebdavFileSystemRule;;transaction;get;FILE,webdavfilesystem.properties</entry>
</section>
<section name="ReqType:WEBDAVLOCK">
  <!-- ***End WebDav rules for a file system. -->
```

```
You must also create a file system directory on the IDS side. The file system directory must reside on a location accessible to IDS and should contain the resources that should be updated via WebDav commands. In addition, each of the Java rules listed above uses a FILE argument which points to a properties file with settings for the file system. Here is a sample properties file:

WDROOTNAME=/idswebdav/
WDROOTDIR=c:/ids/idswebdav/
**propFind**

Use this rule to return properties for a resource or collection. This rule supports these WebDAV commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ls [path]</td>
<td>Lists contents of a collection.</td>
</tr>
<tr>
<td>cd [path]</td>
<td>Changes a directory.</td>
</tr>
<tr>
<td>propget [path [property]]</td>
<td>Gets a property.</td>
</tr>
<tr>
<td>propfind [path [property]]</td>
<td>Finds a property.</td>
</tr>
<tr>
<td>propgetall [path]</td>
<td>Lists all properties for a resource.</td>
</tr>
</tbody>
</table>

**Input attachments**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESOURCEURI</td>
<td>A resource URI specifying a collection or resource. Here are some examples:</td>
</tr>
<tr>
<td></td>
<td>/collection/resource/</td>
</tr>
<tr>
<td></td>
<td>/resource</td>
</tr>
<tr>
<td></td>
<td>/collection</td>
</tr>
<tr>
<td>DEPTH</td>
<td>Enter one (1) for collections. Enter zero (0) for file resources.</td>
</tr>
</tbody>
</table>
### Output attachments

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| PROPERTIES | A rowset of rows that match each of the file resources available for a particular collection. If you set DEPTH to one (1) and RESOURCEURI specifies a collection, the PROPERTIES rowset returns a row for each resource available in the collection. If you set DEPTH to zero (0) and RESOURCEURI specifies a file resource, the PROPERTIES rowset returns a single row with the properties for the resource specified. Each row in the PROPERTIES rowset contains the following properties for a file resource: supportedlock - If locking is allowed, the following XML string is displayed for this property:  

```xml
<lockentry>
  <lockscope>
    <exclusive/>
  </lockscope>
  <locktype>
    <write/>
  </locktype>
</lockentry>
```

getContentLanguage - currently, the value *en_US*. 
resourcetype - blank if the resource is a file, otherwise *collection* if the resource is a file type or directory. 
displayname - the display name of the resource. 
HREF - the resource URI for this resource. 
getlastmodified - a date and time indicating when the resource was last modified. This is a long value that contains the number of milliseconds since January 1, 1970. 
getContentLength - currently, always zero because there is no support for retrieving the file size of a document stored in Library Manager. If a resource is locked, these additional properties are returned: LOCKOWNER - The user ID that owns the lock. LOCKSCOPE - The scope of the lock (exclusive). LOCKSUBJECT - The name of the resource locked. LOCKDEPTH - The depth of the resource locked (0). LOCKTYPE - The type of lock (write). LOCKTIMEOUT - The time-out value after which the lock will expire (infinity). LOCKTOKEN - A unique ID that identifies the resource locked. This rowset is only present if RESULTS equals SUCCESS. |
| RESULTS   | SUCCESS or ERROR. |
**get**

Use this rule to return a resource from the file system. This rule supports these WebDav commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get [path] file</td>
<td>Gets a resource.</td>
</tr>
<tr>
<td>head [path] file</td>
<td>Gets header information for a resource. (works same as get)</td>
</tr>
</tbody>
</table>

**Variable Description**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| WEBDAERRORCODE | This attachment variable only exists if RESULTS equals ERROR. It can contain one of these values: 
404 - (WebDav 'not found' error code) - The RESOURCEURI cannot be found.
409 - (WebDav 'conflict' error code) - The RESOURCEURI specified is invalid.
420 - (WebDav 'method error' error code) - An internal API error or memory error occurred. |
| RESOURCEURI  | The resource URI of the resource you want to retrieve. Here is an example: /collection/resource |
### Output attachments

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROPERTIES</td>
<td>A rowset with a row for the resource specified in RESOURCEURI. The row contains the following properties for a resource: supportedlock - If locking is allowed, the following XML string is displayed for this property:</td>
</tr>
<tr>
<td></td>
<td><code>&lt;lockentry&gt;</code>&lt;br&gt;  <code>&lt;lockscope&gt;</code>&lt;br&gt;  <code>&lt;exclusive/&gt;</code>&lt;br&gt;  <code>&lt;/lockscope&gt;</code>&lt;br&gt;  <code>&lt;locktype&gt;</code>&lt;br&gt;  <code>&lt;write/&gt;</code>&lt;br&gt;  <code>&lt;/locktype&gt;</code>&lt;br&gt;  <code>&lt;/lockentry&gt;</code>&lt;br&gt;</td>
</tr>
<tr>
<td></td>
<td>getContentLanguage - currently, the value en_US.&lt;br&gt; resourceType - blank if the resource is a file, otherwise collection if the resource is a file type or directory.&lt;br&gt;  displayname - the display name of the resource.&lt;br&gt;  HREF - the resource URI for this resource.&lt;br&gt;  getlastmodified - a date and time indicating when the resource was last modified. This is a long value that contains the number of milliseconds since January 1, 1970.&lt;br&gt;  getContentLength - currently, always zero because there is no support for retrieving the file size of a document stored in Library Manager.&lt;br&gt;  LOCKOWNER - The user ID that owns the lock.&lt;br&gt;  LOCKSCOPE - The scope of the lock (exclusive).&lt;br&gt;  LOCKSUBJECT - The name of the resource locked.&lt;br&gt;  LOCKDEPTH - The depth of the resource locked (0).&lt;br&gt;  LOCKTYPE - The type of lock (write).&lt;br&gt;  LOCKTIMEOUT - The time-out value after which the lock will expire (infinity).&lt;br&gt;  LOCKTOKEN - A unique ID that identifies the resource locked. This rowset is only present if RESULTS equals SUCCESS.</td>
</tr>
<tr>
<td>RESULTS</td>
<td>SUCCESS or ERROR.</td>
</tr>
<tr>
<td>WEBDAVERRORCODE</td>
<td>This attachment variable is only present if RESULTS equals ERROR. It can contain one of these values:&lt;br&gt;  404 - (WebDav 'not found' error code) - The RESOURCEURI cannot be found.&lt;br&gt;  409 - (WebDav 'conflict' error code) - The RESOURCEURI specified is invalid.&lt;br&gt;  420 - (WebDav 'method error' error code) - An internal API error or memory error occurred.</td>
</tr>
</tbody>
</table>
### put

<table>
<thead>
<tr>
<th><strong>Command</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>put [path]</code></td>
<td>Puts the specified file into Library Manager.</td>
</tr>
</tbody>
</table>

If the resource is locked, the put operation will fail.

**Input attachments**

<table>
<thead>
<tr>
<th><strong>Variable</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>RESOURCEURI</code></td>
<td>A resource URI that specifies the resource you want to place into the file system. Here is an example: /collection/resource/</td>
</tr>
</tbody>
</table>

**Output attachments**

<table>
<thead>
<tr>
<th><strong>Variable</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>RESULTS</code></td>
<td>SUCCESS or ERROR.</td>
</tr>
<tr>
<td><code>WEBDAVERRORCODE</code></td>
<td>This attachment variable only exists if RESULTS equals ERROR. It can contain one of these values:</td>
</tr>
<tr>
<td></td>
<td>404 - (WebDav 'not found' error code) - The RESOURCEURI cannot be found.</td>
</tr>
<tr>
<td></td>
<td>409 - (WebDav 'conflict' error code) - The RESOURCEURI specified is invalid.</td>
</tr>
<tr>
<td></td>
<td>420 - (WebDav 'method error' error code) - An internal API error or memory error occurred.</td>
</tr>
<tr>
<td></td>
<td>423 - (WebDav 'locked' error code) - The resource is locked.</td>
</tr>
</tbody>
</table>

### lock

Use this rule to lock a resource in the file system. This rule supports these WebDav commands:

<table>
<thead>
<tr>
<th><strong>Command</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lock [path]</code></td>
<td>Locks a resource.</td>
</tr>
</tbody>
</table>

**Input attachments**

<table>
<thead>
<tr>
<th><strong>Variable</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>RESOURCEURI</code></td>
<td>The resource URI of the resource that should be locked in the file system. Here is an example: /collection/resource</td>
</tr>
</tbody>
</table>

**Output attachments**

<table>
<thead>
<tr>
<th><strong>Variable</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>LOCKOWNER</code></td>
<td>The user ID that owns the lock.</td>
</tr>
<tr>
<td><code>LOCKSCOPE</code></td>
<td>The scope of the lock (exclusive).</td>
</tr>
</tbody>
</table>
### Unlock

Use this rule to unlock a resource in the file system. This rule supports these WebDav commands:

**Command** | **Description**
---|---
`unlock [path] file` | Unlock a resource.

**Input attachments**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESOURCEURI</td>
<td>The resource URI of the resource that should be unlocked. Here is an example: <code>/collection/resource</code></td>
</tr>
</tbody>
</table>

**Output attachments**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESULTS</td>
<td>SUCCESS or ERROR.</td>
</tr>
</tbody>
</table>
**delete**

Use this rule to remove a resource or collection from the file system. If the resource you specified is a collection, all resources for the collection will be removed, provided none are locked. This rule supports these WebDAV commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delete [path] file</td>
<td>Delete a resource.</td>
</tr>
</tbody>
</table>

**Input attachments**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESOURCEURI</td>
<td>The resource URI of the resource you want to delete. Here are some examples:</td>
</tr>
<tr>
<td></td>
<td>/collection/resource</td>
</tr>
<tr>
<td></td>
<td>/collection</td>
</tr>
<tr>
<td>DEPTH</td>
<td>(Optional) If a depth value is specified for collections, its value must be</td>
</tr>
<tr>
<td></td>
<td>set to infinity. If a depth value is omitted, the rule assumes a depth of</td>
</tr>
<tr>
<td></td>
<td>infinity. You do not have to provide a depth value for a file resource.</td>
</tr>
</tbody>
</table>

**Output attachments**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESULTS</td>
<td>SUCCESS or ERROR.</td>
</tr>
<tr>
<td>WEBDAVERRORCODE</td>
<td>This attachment variable only exists if RESULTS equals ERROR. It can contain one of these values:</td>
</tr>
<tr>
<td></td>
<td>404 - (WebDAV 'not found' error code) - The RESOURCEURI cannot be found.</td>
</tr>
<tr>
<td></td>
<td>409 - (WebDAV 'conflict' error code) - The RESOURCEURI specified is invalid.</td>
</tr>
<tr>
<td></td>
<td>420 - (WebDAV 'method error' error code) - An internal API error or memory error occurred.</td>
</tr>
<tr>
<td></td>
<td>423 - (WebDAV 'locked' error code) - The resource is locked by another user.</td>
</tr>
</tbody>
</table>
options

Use this rule to display the WebDav commands supported by the file system. This rule supports these WebDav commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>display options available for path or URL.</td>
</tr>
</tbody>
</table>

This rule displays these WebDav commands that are supported by the file system:

<table>
<thead>
<tr>
<th>options</th>
<th>get</th>
<th>head</th>
</tr>
</thead>
<tbody>
<tr>
<td>propfind</td>
<td>propgetall</td>
<td>lock</td>
</tr>
<tr>
<td>unlock</td>
<td>delete</td>
<td>copy</td>
</tr>
<tr>
<td>move</td>
<td>proppatch</td>
<td>mkcol</td>
</tr>
</tbody>
</table>

Input attachments
None

Output attachments

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTIONS</td>
<td>A comma-delimited string of WebDav commands supported by the file system.</td>
</tr>
<tr>
<td>RESULTS</td>
<td>Success.</td>
</tr>
</tbody>
</table>

copy

Use this rule to copy a resource or collection from one location to another. This rule supports these WebDav commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy</td>
<td>Copies a resource.</td>
</tr>
</tbody>
</table>

If any destination resource exists and is locked, the copy operation fails. If any destination resource exists and the overwrite flag is set to false, the copy operation fails.

Input attachments

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESOURCEURI</td>
<td>The resource you want to use for the copy operation. Here is an example: /collection/resource</td>
</tr>
<tr>
<td>DESTINATIONURI</td>
<td>A URI containing the destination of the resource you want to copy. Here is an example: /collection/destination</td>
</tr>
</tbody>
</table>
Output attachments

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPTH</td>
<td>A depth indicator. Used for copying collections. If you omit the depth for a collection, the rule assumes a depth of infinity. If you enter anything other than infinity for a collection, the rule only copies the collection directory. You do not have to provide a depth value for a file resource.</td>
</tr>
<tr>
<td>OVERWRITE</td>
<td>An overwrite flag indicator. If any resource in the destination already exists and the overwrite flag is set to True, the copy operation proceeds, otherwise it will fail.</td>
</tr>
</tbody>
</table>

Input attachments

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESOURCEURI</td>
<td>The resource you want to use for the move operation. Here is an example: /collection/resource</td>
</tr>
<tr>
<td>DESTINATIONURI</td>
<td>A URI containing the destination of the resource you want to move. Here is an example: /collection/destination</td>
</tr>
</tbody>
</table>

**move**

Use this rule to move a resource or collection from one location to another. This rule supports these WebDav commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>move [source] [destination]</td>
<td>Moves a resource.</td>
</tr>
</tbody>
</table>

If any destination or source resource exists and is locked, the move operation fails. If any destination resource exists and the overwrite flag is set to False, the move operation fails. If the resource you specify is a collection and its depth value is something other than infinity, the move operation fails.
Using the IDSWebdav Servlet Client APIs and DPRLIB Rules

### Output attachments

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPTH</td>
<td>A depth indicator used for moving collections. If you omit the depth for a collection, the rule assumes a depth of infinity. If you enter anything other than infinity for a collection, the rule fails. You do not have to provide a depth value for a file resource.</td>
</tr>
<tr>
<td>OVERWRITE</td>
<td>An overwrite flag indicator. If any resource in the destination already exists and the overwrite flag is set to True, the move operation proceeds, otherwise it fails.</td>
</tr>
</tbody>
</table>

### Input attachments

None

### Output attachments

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESULTS</td>
<td>SUCCESS or ERROR.</td>
</tr>
</tbody>
</table>
| WEBDAVERRORCODE | This attachment variable only exists if RESULTS equals ERROR. It can contain one of these values:  
403 (WebDAV 'forbidden' error code) - The source and destination URIs are the same.  
409 (WebDAV 'conflict' error code) - The resource cannot be created at the destination.  
420 (WebDAV 'method failure' error code) - An internal error or memory error occurred.  
423 (WebDAV 'locked' error code) - A source or existing destination resource was locked.  |

---

**propPatch**

Use this rule to set and remove properties defined on the resource identified by RESOURCEURI. This rule supports these WebDav commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>proppatch</td>
<td>Not supported by the file system.</td>
</tr>
</tbody>
</table>

The proppatch command is not supported by the file system. The system does not allow modification of properties for a resource in the file system.

---

**Variable Description**

| RESULTS | Error. |
| WEBDAVERRORCODE | This attachment variable only exists if RESULTS equals ERROR, which in this case is always true. It will contain the following value:  
405 - (WebDAV 'method not allowed' error code) - The server does not allow or support this method. |
**mkCol**

Use this rule to creates a collection in the file system. This rule supports these WebDav commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mkcol</td>
<td>Makes a collection.</td>
</tr>
</tbody>
</table>

The rule will fail if the collection already exists or if it failed to create the collection because one or more parents specified in RESOURCEURI does not exist.

**Input attachments**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESOURCEURI</td>
<td>The collection you want to create. Here is an example: /collection</td>
</tr>
</tbody>
</table>

**Output attachments**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESULTS</td>
<td>Success or Error.</td>
</tr>
<tr>
<td>WEBDAVERRORCODE</td>
<td>This attachment variable only exists if RESULTS equals ERROR. It can contain one of these values: 409 (WebDav 'conflict' error code) - The resource cannot be created at the destination. 420 (WebDav 'method failure' error code) - An internal error or memory error occurred.</td>
</tr>
</tbody>
</table>
Using the IDSWebdavServlet

The IDSWebdavServlet client component is a Java servlet that receives WebDav requests from WebDav client programs and submits them to IDS for processing.

Follow these steps to use the IDSWebdavServlet:

1. Create an idswebdav directory under the JSP engine webapps directory. Make sure the name is in lowercase.
2. Add IDSWebDavServlet.jar to the common\lib directory of the JSP engine.
3. Make sure the idswebdav directory contains a sub directory named WEB-INF. Make sure the name is in uppercase.
4. Add the following web.xml file to the WEB-INF directory.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <servlet>
    <servlet-name>idswebdav</servlet-name>
    <servlet-class>com.docucorp.ids.webdav.IDSWebdavServlet</servlet-class>
    <init-param>
      <param-name>debug</param-name>
      <param-value>0</param-value>
    </init-param>
    <init-param>
      <param-name>listings</param-name>
      <param-value>true</param-value>
    </init-param>
    <!-- Uncomment this to enable read and write access -->
    <!--<init-param>
      <param-name>readonly</param-name>
      <param-value>false</param-value>
    </init-param> -->
    <!--load-on-startup>1</load-on-startup-->  
    <!--</init-param> -->
  </servlet>
  <servlet-mapping>
    <servlet-name>idswebdav</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
  <!-- Establish the default MIME type mappings -->
  <mime-mapping>
    <extension>txt</extension>
    <mime-type>text/plain</mime-type>
  </mime-mapping>
</web-app>
```
<mime-mapping>
  <extension>csh</extension>
  <mime-type>application/x-csh</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>sh</extension>
  <mime-type>application/x-sh</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>tcl</extension>
  <mime-type>application/x-tcl</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>tex</extension>
  <mime-type>application/x-tex</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>texinfo</extension>
  <mime-type>application/x-texinfo</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>texi</extension>
  <mime-type>application/x-texinfo</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>t</extension>
  <mime-type>application/x-troff</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>tr</extension>
  <mime-type>application/x-troff</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>roff</extension>
  <mime-type>application/x-troff</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>man</extension>
  <mime-type>application/x-troff-man</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>me</extension>
  <mime-type>application/x-troff-me</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>ms</extension>
  <mime-type>application/x-wais-source</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>src</extension>
  <mime-type>application/x-wais-source</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>zip</extension>
  <mime-type>application/zip</mime-type>
</mime-mapping>
<extension>bcpio</extension>
<mime-type>application/x-bcpio</mime-type>
</mime-mapping>
<mime-mapping>
<extension>cpio</extension>
<mime-type>application/x-cpio</mime-type>
</mime-mapping>
<mime-mapping>
<extension>gtar</extension>
<mime-type>application/x-gtar</mime-type>
</mime-mapping>
<mime-mapping>
<extension>shar</extension>
<mime-type>application/x-shar</mime-type>
</mime-mapping>
<mime-mapping>
<extension>sv4cpio</extension>
<mime-type>application/x-sv4cpio</mime-type>
</mime-mapping>
<mime-mapping>
<extension>sv4crc</extension>
<mime-type>application/x-sv4crc</mime-type>
</mime-mapping>
<mime-mapping>
<extension>tar</extension>
<mime-type>application/x-tar</mime-type>
</mime-mapping>
<mime-mapping>
<extension>ustar</extension>
<mime-type>application/x-ustar</mime-type>
</mime-mapping>
<mime-mapping>
<extension>dvi</extension>
<mime-type>application/x-dvi</mime-type>
</mime-mapping>
<mime-mapping>
<extension>hdf</extension>
<mime-type>application/x-hdf</mime-type>
</mime-mapping>
<mime-mapping>
<extension>latex</extension>
<mime-type>application/x-latex</mime-type>
</mime-mapping>
<mime-mapping>
<extension>bin</extension>
<mime-type>application/octet-stream</mime-type>
</mime-mapping>
<mime-mapping>
<extension>oda</extension>
<mime-type>application/oda</mime-type>
</mime-mapping>
<mime-mapping>
<extension>pdf</extension>
<mime-type>application/pdf</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>ps</extension>
  <mime-type>application/postscript</mime-type>
</mime-mapping>

<mime-mapping>
  <extension>eps</extension>
  <mime-type>application/postscript</mime-type>
</mime-mapping>

<mime-mapping>
  <extension>ai</extension>
  <mime-type>application/postscript</mime-type>
</mime-mapping>

<mime-mapping>
  <extension>rtf</extension>
  <mime-type>application/rtf</mime-type>
</mime-mapping>

<mime-mapping>
  <extension>nc</extension>
  <mime-type>application/x-netcdf</mime-type>
</mime-mapping>

<mime-mapping>
  <extension>cdf</extension>
  <mime-type>application/x-netcdf</mime-type>
</mime-mapping>

<mime-mapping>
  <extension>cer</extension>
  <mime-type>application/x-x509-ca-cert</mime-type>
</mime-mapping>

<mime-mapping>
  <extension>exe</extension>
  <mime-type>application/octet-stream</mime-type>
</mime-mapping>

<mime-mapping>
  <extension>gz</extension>
  <mime-type>application/x-gzip</mime-type>
</mime-mapping>

<mime-mapping>
  <extension>z</extension>
  <mime-type>application/x-compress</mime-type>
</mime-mapping>

<mime-mapping>
  <extension>hqx</extension>
  <mime-type>application/mac-binhex40</mime-type>
</mime-mapping>

<mime-mapping>
  <extension>mif</extension>
  <mime-type>application/x-mif</mime-type>
</mime-mapping>

<mime-mapping>
  <extension>ief</extension>
  <mime-type>image/ief</mime-type>
</mime-mapping>
</mime-mapping>
<mime-mapping>
  <extension>tiff</extension>
  <mime-type>image/tiff</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>tif</extension>
  <mime-type>image/tiff</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>ras</extension>
  <mime-type>image/x-cmu-raster</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>pm</extension>
  <mime-type>image/x-portable-anymap</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>p</extension>
  <mime-type>image/x-portable-pixmap</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>rgb</extension>
  <mime-type>image/x-rgb</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>xbm</extension>
  <mime-type>image/x-xbitmap</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>xpm</extension>
  <mime-type>image/x-xpixmap</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>xwd</extension>
  <mime-type>image/x-xwindowdump</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>au</extension>
  <mime-type>audio/basic</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>snd</extension>
  <mime-type>audio/basic</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>aif</extension>
</mime-mapping>
<mime-type>audio/x-aiff</mime-type></mime-mapping>
<mime-mapping>
<extension>aiff</extension>
<mime-type>audio/x-aiff</mime-type></mime-mapping>
<mime-mapping>
<extension>aifc</extension>
<mime-type>audio/x-aiff</mime-type></mime-mapping>
<mime-mapping>
<extension>wav</extension>
<mime-type>audio/x-wav</mime-type></mime-mapping>
<mime-mapping>
<extension>mpeg</extension>
<mime-type>video/mpeg</mime-type></mime-mapping>
<mime-mapping>
<extension>mpg</extension>
<mime-type>video/mpeg</mime-type></mime-mapping>
<mime-mapping>
<extension>mpe</extension>
<mime-type>video/mpeg</mime-type></mime-mapping>
<mime-mapping>
<extension>qt</extension>
<mime-type>video/quicktime</mime-type></mime-mapping>
<mime-mapping>
<extension>mov</extension>
<mime-type>video/quicktime</mime-type></mime-mapping>
<mime-mapping>
<extension>avi</extension>
<mime-type>video/x-msvideo</mime-type></mime-mapping>
<mime-mapping>
<extension>movie</extension>
<mime-type>video/x-sgi-movie</mime-type></mime-mapping>
<mime-mapping>
<extension>avx</extension>
<mime-type>video/x-rad-screenplay</mime-type></mime-mapping>
<mime-mapping>
<extension>wrl</extension>
<mime-type>x-world/x-vrml</mime-type></mime-mapping>
<mime-mapping>
<extension>mpv2</extension>
<mime-type>video/mpeg2</mime-type></mime-mapping>
<!-- Establish the default list of welcome files -->
<welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
  <welcome-file>index.html</welcome-file>
  <welcome-file>index.htm</welcome-file>
</welcome-file-list>

<!--security-constraint-->
<web-resource-collection>
  <web-resource-name>The Entire Web Application</web-resource-name>
  <url-pattern>/*</url-pattern>
</web-resource-collection>
<auth-constraint>
  <role-name>tomcat</role-name>
</auth-constraint>
</security-constraint>

<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>Tomcat Supported Realm</realm-name>
</login-config>

<security-role>
  <description>
    An example role defined in "conf/tomcat-users.xml"
  </description>
  <role-name>tomcat</role-name>
</security-role>

</web-app>

5 Restart the JSP engine.

6 To send requests to the servlet, use the following URL format:

http://userid@boxname:port#/idswebdav/

where userid is the user ID used for the WebDav operations, boxname is the name of the box hosting the JSP engine plus the new idswebdav directory plus the port# is the port number, if any, of the JSP engine.

(An example WebDav client program that can be downloaded and used to send requests to the IDSWebdavServlet is the Jakarta slide client program.)

You can also use Windows’ Add Network Places wizard and add a new network place using a URL with the following format:

http://boxname:port#/idswebdav/userid/
Writing Processing Rules in Visual Basic

In this topic, you will learn how to write rules for the Internet Document Server using Microsoft Visual Basic (VB). Here you will learn how to:

- Use the VB rule wizard
- Add your rule to the DOCSERV configuration file
- Use general debugging techniques

You will also find a general overview of server support for Visual Basic rules.

You can write rules for the Internet Document Server in Visual Basic 5 by building VB class files into ActiveX DLLs. Using the DSI Visual Basic rule wizard and the steps outlined below, you can easily put together a rule.

The DSI Visual Basic API includes a project in the samples with a sample rule, Fish.vpb, which we’ll refer to in the discussion.

This illustration shows the general structure of Visual Basic rule processing. Notice that:

- All VB rule processing is routed through DSICoRul.DLL
- A VB rule DLL can have many rules within multiple classes in a single DLL

Structure

An ActiveX DLL created under VB has these naming levels:

- DLL name
- Class module name
- Rule name

VB does not articulate COM interfaces.

Multiple class modules are permitted as are multiple functions within each class module. As in C++, the function names are valid only when attached to their class—you can have the same function name in multiple classes.

Visual Basic maps these names to COM in this manner:

```
ProgID= <DLL name>.<Class Module name>
```

The COM ProgID appears in the registry and is the most common human-readable means by which a COM object is identified.

For instance, if you create a VB project Fish, with two classes, IBass and ITroun, each with two rules, the following will appear in the registry after you run regsvr32.exe Fish.DLL

```
ProgID  Fish.IBass
```
Interface: IBass
Methods: GoFish
StopFishing
ProgID Fish.ITrout
Interface: ITrout
Methods: GoFish
StopTheBoat

The DLL must be an ActiveX DLL and must contain at least one class module (.cls) with the public functions to be called by IDS. Continuing the above example, there will be these files in the ActiveX DLL project:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fish.vpb</td>
<td>Fish project</td>
</tr>
<tr>
<td>Fish.vpw</td>
<td>Fish work space</td>
</tr>
<tr>
<td>IBass.cls</td>
<td>IBass Class Module</td>
</tr>
<tr>
<td>ITrout.cls</td>
<td>ITrout Class Module</td>
</tr>
</tbody>
</table>

Installing the DSI VB rule wizard

To help you create VB rules, the system includes a VB add-in wizard. To install this wizard, run this command:

```
addinst.exe
```

Building rules with the wizard

The VB rule wizard will either work with an existing project or it can start a new one for you. Likewise, the wizard will create a new class for you or use one that’s already in an existing project.

To use, start the Visual Basic IDE and select Add-Ins, DSI Rule Wizard. The wizard guides you through the process of creating a template DSI VB rule. After the wizard has run, you will have at least the following:

- A Visual Basic project (.vbp)
- A Visual Basic workgroup (.vpw)
- A Visual Basic class file (.cls)

The code the rule wizard generates contains references to all possible messages that can be sent to a DSI rule. Although the VB compiler will drop processing of case statements that do not have any code, remove the unneeded case statements to make your code easier to read.

Next, add in your business logic.

Compile your ActiveX DLL. When you compile the project, DLL, LIB, and EXP files will be created. After you debug the project, you only need to copy the DLL to the IDS directory and register it—if and only if the server is on a different machine.
NOTE: If you are developing on a system different from IDS, you must move your DLL into the IDS directory.

If you are developing on the same system that is running IDS you should not move the DLL without registering it.

Add your rule to the DOCSERV configuration file (see below).

Test your rule under the server using DSICoTB - the DSI Test Bed program.

Troubleshooting

If you are getting messages about not being able to find your rule, consider the following:

DSICoRul may not be able to find your DLL in the IDS directory. ActiveX DLLs must be registered (they are COM objects). DSICoRul will register your DLL if you have not already done so but to do this it must be able to find the DLL. If you don’t want your DLL to be in the IDS directory, register it using this command:

```
regsvr32.exe <dllname>
```

DSICoRul first attempts to locate your rule in the system registry which contains a path to your DLL. When you compile your rule DLL, VB automatically registers it for you. If you then move the DLL, the registry will not be able to find it, which causes an error. Therefore, if you are developing on the same system as IDS, do not move your DLL to the server directory.

If the DLL is in the server directory or you have registered it yourself and DSICoRul is still complaining that it cannot find it, then it is time to start looking with the OLEVIEW.EXE program. If you do not already have this program on your system, you can find it on the MDSN CD or on Microsoft’s web site.

Start the OLEVIEW program and choose the File, View option. Enter Lib and point it at your DLL. The CoClasses folder will contain the names of your classes and within those, eventually, your methods (which are your rules). Check the program ID against the DOCSERV configuration file.

All VB rules will be specified as follows

```
<entry name="function">DSICoRUL->Invoke,COM OBJECT NAME->METHOD,OTHERPARMS</entry>
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSICoRul&gt;Invoke&gt;</td>
<td>Invoke provides the interface between the server and Visual Basic. When a rule is to be executed, IDS calls the Invoke entry point of DISCoRUL.DLL with the remainder of the line as parameters:</td>
</tr>
<tr>
<td>COM_OBJECT_NAME</td>
<td>a COM ProgID which flows naturally from VB and is composed of the name of the name of the DLL and the VB class separated by a period. The server user must register the COM object before starting the server.</td>
</tr>
<tr>
<td>METHOD</td>
<td>your VB rule</td>
</tr>
<tr>
<td>OTHERPARMS</td>
<td>other parameters in an alphabetic string</td>
</tr>
</tbody>
</table>
You must add at least two entries into the configuration file:

In the ReqType:INI control group, initialize DSICoRul by including this reference:

```xml
<section name="ReqType:THREADINI">
  ...
  <entry name="function">DSICoRUL::Init</entry>
</section>
```

Then add the specifications of your rule to the appropriate request. For instance, to add the TestRule,

```xml
<section name="ReqType:SSS">
  ...
  <entry name="function">DSICoRul::Invoke,TestRule.ITestRule::HelloWorld</entry>
</section>
```

### Interface

Each class module must contain at least one Public Function which will be the rule. Functions must be used as Subs do not support return values, which all rules must provide.

Each Public Function must conform to the following prototype:

```csharp
Public Function GoFish(ByRef oDSI As DSICoAPI, ByVal hInstance As Long, ByVal pszParms As String, ByVal ulMsg As Long, ByVal ulOptions As Long) As Long
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ByRef oDSICoAPI as DSICoAPI</td>
<td>The DSICoAPI object will provide access to the DSI API</td>
</tr>
<tr>
<td>ByVal hInstance As Long</td>
<td>The server message, dsiMSG_INIT, dsiMSG_RUNF, dsiMSG_RUNR, dsiMSG_TERM</td>
</tr>
<tr>
<td>ByVal pszParms As String</td>
<td>The parameter string passed in from the configuration file</td>
</tr>
<tr>
<td>ByVal ulOptions As Long</td>
<td>Reserved for future use</td>
</tr>
</tbody>
</table>

The public function will return the appropriate dsiERR, usually dsiERR_SUCCESS. If the message is unsupported, then dsiERR_MSGNOTFOUND must be returned to avoid the overhead of subsequent calls.

### Using global data methods

You can use global methods with DSICo. This lets you store data in one location for use with multiple IDS Servers. To do this, your configuration files must have identical settings for the Path option:
Writing Processing Rules in Visual Basic

<section name="ReqType:SSS">

NOTE: All servers that are required to share global data must have access to a single global data folder. You can use these global methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GlobalDataCreate</td>
<td>Lets you create a global entry file which you can retrieve later. The data is stored in the directory you define in the configuration file.</td>
</tr>
<tr>
<td>GlobalDataDestroy</td>
<td>Lets you remove the global data entry associated with GUID.</td>
</tr>
<tr>
<td>GlobalDataSize</td>
<td>Use this method to get the size of the data associated with GUID. You can use this information to create a buffer before calling the GlobalDataRead method.</td>
</tr>
<tr>
<td>GlobalDataRead</td>
<td>Use this method to read the contents of the global data entry.</td>
</tr>
<tr>
<td>GlobalDataClean</td>
<td>Use this method to remove expired files from the global data directory.</td>
</tr>
</tbody>
</table>

DSI API support

The DSICoAPI object is passed into the rule to provide easy access to the DSI API. If you want to write to the DSI API directly, DSI.bas contains the function prototypes but the advantages are few and the details that must be managed are many. For instance, VB strings are not null terminated so all strings must have + Chr(0) at the end.

Error handling

When IDS encounters fatal errors it passes those errors to your On Error routine, if there is one. In general, your error routine should pass the fatal error to DSI for logging. Errors which your program is normally expected to handle, like dsiERR_NOTFOUND (ERR.RAISE), will be available as a return value from DSICoAPI and should not be passed to the server.

Registration

Visual Basic automatically registers your ActiveX DLL when you compile it. DSICoRul will automatically register your ActiveX DLL if necessary, provided it can find the DLL and the file name is well formed.

Testing with IDS

To test under IDS you must also have the Visual C++ 5.0 debugger. The general procedure is detailed in Microsoft knowledge base article Q166275 (http://support.microsoft.com/support/kb/articles/q166/2/75.asp). The following procedure assumes you have read and understood this article.

Make sure your rule is compiled with Debug Info.

Bring up OLEVIEW.EXE, locate your rule DLL under “All Objects”. Click on the “+” sign to make OLEVIEW display the supported interfaces. This loads your ActiveX DLL.
Follow the procedure outlined in the knowledge base article. Since this is a DLL you must specify DSRVW32.EXE as the debug target in the settings. Also take care to set the working directory to the directory in which DSRVW32.EXE normally runs.

At this point you may use any program you like to initiate the transaction your rule will process. If you don’t have an application of your own, DSICoTB lets you build an attachment and hand it off to the server for processing.

**Miscellaneous Notes**

**GUIDs**

GUIDs are 128-bit values used to identify COM objects globally. IDS handles VB rules in such a way that you don’t have to worry about GUIDs in spite of the COM documentation’s warnings that you should never change a GUID once it goes into production.

**State and threads**

IDS can call your rule on any thread—that’s what the instance handle is for—and the thread state is held in the server. This means that your rules should be stateless. Stateless means that you don’t retain any information from one call to the next in the rule itself. If you want to pass some value from one rule to another or from one thread to another, use CreateValue and LocateValue.

**Sharing violations**

IDS holds a reference to your ActiveX DLL from the first time it is called until IDS is shut down. Expect a sharing violation if you try to replace your rule DLL without first shutting down the server.

**Crashing the server**

Remember, your rule will be running in-process. Loops (polling and bugs) can hang the server or degrade performance. Memory leaks can exhaust server memory, given enough time, so be careful.

**Check the server log**

Assuming the server survives the experience, many fatal errors, such as not being able to load your rule, are logged to DUTTRACE.LOG, found in the IDS directory.

**Performance**

If you are concerned about first-execution performance, such as how long it takes to load your rule DLL the first time, change the DLL load address in your VB project from the default. Using the default makes it likely there will be an expensive collision and relocation at load time.

**COM, ProgIDs, and VB**

The ProgID is a string that shows up in the registry to identify your classes. There are many Win32 APIs that deal with ProgID and scripting languages, such as VB Script, use it to locate and load ActiveX DLLs. DSI VB rule processing uses the ProgID you put into the configuration file.

The ProgID is very important. Unless you get in the way, VB generates a ProgID from the combination of DLL name and class name and DSI VB rule processing depends on this convention. Unless you leave it blank, the project description in the VB project properties will be used by VB to assign your ProgID. Therefore, it is important to leave the Project Description field blank.

**Example**

This example was created using the DSI rule wizard and can be found in the samples:

```sql
' ============================================================
' GoFish - DSI rule
'```
Writing Processing Rules in Visual Basic

' Arguments
'    oDSI - object to access the DSI API
'    pszParms - parameter string from the .INI file
'    ulMsg - message number from the server. See case statement below
'    ulOptions - reserved for future use
'
' Generated by the DSI Rule Wizard version 1.0
' ============================================================
Public Function GoFish(ByVal oDSI As DSICoAPI, ByVal hInstance As Long, ByVal pszParms As String, ByVal ulMsg As DSI_MSG, ByVal ulOptions As Long) As Long
ByVal hInstance As Long

On Error GoTo ErrorHandler

' TO DO: for each of the messasges you support, add logic to the case statement. For the messages you don't support, delete the entire case statement so processing falls through to the else
' TO DO: Include your rule in the docserv.ini. The syntax is
'   function        = DSICoRul->Invoke,Fish.IBass->GoFish
'
GoFish = dsiSUCCESS
Select Case ulMsg

    Case dsiMSGRUNF ' Forward (ie, inbound) logic
        oDSI.AddAttachVar hInstance, dsiOUTPUTQUEUE, "MyStatistics", "Honest!"
        Dim sRecName As String
        oDSI.AddAttachRec hInstance, dsiOUTPUTQUEUE, "Libraries", sRecName
        oDSI.AddToAttachRec hInstance, dsiOUTPUTQUEUE, sRecName, "Name", "Fish"
        oDSI.AddToAttachRec hInstance, dsiOUTPUTQUEUE, sRecName, "Date", "date"
        oDSI.AddToAttachRec hInstance, dsiOUTPUTQUEUE, sRecName, "Version", "1.0"

    Case Else ' We don't support the other messages
        GoFish = dsiMSGNOTFOUND
    End Select

Exit Function

ErrorHandler:
' This error handler will pass the error on to the error handling routine in the caller
' You should not display messages in a DSICo Rule
   GoFish = dsiRULECRASH
End Function

SAMPLES

The Internet Document Server includes several samples you can use. These include:

- DSICoTB on page 68
- DSITest on page 69
- DSIDiag on page 71
- DSIDiag.exe on page 71
- Debug.ASP on page 72
- DSICoSAM on page 73
- DSICoExV on page 74
- DSICoEx.cpp on page 75
- DSICoAdm and ADMasp on page 76
- DSI COM Objects under ASP on page 76

DSICoTB

DSICoTB—the Visual Basic Test Bed—lets you test customer rules. In addition to executing the server administration requests, you can build your own requests and attachment lists.

To use the custom attachment list, select the Roll Your Own button and then enter the request code you want.

The grid on the left can be filled with your name/value pairs.
Click Execute to send your attachment to the server and return to the main form, which displays the calls to Visual Basic and the results.

This sample includes these files:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSICoTB.frm</td>
<td>VB form</td>
</tr>
<tr>
<td>DSICoTTTr.frm</td>
<td>VB form layout</td>
</tr>
<tr>
<td>DSICoTB.frx</td>
<td>VB form layout</td>
</tr>
<tr>
<td>DSICoTB.vbp</td>
<td>VB project</td>
</tr>
<tr>
<td>DSICoTB.vbw</td>
<td>VB work space</td>
</tr>
<tr>
<td>About.frm</td>
<td>VB form</td>
</tr>
<tr>
<td>About.frx</td>
<td>VB form layout</td>
</tr>
<tr>
<td>DSICoTB.bas</td>
<td>common data</td>
</tr>
</tbody>
</table>

**DSITest**

This version includes the DSITEST program which you can use to test sending files to IDS and receiving files from IDS.

**Usage**

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>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.</td>
</tr>
<tr>
<td>WaitOnLast</td>
<td>Waits on the last message before capturing the ending time.</td>
</tr>
</tbody>
</table>
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:
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 NoAttchs 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 NoCRVs 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.

**DSIDiag**

DSIDiag consists of two samples, an application written Visual Basic (VB), DSIDiag.exe, and an Active Server Page (ASP), Debug.ASP.

**DSIDiag.exe**

DSIDiag interrogates the DSI diagnostic interface to display key information, including the current directory and the location of the queue files. You can also print the information. You do not have to have IDS running to get this information.

The content and layout of the information displayed is context-sensitive and can change with new system versions and updates. Refer to your latest documentation or read.me updates for information on how to interpret the content.
Chapter 1
Using the Internet Document Server SDK

Setup
Run DSIDiag from the same directory as your client application or web server to get accurate information.

Execution
DSIDiag displays diagnostic information as soon as you start it. You can refresh the information, print it, or copy it to the clipboard.

This sample includes these files:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIDiag.frm</td>
<td>VB form source file</td>
</tr>
<tr>
<td>DSIDiag.frx</td>
<td>VB form layout file</td>
</tr>
<tr>
<td>DSIDiag.vbp</td>
<td>VB IDE project file</td>
</tr>
<tr>
<td>DSIDiag.vbw</td>
<td>VB IDE work space file</td>
</tr>
</tbody>
</table>

Debug.ASP

This Active Server Page recovers the same information as DSIDiag using your browser. Debug.asp references an ASP ActiveX component that makes the necessary calls to the DSI library.

The content and layout of the information displayed is context-sensitive and can change with new system versions and updates. Refer to your latest documentation or read.me updates for information on how to interpret the content.

Setup
The IDS setup routine places the DLL and Debug.ASP files in their proper locations.

Execution
Select DEBUG.ASP using your browser. First the system PATH appears, followed by the debug information.
DSICoSAM

DSICoSAM is a Visual Basic application which contains much of the sample code that appears in the documentation. This makes it a good source of working code you can cut-and-paste into applications you build. In addition, you can use it as a guide by taking a working program and modifying it.

Execution

There are two list boxes to choose from before you run the test. The first, Choose Object, chooses the COM object to test, such as DSICoAPI; the second chooses the individual method to test.

To execute the test (or all the tests) select the appropriate button. The left pane shows a log of the activity, the right the output or results. If you want to retain the log or output, you can copy both panes to the clipboard by pressing their respective Copy To Clipboard buttons.

Of course, IDS must be running and configured. The IDS setup routine configures IDS for you, which includes the following:

```plaintext
< ReqType:INI >
  function = DSICoRul->Init
</ ReqType:ECH >
< ReqType:ECH >
  function = atcw32->ATCLoadAttachment
  function = DSICoRul->Invoke,Docucorp_IDS_SAMSupp.CSAMSupp->Echo
```
function = atcw32->ATCUnloadAttachment

This sample includes these files:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>csamapi.cls</td>
<td>Tests class file</td>
</tr>
<tr>
<td>csamsupp.cls</td>
<td>ECH (Echo) rule class file</td>
</tr>
<tr>
<td>csamobj.cls</td>
<td>Test object used in some tests. Has no code.</td>
</tr>
<tr>
<td>DSICoSAM.frm</td>
<td>DSICoSAM form source code</td>
</tr>
<tr>
<td>DSICoSAM.frx</td>
<td>DSICoSAM layout</td>
</tr>
<tr>
<td>Dsicosam.vbp</td>
<td>DSICoSAM VB project</td>
</tr>
<tr>
<td>DSCOSAM.VBW</td>
<td>DSICoSAM VB work space</td>
</tr>
<tr>
<td>samsupp.vbp</td>
<td>ECH (Echo) rule VB project</td>
</tr>
<tr>
<td>SAMSUPP.VBW</td>
<td>CH (Echo) rule VB work space</td>
</tr>
<tr>
<td>samobj.vbp</td>
<td>Test object used in some tests; VB project</td>
</tr>
<tr>
<td>SAMTOBJ.VBW</td>
<td>Test object used in some tests; VB work space</td>
</tr>
</tbody>
</table>

**DSICoExV**

DSICoExV is the Visual Basic version of DSIEEx.c, duplicating the functionality of DSIEEx and more-or-less duplicating the logic. Instead of calling the DSI API directly, it calls the equivalent Visual Basic COM objects.

**NOTE:** Although there is a simpler way under Visual Basic to accomplish the functionality using, for instance, InitSession instead of Init, the direct calls were used to make easier the comparison with DSIEEx.c.

The application, after initializing COM, establishes a connection with IDS and places the selected IDS Server administration command (such as SSS) in the queue. Each Visual Basic call is logged in the left pane and the output in the right pane.

**Execution**

Run DSICoExV.exe. Select the server administration command to run. SSS, the server statistics, is set up as the default.
This sample includes these files:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSICoExV.frm</td>
<td>VB form</td>
</tr>
<tr>
<td>DSICoExV.frx</td>
<td>VB form layout</td>
</tr>
<tr>
<td>DSICoExV.vbp</td>
<td>VB project</td>
</tr>
<tr>
<td>DSICoExV.vbw</td>
<td>VB work space</td>
</tr>
</tbody>
</table>

**DSICoEx.cpp**

DSICoEx is the Visual Basic version of DSIEx.c. DSICoEx duplicates the functionality of DSIEx and, essentially, duplicates its logic.

Instead of calling the DSI API directly, DSICoEx calls the equivalent Visual Basic COM objects. Although there is a simpler way under Visual Basic to accomplish this functionality—for instance by using InitSession instead of Init—the direct calls were used to make easier the comparison with DSIEx.c.

The application, after initializing COM, establishes a connection with IDS and places IDSIDS administration command SS3 in the queue. The response attachment is written in its entirety to stdout.

**Setup:**

Visual Basic must be installed on the system. To use, the VC project file %DSICO% must point to the head of the DSICo directory tree. To compile, load DSICoEx.dsp into VC and compile.

**Execution:**

DSICoEx.exe is included in the installation. DSICoEx is a console application and should be run from the command line. It outputs to sysout. DSICo.dll should be registered as part of the installation.

This sample includes these files:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSICoEx.cpp</td>
<td>source files</td>
</tr>
</tbody>
</table>
Chapter 1
Using the Internet Document Server SDK

Visual Basic files used:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSICo.hpp</td>
<td>Visual Basic specific macros</td>
</tr>
<tr>
<td>DSICo.tlb</td>
<td>Visual Basic type library created by the Visual Basic MIDL</td>
</tr>
</tbody>
</table>

**DSICoAdm and ADMAsp**

DSICoADM and ADMAsp are versions of the same function, which interrogates IDS Server statistics.

- **DSICoADM** is a Visual Basic application which interrogates IDS statistics and presents them in a Visual Basic grid.
- **ADMAsp** is an Active Server Page which does the same thing through an ActiveX component and presents IDS statistics on the browser.

These files are included in this sample:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSICoADM.frm</td>
<td>VB form</td>
</tr>
<tr>
<td>DSICoADM.frx</td>
<td>VB form layout</td>
</tr>
<tr>
<td>DSICoADM.vbp</td>
<td>VB project</td>
</tr>
<tr>
<td>DSICoADM.vbw</td>
<td>VB work space</td>
</tr>
<tr>
<td>ADMAsp.vbp</td>
<td>VB project</td>
</tr>
<tr>
<td>ADMAsp.vbw</td>
<td>VB work space</td>
</tr>
<tr>
<td>SSS.cls</td>
<td>ASP ActiveX component class</td>
</tr>
</tbody>
</table>

**DSI COM Objects under ASP**

This sample shows you how to use DSI COM objects and Visual Basic to create ActiveX DLLs that run under the Microsoft Internet Information Server and Active Server Page (ASP) to interface with Oracle Insurance's Internet Document Server.

**Setup**

Load the project into the VB IDE and select the Make AdmASP.dll option. You may have to shut down the IIS and IIS administration to unlock the DLL.

Move the ADMIN.ASP and DOCC.BMP files into the wwwroot directory. Once you have compiled the project, you do not have to relocate or register the DLL.
Execution

Point your web browser to Admin.asp. The server statistics appear. Click Server Statistics to refresh the display with new values.

This sample includes these files:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdmASP.vbp</td>
<td>Project</td>
</tr>
<tr>
<td>AdmASP.vpw</td>
<td>Work space</td>
</tr>
<tr>
<td>SSS.cls</td>
<td>Class file</td>
</tr>
<tr>
<td>Admin.asp</td>
<td>ASP script file</td>
</tr>
<tr>
<td>docc.bmp</td>
<td>Docucorp logo</td>
</tr>
</tbody>
</table>
REFERENCING ATTACHMENT VARIABLES

This feature lets you reference the attachment variable from a configuration 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 2.0, 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 a configuration file option:

```
< Group >
  Option = ~GetAttach VARNAME,INPUT
</Group>
```

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

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

The VARNAME is the name of the variable. INPUT or OUTPUT specify 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 the XML can be modified to read:

```
<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.

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.
Chapter 2

DSI C APIs

Use this chapter as a reference to the DSI C API functions you can use to create applications to interface with Oracle Insurance’s Internet Document Server.

This information will help you build either a proprietary client interface or a custom set of rules which will interact with the Internet Document Server.

The APIs documented on the following pages provide a large number of services, including...

• Interprocess communication
• Persistent variables
• Accessible across function calls
• Error reporting

Several general purpose utility functions are also available.

**NOTE:** The DSI API includes multiple interfaces (APIs). This lets you use the language you choose to build custom rules and applications. You will also find sample clients written in each language, which you can use as a reference as you build your own solution.
Here is a list of DSI C APIs, grouped by functional area. Following this list is a discussion of each function, listed in alphabetical order.

**C API FUNCTIONS**

**Client functions**
Use these functions for writing a client program:
- `DSIAddToQueue` on page 89
- `DSICopyQRecord` on page 118
- `DSIFindInQueue` on page 126
- `DSIGetFirstFromQueue` on page 127
- `DSIGetSOAPMessage` on page 128
- `DSIGetSOAPMessageSize` on page 129
- `DSIGetQError` on page 130
- `DSIGetQField` on page 131
- `DSIGetQFieldLength` on page 133
- `DSISetQField` on page 156
- `DSIGetQueueRec` on page 134
- `DSIIInit` on page 136
- `DSIIInitInstance` on page 137
- `DSIIInitQueue` on page 138
- `DSIParseAttachment` on page 144
- `DSIStoreAttachment` on page 158
- `DSITerm` on page 159
- `DSITermInstance` on page 160
- `DSITermQueue` on page 161
- `LDAPGetErrorCode` on page 162
- `LDAPGetErrorMessage` on page 163
- `LDAPInit` on page 164
- `LDAPSearchDirectory` on page 169
- `LDAPTerm` on page 170

**Server functions**
Use these functions for writing rules on the server:
- `DSIErrorMessage` on page 124
- `DSIErrorMsg` on page 125

**Common functions**
You can use these functions for both a client or a server:
• DSIAddAttachRec on page 83
• DSIAddAttachVar on page 84
• DSIAddToAttachRec on page 86
• DSIAddToQueue on page 89
• DSIAattachCursorFirst on page 90
• DSIAattachCursorLast on page 94
• DSIAattachCursorName on page 98
• DSIAattachCursorNext on page 100
• DSIAattachCursorPrev on page 104
• DSIAattachCursorValue on page 108
• DSICacheFile on page 115
• DSICloseAttachCursor on page 116
• DSICopyAttachVars on page 117
• DSICreateValue on page 119
• DSIDeleteAttachVar on page 120
• DSIDestroyValue on page 121
• DSIGetUniqueString on page 135
• DSILocateAttachVar on page 139
• DSILocateValue on page 141
• DSIOpenAttachCursor on page 142
• DSIQqueryValueSize on page 146
• DSIRecieveFile on page 147
• DSIRecieveFileAsBuffer on page 148
• DSIRecieveFileAsBufferSize on page 150
• DSIRowset2XML on page 152
• DSIRowset2XMLSize on page 153
• DSISendBuffer on page 154
• DSISendFile on page 155
• DSISetQField on page 156
• DSISoreAttachment on page 158
• DSITerm on page 159
• DSITermInstance on page 160
- DSITermQueue on page 161
**DSIAddAttachRec**

Use this function to create a stem variable in the attachment list. This function returns the new record name with its sequence number.

**Syntax**

```c
long DSIAddAttachRec(DSIHANDLE hInstance, long iQueue, char* szRecName, char* szRecID, size_t cbRecID);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue attachment to which record should be added</td>
</tr>
<tr>
<td>szRecName</td>
<td>name of stem variable to be added</td>
</tr>
<tr>
<td>szRecID</td>
<td>buffer in which to store record name with sequence number. The calling function should pass this to DSIAddToAttachRec</td>
</tr>
<tr>
<td>cbRecID</td>
<td>size of szRecID parameter</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_MEMORY</td>
<td>out of memory</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

**Errors**

- Invalid parameter
- Memory error

**See also** [DSIAddToAttachRec on page 86](#)

**Example**

```c
char szRecName [ DSI_MAXNAMESIZE ];
DSIAddAttachRec(    hInstance, 
    DSI_OUTPUTQUEUE, 
    "Employee", 
    szRecName, 
    sizeof( szRecName ) );
DSIAddToAttachRec( ..., szRecName, ... );
```
DSIAddAttachVar

Use this function to add an attachment variable. This function will overwrite the variable, if one exists, with the new value.

After you use this function, you must next call DSIStoreAttachment.

Syntax

```c
long DSIAddAttachVar(DSIHANDLE hInstance, long iQueue, char* szName, char* szValue);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue attachment to which variable should be added</td>
</tr>
<tr>
<td>szName</td>
<td>name of the variable to be added</td>
</tr>
<tr>
<td>szValue</td>
<td>data to be associated with attachment variable</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_MEMORY</td>
<td>out of memory</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

Errors

Invalid parameter

Uninitialized attachment list

See also

DSILocateAttachVar on page 139
DSIDeleteAttachVar on page 120
DSIStoreAttachment on page 158

Example

```c
DSIAddAttachVar( hInstance, DSI_OUTPUTQUEUE, "RESULTS", "SUCCESS" );
```
**DSIAddAttachVarEx**

Use this function to add an attachment variable. This function will overwrite the variable, if one exists, with the new value.

After you use this function, you must next call DSIStoreAttachment.

**Syntax**

```c
long DSIAddAttachVarEx(DSIHANDLE hdsi, long iQueue, char* szName, char* szValue, long IEncoding);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>encoding</td>
<td>DSIENCODING_SINGLE_BYTE or DSIENCODING_UTF_8. DSIENCODING_SINGLE_BYTE uses codepage 1252 encoding, similar to ASCII but is compatible with Documaker RP handles Euro characters and others. DSIENCODING_UTF_8 translates Unicode into a format compatible with null-terminated C language strings.</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue attachment to which variable should be added</td>
</tr>
<tr>
<td>szName</td>
<td>name of the variable to be added</td>
</tr>
<tr>
<td>szValue</td>
<td>data to be associated with attachment variable</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_MEMORY</td>
<td>out of memory</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

**Errors**

Invalid parameter

Uninitialized attachment list

**See also**

DSILocateAttachVar on page 139

DSIDeleteAttachVar on page 120

DSIStoreAttachment on page 158

**Example**

```c
DSIAddAttachVarEx(hInstance, DSI_OUTPUTQUEUE, "RESULTS", "SUCCESS", DSIENCODING_UTF_8);
```
DSIAddToAttachRec

Use this function to append a value to a stem variable.

**Syntax**

```c
long DSIAddToAttachRec (DSIHANDLE hInstance, long iQueue, char* szRecName, char* szVarName, char* szValue);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue attachment to which value should be added</td>
</tr>
<tr>
<td>szRecName</td>
<td>record to which variable should be added, generally returned by the DSIAddAttachRec function</td>
</tr>
<tr>
<td>szVarName</td>
<td>name of field within record</td>
</tr>
<tr>
<td>szValue</td>
<td>data to be associated with variable</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_MEMORY</td>
<td>out of memory</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

**Errors**

- Invalid parameter
- Memory error

**See also**

DSIAddAttachRec on page 83

**Example**

```c
char szRecName [ DSI_MAXNAMESIZE ];

DSIAddAttachRec(    hInstance,    DSI_OUTPUTQUEUE,    "Employee",    szRecName,    sizeof( szRecName ) );

DSIAddToAttachRec(    hInstance,    DSI_OUTPUTQUEUE,    szRecName,    "Name",    "H. R. Pufnstuf" );

DSIAddToAttachRec(    hInstance,    DSI_OUTPUTQUEUE,    szRecName,    "DependentName",    "Jimmy" );
```
DSIAddToAttachRecEx

Use this function to append a value to a stem variable.

Syntax

```c
long DSIAddToAttachRecEx (DSIHANDLE hdsi, long iQueue, char* szRecName, char* szFieldName, char* szValue, long IEncoding);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>encoding</td>
<td>DSIENCODING_SINGLE_BYTE or DSIENCODING_UTF_8. DSIENCODING_SINGLE_BYTE uses codepage 1252 encoding, similar to ASCII but is compatible with Documaker RP handles Euro characters and others. DSIENCODING_UTF_8 translates Unicode into a format compatible with null-terminated C language strings.</td>
</tr>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue attachment to which value should be added</td>
</tr>
<tr>
<td>szRecName</td>
<td>record to which variable should be added, generally returned by the DSIAddAttachRec function</td>
</tr>
<tr>
<td>szVarName</td>
<td>name of field within record</td>
</tr>
<tr>
<td>szValue</td>
<td>data to be associated with variable</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_MEMORY</td>
<td>out of memory</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

Errors

Invalid parameter
Memory error

See also

DSIAddAttachRec on page 83

Example

```c
char szRecName [ DSI_MAXNAMESIZE ];
DSIAddAttachRec(    hInstance,
                    DSI_OUTPUTQUEUE,
                    "Employee",
                    szRecName,
                    sizeof( szRecName ) );
DSIAddToAttachRecEx(  hInstance,
                    DSI_OUTPUTQUEUE,
                    szRecName,
                    "Name",
                    "H. R. Pufnstuf",
                    DSIENCODING_UTF_8);  
```
DSIAddToAttachRecEx( hInstance,
    DSI_OUTPUTQUEUE,
    szRecName,
    "DependentName",
    "Jimmy"
    DSIENCODING_UTF_8);
DSIAddToQueue

Use this function to add a record to a queue.

Syntax

```c
long DSIAddToQueue(DSIHANDLE hInstance, long iQueue);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>Queue on which to post</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INV Parm</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_MEMORY</td>
<td>out of memory</td>
</tr>
<tr>
<td>DSIERR_NOT FOUND</td>
<td>no more elements in the list</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
<tr>
<td>DSIERR Q ERR</td>
<td>uninitialized queue</td>
</tr>
<tr>
<td>DSIERR IO ERR</td>
<td>end of file</td>
</tr>
</tbody>
</table>

Errors

Invalid parameter

Lower level API failure-call DSIGetQError

Example

```c
DSIAddToQueue( hInstance, DSI_OUTPUTQUEUE );
```
DSIAttachCursorFirst

Use this function to retrieve the first element from the attachment list and get the cursor.

**Syntax**

```c
long DSIAttachCursorFirst(DSIHANDLE hCursor, char* pszName, size_t cbName, char* pszValue, size_t cbValue);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCursor</td>
<td>handle to cursor initialized by prior call to DSIOpenAttachCursor</td>
</tr>
<tr>
<td>pszName</td>
<td>buffer in which to retrieve the name of the first element of the attachment</td>
</tr>
<tr>
<td>cbName</td>
<td>size of buffer in pszName parameter</td>
</tr>
<tr>
<td>pszValue</td>
<td>buffer in which to retrieve the value of the first element of the attachment</td>
</tr>
<tr>
<td>cbValue</td>
<td>size of buffer in pszValue parameter</td>
</tr>
</tbody>
</table>

**NOTE:** The parameters pszName and pszValue will be zero-filled to the length specified in cbName and cbValue.

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVParm</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_NOTFOUND</td>
<td>empty list</td>
</tr>
</tbody>
</table>

**Errors**

- Invalid parameter
- Memory allocation failure
- Uninitialized queue
- Empty attachment list

**See also**

- DSIAttachCursorNext on page 100
- DSIAttachCursorLast on page 94
- DSIAttachCursorPrev on page 104
- DSICloseAttachCursor on page 116
- DSIParseAttachment on page 144

**Example**

```c
DSIHANDLE hApp;
DSIHANDLE hInstance;
DSIHANDLE hCursor;
char szName [ DSI_MAXNAMESIZE ];
char szValue [ DSI_MAXVALUESIZE ];
```
hApp = DSIInit();
hInstance = DSIInitInstance( hApp );
hCursor = DSIOpenAttachCursor( hInstance,
                             DSI_INPUTQUEUE );

if ( DSIAttachCursorFirst( hCursor,
                           szName,
                           sizeof(szName),
                           szValue,
                           sizeof(szValue)) == DSIERR_SUCCESS )
{
    printf( "The first element is: %s = %s", szName, szValue );
}
.
.
.
.
**DSIAttachCursorFirstEx**

Use this function to retrieve the first element from the attachment list and get the cursor.

**Syntax**

```
long DSIAttachCursorFirstEx(DSIHANDLE hCursor, char* pszName, size_t cbName, char* pszValue, size_t cbValue, long IEncoding);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>encoding</td>
<td>DSIENCODING_SINGLE_BYTE or DSIENCODING_UTF_8. DSIENCODING_SINGLE_BYTE uses codepage 1252 encoding, similar to ASCII but is compatible with Documaker RP handles Euro characters and others. DSIENCODING_UTF_8 translates Unicode into a format compatible with null-terminated C language strings.</td>
</tr>
<tr>
<td>hCursor</td>
<td>handle to cursor initialized by prior call to DSIOpenAttachCursor</td>
</tr>
<tr>
<td>pszName</td>
<td>buffer in which to retrieve the name of the first element of the attachment</td>
</tr>
<tr>
<td>cbName</td>
<td>size of buffer in pszName parameter</td>
</tr>
<tr>
<td>pszValue</td>
<td>buffer in which to retrieve the value of the first element of the attachment</td>
</tr>
<tr>
<td>cbValue</td>
<td>size of buffer in pszValue parameter</td>
</tr>
</tbody>
</table>

**NOTE:** The parameters pszName and pszValue will be zero-filled to the length specified in cbName and cbValue.

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_NOTFOUND</td>
<td>empty list</td>
</tr>
</tbody>
</table>

**Errors**

Invalid parameter
Memory allocation failure
Uninitialized queue
Empty attachment list

**See also**

- DSIAttachCursorNext on page 100
- DSIAttachCursorLast on page 94
- DSIAttachCursorPrev on page 104
- DSICloseAttachCursor on page 116
- DSIParseAttachment on page 144
Example

DSIHANDLE hApp;
DSIHANDLE hInstance;
DSIHANDLE hCursor;
char szName [DSI_MAXNAMESIZE];
char szValue [DSI_MAXVALUESIZE];

hApp = DSIInit();
hInstance = DSIInitInstance( hApp );
hCursor = DSIOpenAttachCursor( hInstance,
                               DSI_INPUTQUEUE );

if ( DSIAttachCursorFirstEx( hCursor,
                           szName,
                           sizeof(szName),
                           szValue,
                           sizeof(szValue)
                           DSIENCODING_UTF_8) == DSIERR_SUCCESS )
{
    printf( "The first element is: %s = %s", szName, szValue );
}
.
.
.
DSIAttachCursorLast

Use this function to retrieve the last element from the attachment list.

Syntax

```c
long DSIAttachCursorLast(DSIHANDLE hCursor, char* pszName, size_t cbName, char* pszValue, size_t cbValue);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCursor</td>
<td>handle to attachment cursor initialized by a prior call to DSIOpenAttachCursor</td>
</tr>
<tr>
<td>pszName</td>
<td>buffer in which to retrieve the name of the first element of the attachment</td>
</tr>
<tr>
<td>cbName</td>
<td>size of buffer in pszName parameter</td>
</tr>
<tr>
<td>pszValue</td>
<td>buffer in which to retrieve the value of the first element of the attachment</td>
</tr>
<tr>
<td>cbValue</td>
<td>size of buffer in pszValue parameter</td>
</tr>
</tbody>
</table>

**NOTE:** The parameters pszName and pszValue will be zero-filled to the length specified in cbName and cbValue.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_NOTFOUND</td>
<td>empty list</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

Errors

Invalid parameter

No more list elements

See also

- DSIOpenAttachCursor on page 142
- DSICloseAttachCursor on page 116
- DSIAttachCursorFirst on page 90
- DSIAttachCursorNext on page 100
- DSIAttachCursorPrev on page 104
- DSIParseAttachment on page 144

Example

```c
DSIHANDLE hApp;
DSIHANDLE hInstance;
DSIHANDLE hCursor;
char szName [ DSI_MAXNAMESIZE ];
char szValue [ DSI_MAXVALUESIZE ];
```
hApp = DSIInit();
hInstance = DSIInitInstance( hApp );
hCursor = DSIOpenAttachCursor( hInstance, DSI_INPUTQUEUE );
if ( hCursor )
{
    if ( DSIAttachCursorLast( hCursor, szName, sizeof(szName), szValue, sizeof(szValue) ) == DSIERR_SUCCESS )
    {
        printf( "The last element is %s=%s", szName, szValue );
        while ( DSIAttachCursorPrev( hCursor, szName, sizeof(szName), szValue, sizeof(szValue) ) == DSIERR_SUCCESS )
        {
            printf( "The previous element is %s=%s", szName, szValue );
        }
    }
}
.
.
.

DSIAttachCursorLast
DSIAttachCursorLastEx

Use this function to retrieve the last element from the attachment list.

Syntax

```c
long DSIAttachCursorLastEx(DSIHANDLE hCursor, char* pszName, size_t cbName, char* pszValue, size_t cbValue, long IEncoding);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>encoding</td>
<td>DSIENCODING_SINGLE_BYTE or DSIENCODING_UTF_8. DSIENCODING_SINGLE_BYTE uses codepage 1252 encoding, similar to ASCII but is compatible with Documaker RP handles Euro characters and others. DSIENCODING_UTF_8 translates Unicode into a format compatible with null-terminated C language strings.</td>
</tr>
<tr>
<td>hCursor</td>
<td>handle to attachment cursor initialized by a prior call to DSIOpenAttachCursor</td>
</tr>
<tr>
<td>pszName</td>
<td>buffer in which to retrieve the name of the first element of the attachment</td>
</tr>
<tr>
<td>cbName</td>
<td>size of buffer in pszName parameter</td>
</tr>
<tr>
<td>pszValue</td>
<td>buffer in which to retrieve the value of the first element of the attachment</td>
</tr>
<tr>
<td>cbValue</td>
<td>size of buffer in pszValue parameter</td>
</tr>
</tbody>
</table>

**NOTE:** The parameters pszName and pszValue will be zero-filled to the length specified in cbName and cbValue.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVAPRM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_NOTFOUND</td>
<td>empty list</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

Errors

- Invalid parameter
- No more list elements

See also

- DSIOpenAttachCursor on page 142
- DSICloseAttachCursor on page 116
- DSIAttachCursorFirst on page 90
- DSIAttachCursorNext on page 100
- DSIAttachCursorPrev on page 104
- DSIParseAttachment on page 144
Example

```c
DSIHANDLE hApp;
DSIHANDLE hInstance;
DSIHANDLE hCursor;
char szName [ DSI_MAXNAMESIZE ];
char szValue [ DSI_MAXVALUESIZE ];

hApp = DSInit();
hInstance = DSInitInstance( hApp );
hCursor = DSIOpenAttachCursor( hInstance,
                             DSI_INPUTQUEUE );

if( hCursor )
{
    if ( DSIAttachCursorLastEx( hCursor,
                  szName,
                  sizeof(szName),
                  szValue,
                  sizeof(szValue)
                  DSIENCODING_UTF_8) == DSIERR_SUCCESS )
    {
        printf( "The last element is %s=%s", szName, szValue );
        while( DSIAttachCursorPrev( hCursor,
                                szName,
                                sizeof(szName),
                                szValue,
                                sizeof(szValue)
                                DSIENCODING_UTF_8) == DSIERR_SUCCESS )
        {
            printf( "The previous element is %s=%s", szName, szValue );
        }
    }
}
```
DSIAttachCursorName

Use this function to retrieve the name of the current element from the attachment list.

**Syntax**

```c
long DSIAttachCursorName(DSIHANDLE hCursor, char* pszName, size_t cbName);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCursor</td>
<td>handle to attachment cursor initialized by a prior call to DSIOpenAttachCursor and positioned by calls to DSAAttchCursor&lt;sup&gt;*&lt;/sup&gt; call</td>
</tr>
<tr>
<td>pszName</td>
<td>buffer in which to retrieve the name of the element of the attachment</td>
</tr>
<tr>
<td>cbName</td>
<td>size of buffer in pszName parameter</td>
</tr>
</tbody>
</table>

**NOTE:** The parameter pszName will be zero-filled to the length specified in cbName.

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_NOTFOUND</td>
<td>no such element in the list</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

**Errors**

Invalid parameter

The position of the cursor is invalid

**See also**

- DSIOpenAttachCursor on page 142
- DSICloseAttachCursor on page 116
- DSIAttachCursorFirst on page 90
- DSIAttachCursorNext on page 100
- DSIAttachCursorLast on page 94
- DSIAttachCursorPrev on page 104
- DSIAttachCursorValue on page 108
- DSIParseAttachment on page 144

**Example**

```c
DSIHANDLE   hApp;
DSIHANDLE   hInstance;
DSIHANDLE   hCursor;
char        szName[DSI_MAXNAMESIZE];
char        szValue[DSI_MAXVALUESIZE];
```
hApp = DSIInit();
hInstance = DSIInitInstance( hApp );
hCursor = DSIOpenAttachCursor( hInstance,
  DSI_INPUTQUEUE );
if( hCursor )
{
  if ( DSIAttachCursorLast( hCursor,
      NULL,
      0,
      NULL,
      0 ) == DSIERR_SUCCESS )
  {
    DSIAttachCursorName( hCursor, szName, sizeof(szName) );
    DSIAttachCursorValue( hCursor, szValue, sizeof(szValue) );
    printf( "The last element is %s=%s", szName, szValue );
  }
}

.
DSIAttachCursorNext

Use this function to retrieve the next element from the attachment list.

Syntax

```c
long DSIAttachCursorNext(DSIHANDLE hCursor, char* pszName, size_t cbName, char* pszValue, size_t cbValue);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCursor</td>
<td>handle to attachment cursor initialized by a prior call to DSIOpenAttachCursor</td>
</tr>
<tr>
<td>pszName</td>
<td>buffer in which to retrieve the name of the first element of the attachment</td>
</tr>
<tr>
<td>cbName</td>
<td>size of buffer in pszName parameter</td>
</tr>
<tr>
<td>pszValue</td>
<td>buffer in which to retrieve the value of the first element of the attachment</td>
</tr>
<tr>
<td>cbValue</td>
<td>size of buffer in pszValue parameter</td>
</tr>
</tbody>
</table>

NOTE: The parameters pszName and pszValue will be zero-filled to the length specified in cbName and cbValue.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_NOTFOUND</td>
<td>no more elements in the list</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

Errors

Invalid parameter

No more list elements

See also

DSIOpenAttachCursor on page 142
DSIAttachCursorPrev on page 104
DSIParseAttachment on page 144

Example

```c
DSIHANDLE hApp;
DSIHANDLE hInstance;
DSIHANDLE hCursor;
char szName [ DSI_MAXNAMESIZE ];
char szValue [ DSI_MAXVALUESIZE ];
```
hApp = DSIInit();
hInstance = DSIInitInstance( hApp );
hCursor = DSIOpenAttachCursor(  hInstance,
DSI_INPUTQUEUE,
szName,
sizeof(szName),
szValue,
sizeof(szValue));

if( hCursor )
{
  printf( "The first element is %s", szValue );
  while( DSIAttachCursorNext( hCursor,
szName,
sizeof(szName)
szValue,
sizeof(szValue) )
      == DSIERR_SUCCESS )
  {
    printf( "The next element is %s=%s", szName,szValue );
  }
}
  .
  .
  .
DSIAttachCursorNextEx

Use this function to retrieve the next element from the attachment list.

**Syntax**

```c
long DSIAttachCursorNextEx(DSIHANDLE hCursor, char* pszName, size_t cbName, char* pszValue, size_t cbValue, long IEncoding);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>encoding</td>
<td>DSIENCODING_SINGLE_BYTE or DSIENCODING_UTF_8. DSIENCODING_SINGLE_BYTE uses codepage 1252 encoding, similar to ASCII but is compatible with Documaker RP handles Euro characters and others. DSIENCODING_UTF_8 translates Unicode into a format compatible with null-terminated C language strings.</td>
</tr>
<tr>
<td>hCursor</td>
<td>handle to attachment cursor initialized by a prior call to DSIOpenAttachCursor</td>
</tr>
<tr>
<td>pszName</td>
<td>buffer in which to retrieve the name of the first element of the attachment</td>
</tr>
<tr>
<td>cbName</td>
<td>size of buffer in pszName parameter</td>
</tr>
<tr>
<td>pszValue</td>
<td>buffer in which to retrieve the value of the first element of the attachment</td>
</tr>
<tr>
<td>cbValue</td>
<td>size of buffer in pszValue parameter</td>
</tr>
</tbody>
</table>

**NOTE:** The parameters pszName and pszValue will be zero-filled to the length specified in cbName and cbValue.

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_NOTFOUND</td>
<td>no more elements in the list</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

**Errors**

- Invalid parameter
- No more list elements

**See also**

- DSIOpenAttachCursor on page 142
- DSIAttachCursorPrev on page 104
- DSIParseAttachment on page 144
Example

```c
DSIHANDLE   hApp;
DSIHANDLE   hInstance;
DSIHANDLE   hCursor;
char        szName [ DSI_MAXNAMESIZE ];
char        szValue [ DSI_MAXVALUESIZE ];
hApp = DSIInit();
hInstance = DSIInitInstance( hApp );
hCursor = DSIOpenAttachCursor( hInstance,
                                DSI_INPUTQUEUE);
if( hCursor )
{
    if ( DSIAttachCursorFirstEx( hCursor,
                                szName,
                                sizeof(szName),
                                szValue,
                                sizeof(szValue),
                                DSIENCODING_UTF_8)==DSIERR_SUCCESS)
    {
        printf( "The first element is %s", szValue );
        while( DSIAttachCursorNextEx( hCursor,
                                      szName,
                                      sizeof(szName)
                                      szValue,
                                      sizeof(szValue)
                                      DSIENCODING_UTF_8)
                        == DSIERR_SUCCESS )
        {
            printf( "The next element is %s=%s", szName,szValue );
        }
    }
}
```

.
DSIAttachCursorPrev

Use this function to retrieve the previous element from the attachment list.

**Syntax**

```c
long DSIAttachCursorPrev(DSIHANDLE hCursor, char* pszName, size_t cbName, char* pszValue, size_t cbValue);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCursor</td>
<td>handle to attachment cursor initialized by a prior call to DSIOpenAttachCursor</td>
</tr>
<tr>
<td>pszName</td>
<td>buffer in which to retrieve the name of the first element of the attachment</td>
</tr>
<tr>
<td>cbName</td>
<td>size of buffer in pszName parameter</td>
</tr>
<tr>
<td>pszValue</td>
<td>buffer in which to retrieve the value of the first element of the attachment</td>
</tr>
<tr>
<td>cbValue</td>
<td>size of buffer in pszValue parameter</td>
</tr>
</tbody>
</table>

**NOTE:** The parameters pszName and pszValue will be zero-filled to the length specified in cbName and cbValue.

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_NOTFOUND</td>
<td>no more elements in the list</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

**Errors**

Invalid parameter

No more list elements

**See also**

- DSIOpenAttachCursor on page 142
- DSICloseAttachCursor on page 116
- DSIAttachCursorFirst on page 90
- DSIAttachCursorNext on page 100
- DSIAttachCursorLast on page 94
- DSIParseAttachment on page 144
Example

```c
DSIHANDLE   hApp;
DSIHANDLE   hInstance;
DSIHANDLE   hCursor;
char        szName [ DSI_MAXNAMESIZE ];
char        szValue [ DSI_MAXVALUESIZE ];

hApp = DSIInit();
hInstance = DSIInitInstance( hApp );
hCursor = DSIOpenAttachCursor( hInstance,
                                DSI_INPUTQUEUE );
if( hCursor )
{
  if( DSIAttachCursorLast( hCursor,
                            szName,
                            sizeof(szName),
                            szValue,
                            sizeof(szValue) ) == DSIERR_SUCCESS )
  {
    printf( "The last element is %s=%s", szName, szValue );
    while( DSIAttachCursorPrev( hCursor,
                                szName,
                                sizeof(szName),
                                szValue,
                                sizeof(szValue) ) == DSIERR_SUCCESS )
      {
        printf( "The previous element is %s=%s", szName, szValue );
      }
  }
}
```
DSIAttachCursorPrevEx

Use this function to retrieve the previous element from the attachment list.

Syntax

```c
long DSIAttachCursorPrevEx(DSIHANDLE hCursor, char* pszName, size_t cbName, char* pszValue, size_t cbValue, long IEncoding);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>encoding</td>
<td>DSIENCODING_SINGLE_BYTE or DSIENCODING_UTF_8. DSIENCODING_SINGLE_BYTE uses codepage 1252 encoding, similar to ASCII but is compatible with Documaker RP handles Euro characters and others. DSIENCODING_UTF_8 translates Unicode into a format compatible with null-terminated C language strings.</td>
</tr>
<tr>
<td>hCursor</td>
<td>handle to attachment cursor initialized by a prior call to DSIOpenAttachCursor</td>
</tr>
<tr>
<td>pszName</td>
<td>buffer in which to retrieve the name of the first element of the attachment</td>
</tr>
<tr>
<td>cbName</td>
<td>size of buffer in pszName parameter</td>
</tr>
<tr>
<td>pszValue</td>
<td>buffer in which to retrieve the value of the first element of the attachment</td>
</tr>
<tr>
<td>cbValue</td>
<td>size of buffer in pszValue parameter</td>
</tr>
</tbody>
</table>

**NOTE:** The parameters pszName and pszValue will be zero-filled to the length specified in cbName and cbValue.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_NOTFOUND</td>
<td>no more elements in the list</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

Errors

- Invalid parameter
- No more list elements

See also

- DSIOpenAttachCursor on page 142
- DSICloseAttachCursor on page 116
- DSIAttachCursorFirst on page 90
- DSIAttachCursorNext on page 100
- DSIAttachCursorLast on page 94
Example

```c
DSIHANDLE   hApp;
DSIHANDLE   hInstance;
DSIHANDLE   hCursor;
char        szName [ DSI_MAXNAMESIZE ];
char        szValue [ DSI_MAXVALUESIZE ];
hApp = DSIInit();
hInstance = DSIInitInstance( hApp );
hCursor = DSIOpenAttachCursor( hInstance,
                            DSI_INPUTQUEUE );
if( hCursor )
{
    if ( DSIAttachCursorLastEx( hCursor,
                                szName,
                                sizeof(szName),
                                szValue,
                                sizeof(szValue),
                                DSIENCODING_UTF_8) == DSIERR_SUCCESS )
    {
        printf( "The last element is %s=%s", szName,szValue );
        while( DSIAttachCursorPrev( hCursor,
                                   szName,
                                   sizeof(szName),
                                   szValue,
                                   sizeof(szValue))
                == DSIERR_SUCCESS )
        {
            printf( "The previous element is %s=%s", szName,szValue );
        }
    }
}
```

..
DSIAttachCursorValue

Use this function to retrieve the value of the current element from the attachment list.

Syntax

```c
long DSIAttachCursorValue(DSIHANDLE hCursor, char* pszValue, size_t cbValue);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCursor</td>
<td>handle to attachment cursor initialized by a prior call to DSIOpenAttachCursor and positioned by calls to the DSIAttachCursorFirst, Next, Prev, Last calls.</td>
</tr>
<tr>
<td>pszValue</td>
<td>buffer in which to retrieve the value of the element of the attachment</td>
</tr>
<tr>
<td>cbValue</td>
<td>size of buffer in pszValue parameter</td>
</tr>
</tbody>
</table>

**NOTE:** The parameter pszValue will be zero-filled to the length specified in cbValue.

Return values

<table>
<thead>
<tr>
<th>Description</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_NOTFOUND</td>
<td>the position of the cursor is invalid</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

Errors

Invalid parameter

The position of the cursor is invalid

See also

- DSIOpenAttachCursor on page 142
- DSICloseAttachCursor on page 116
- DSIAttachCursorFirst on page 90
- DSIAttachCursorNext on page 100
- DSIAttachCursorLast on page 94
- DSIAttachCursorPrev on page 104
- DSIAttachCursorValue on page 108
- DSIParseAttachment on page 144

Example

```c
DSIHANDLE hApp;
DSIHANDLE hInstance;
DSIHANDLE hCursor;
char szName [ DSI_MAXNAMESIZE ];
char szValue [ DSI_MAXVALUESIZE ];
```
hApp = DSIInit();
getInstance = DSIInitInstance( hApp );
hCursor = DSIOpenAttachCursor( getInstance,
    DSI_INPUTQUEUE );
if( hCursor )
{
    if ( DSIAttachCursorLast( hCursor,
        NULL,
        0,
        NULL,
        0 ) == DSIERR_SUCCESS )
    {
        DSIAttachCursorName(hCursor,szName,sizeof(szName));
        DSIAttachCursorValue(hCursor,szValue,sizeof(szValue));
        printf( "The last element is %s=%s", szName,szValue );
    }
}
DSIAttachCursorValueEx

Use this function to retrieve the value of the current element from the attachment list.

Syntax

```c
long DSIAttachCursorValueEx(DSIHANDLE hCursor, char* pszValue,
size_t cbValue, long IEncoding);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>encoding</td>
<td>DSIENCODING_SINGLE_BYTE or DSIENCODING_UTF_8. DSIENCODING_SINGLE_BYTE uses codepage 1252 encoding, similar to ASCII but is compatible with Documaker RP handles Euro characters and others. DSIENCODING_UTF_8 translates Unicode into a format compatible with null-terminated C language strings.</td>
</tr>
<tr>
<td>hCursor</td>
<td>handle to attachment cursor initialized by a prior call to DSIOpenAttachCursor and positioned by calls to the DSIAttachCursorFirst, Next, Prev, Last calls.</td>
</tr>
<tr>
<td>pszValue</td>
<td>buffer in which to retrieve the value of the element of the attachment</td>
</tr>
<tr>
<td>cbValue</td>
<td>size of buffer in pszValue parameter</td>
</tr>
</tbody>
</table>

NOTE: The parameter pszValue will be zero-filled to the length specified in cbValue.

Return values

<table>
<thead>
<tr>
<th>Description</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARAM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_NOTFOUND</td>
<td>the position of the cursor is invalid</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

Errors

Invalid parameter

The position of the cursor is invalid

See also

- DSIOpenAttachCursor on page 142
- DSICloseAttachCursor on page 116
- DSIAttachCursorFirst on page 90
- DSIAttachCursorNext on page 100
- DSIAttachCursorLast on page 94
- DSIAttachCursorPrev on page 104
- DSIAttachCursorValue on page 108
- DSIParseAttachment on page 144
Example

```c
DSIHANDLE hApp;
DSIHANDLE hInstance;
DSIHANDLE hCursor;
char szName [ DSI_MAXNAMESIZE ];
char szValue [ DSI_MAXVALUESIZE ];

hApp = DSInit();
hInstance = DSInitInstance( hApp );
hCursor = DSIOpenAttachCursor( hInstance,
                                  DSI_INPUTQUEUE );

if( hCursor )
{
    if ( DSIAAttachCursorLast( hCursor,
                               NULL,
                               0,
                               NULL,
                               0 ) == DSIERR_SUCCESS )
    {
        DSIAAttachCursorName( hCursor, szName, sizeof(szName));
        DSIAAttachCursorValueEx( hCursor, szValue, sizeof(szValue), DSIENCODING_UTF_8 );
        printf( "The last element is %s=%s", szName, szValue );
    }
    .
    .
    .
```

DSIAttachVarLength

Locates an attachment variable and returns its length. Useful for getting the value when the size is unknown and can be huge.

Syntax

```c
long DSIAttachVarLength(DSIHANDLE hdsi, long iQueue, char* szName, size_t *pstSize);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hdsi</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue attachment in which variable is to be found</td>
</tr>
<tr>
<td>pstSize</td>
<td>the size of the value including null terminator</td>
</tr>
<tr>
<td>szName</td>
<td>name of the variable to locate</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVARM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_NOTFOUND</td>
<td>variable not found</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

Errors

Invalid parameter

Variable not found in the attachment

See also

DSIAddAttachVar on page 84
DSIDeleteAttachVar on page 120
DSIParseAttachment on page 144

Example

```c
char *pszVar;
size_t size;

DSIAttachVarLength( hdsi,
                   DSI_INPUTQUEUE,
                   "FileName",
                   &size);
pszVar = malloc(size);
DSILocateAttachVar(hdsi,
                   DSI_INPUTQUEUE,
                   "FileName",
                   pszVar,
                   size);
printf("File is: %s\n",pszVar);
free(pszVar);
```
DSIAttachVarLengthEx

Locates an attachment variable and returns its length. Useful for getting the value when the size is unknown and can be huge.

**Syntax**

```c
long DSIAttachVarLengthEx(DSIHANDLE hdsi, long iQueue, char* szName, size_t *pstSize, long encoding);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>encoding</td>
<td>DSIENCODING_SINGLE_BYTE or DSIENCODING_UTF_8. DSIENCODING_SINGLE_BYTE uses codepage 1252 encoding, similar to ASCII but is compatible with Documaker RP handles Euro characters and others. DSIENCODING_UTF_8 translates Unicode into a format compatible with null-terminated C language strings.</td>
</tr>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue attachment in which variable is to be found</td>
</tr>
<tr>
<td>pstSize</td>
<td>the size of the value including null terminator</td>
</tr>
<tr>
<td>szName</td>
<td>name of the variable to locate</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVARM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_NOTFOUND</td>
<td>variable not found</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

**Errors**

Invalid parameter

Variable not found in the attachment

**See also**

DSIAddAttachVar on page 84

DSIDeleteAttachVar on page 120

DSIParseAttachment on page 144

**Example**

```c
char *pszVar;
size_t size;

DSIAttachVarLengthEx(hdsi,
  DSI_INPUTQUEUE,
  "FileName",
  &size
  DSIENCODING_UTF_8);

pszVar = malloc(size);
DSILocateAttachVarEx(hdsi,
```
DSI_INPUTQUEUE,
"FileName",
pszVar,
size,
DSIENCODING_UTF_8);
printf("File is: %s\n", pszVar);
free(pszVar);
DSICacheFile

Use this function to add a file to the cache. You can only use this API from a server rule. This API adds a row to the table of cached files. The server purges these files as time expires in the autorun rules. This API only works if you have registered the IRLInit rule as an INIT rule on the server.

**Syntax**

```c
long DSICacheFile(DSIHANDLE hInstance, char* szFileName, long lExpire);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>szFileName</td>
<td>full name of file to be added</td>
</tr>
<tr>
<td>lExpire</td>
<td>time period until file should be purged, in seconds</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_MEMORY</td>
<td>out of memory</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

**Errors**

Invalid parameter  
Memory error  
Database error  

**Example**

To set the file to expire in one hour:

```c
DSICacheFile( hInstance, "File.dat", 3600L );
```
DSICloseAttachCursor

Use this function to close an attachment cursor and free the associated memory.

Syntax

```c
long DSICloseAttachCursor(DSIHANDLE hCursor);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCursor</td>
<td>handle of the cursor previously created by a call to DSIOpenAttachCursor</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVParm</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

Errors

Invalid parameter

See also

DSIOpenAttachCursor on page 142

Example

```c
DSIHANDLE hApp;
DSIHANDLE hInstance;
DSIHANDLE hCursor;
char szName [ DSI_MAXNAMESIZE ];
char szValue [ DSI_MAXVALUESIZE ];
hApp = DSInit();
hInstance = DSInitInstance( hApp );
hCursor = DSIOpenAttachCursor( hInstance,
                                   DSI_INPUTQUEUE,
                                   szName,
                                   sizeof(szName),
                                   szValue,
                                   sizeof(szValue));

if( hCursor )
{
    if( DSIAttachCursorFirst( hCursor,
                               szName,
                               sizeof(szName),
                               szValue,
                               sizeof(szValue))
        == DSIERR_SUCCESS )
    {
        printf( "The first element is %s=%s",szName,szValue );
    }
    DSICloseAttachCursor( hCursor );
}
```
DSICopyAttachVars

Use this function to copy all attachment variables from one queue to another.

Syntax

```c
long DSICopyAttachVars(DSIHANDLE hInstance, long iSourceQ);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>iSourceQ</td>
<td>queue attachment from which variables are to be copied</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_MEMORY</td>
<td>out of memory</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

Errors

- Invalid parameter
- Memory error

See also

- DSIAddAttachVar on page 84
- DSILocateAttachVar on page 139
- DSIDeleteAttachVar on page 120

Example

This code copies the attachment variables from the input queue to the output queue.

```c
DSICopyAttachVars( hInstance, DSI_INPUTQUEUE );
```
**DSICopyQRecord**

Use this function to copy a queue record from one queue to another.

**Syntax**

```c
long DSICopyQRecord(DSIHANDLE hInstance, long iSrcQ);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>iSrcQ</td>
<td>queue from which to copy (destination is assumed to be the other queue belonging to the hInstance parameter)</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARAM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_MEMORY</td>
<td>out of memory</td>
</tr>
<tr>
<td>DSIERR_EOF</td>
<td>no queue records available</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
<tr>
<td>DSIERR_QERR</td>
<td>uninitialized queue</td>
</tr>
<tr>
<td>DSIERR_IOERR</td>
<td>end of file</td>
</tr>
</tbody>
</table>

**Errors**

Invalid parameter

Lower level API failure-call DSIGetQError

**Example**

```c
DSICopyQRecord( hInstance, DSI_OUTPUTQUEUE ); /* copy output to input */
```
DSICreateValue

Use this function to create a persistent DSI variable. These variables are not part of the queue records or attachments. They exist so rules can pass information to one another. You must destroy these persistent variables using a call to the DSIDestroyValue function.

Syntax

```c
long DSICreateValue(DSIHANDLE hInstance, char* szName, void* pvValue, size_t cbValueSize);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>szName</td>
<td>name of the variable</td>
</tr>
<tr>
<td>pvValue</td>
<td>pointer to the data (may be NULL)</td>
</tr>
<tr>
<td>cbValueSize</td>
<td>size of data</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARAM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_MEMORY</td>
<td>out of memory</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

Errors

- Invalid parameter
- Memory error

See also

- DSIDestroyValue on page 121
- DSILocateValue on page 141
- DSIQeryValueSize on page 146

Example

```c
int iCount;
iCount = 123;
DSICreateValue(hInstance,"MY_ICOUNT",&iCount,sizeof(iCount));
```
DSIDeleteAttachVar

Use this function to remove an attachment variable.

Syntax

```c
long DSIDeleteAttachVar(DSIHANDLE hInstance, long iQueue, char* szName);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue attachment from which variable is to be removed</td>
</tr>
<tr>
<td>szName</td>
<td>name of the variable to be removed</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARAM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_NOTFOUND</td>
<td>variable not known</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

Errors

- Invalid parameter
- Variable not found in the attachment

See also

- DSIAddAttachVar on page 84
- DSILocateAttachVar on page 139

Example

```c
DSIDeleteAttachVar( hInstance, DSI_OUTPUTQUEUE, "DonotWantThis" );
```
**DSIDestroyValue**

Use this function to destroy a persistent DSI variable. To prevent resource leaks, you must use this function to destroy *all* variables created with the DSICreateValue function.

**Syntax**

```c
long DSIDestroyValue(DSIHANDLE hInstance, char* szName);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>szName</td>
<td>name of the variable to destroy</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INV Paramount</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_NOTFOUND</td>
<td>value not found</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

**Errors**

- Invalid parameter
- Variable cannot be located

**See also**

- DSICreateValue on page 119
- DSILocateValue on page 141
- DSIQueryValueSize on page 146

**Example**

```c
DSIDestroyValue( hInstance, "DISPOSABLE" );
```
Chapter 2
DSI C APIs

DSIEncryptValue

Encrypt a text value to a unique string. It is useful for encrypting USERID or PASSWORD, for example.

Syntax

```c
long DSIEncryptValue(DSIHANDLE hdsi, char* szName, char *pszValue, size_t valSize);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hdsi</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>pszInValue</td>
<td>Input buffer of the text string to be encrypted</td>
</tr>
<tr>
<td>pszOutValue</td>
<td>Output buffer of the encrypted text string</td>
</tr>
<tr>
<td>valSize</td>
<td>size of the output buffer</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARAM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_NOTFOUND</td>
<td>value not found</td>
</tr>
</tbody>
</table>

Errors

Invalid parameter

Example

```c
DSIHANDLE hApp;
DSIHANDLE hInstance;
char outValue ??(DSI_MAXVALUESIZE ??);

hApp=DSIInit();
hInstance=DSInitInstance( hApp );
DSIEncryptValue(hInstance, inValue, outValue, sizeoff(outValue));
.
.
.
```
DSIEncryptValueEx

Encrypt a text value to a unique string. It is useful for encrypting USERID or PASSWORD, for example.

Syntax

```c
long DSIEncryptValueEx(DSIHANDLE hdsi, char* szName, char *pszValue, size_t valSize, long IEncoding);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>encoding</td>
<td>DSIENCODING_SINGLE_BYTE or DSIENCODING_UTF_8. DSIENCODING_SINGLE_BYTE uses codepage 1252 encoding, similar to ASCII but is compatible with Documaker RP handles Euro characters and others. DSIENCODING_UTF_8 translates Unicode into a format compatible with null-terminated C language strings.</td>
</tr>
<tr>
<td>hdsi</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>pszInValue</td>
<td>Input buffer of the text string to be encrypted</td>
</tr>
<tr>
<td>pszOutValue</td>
<td>Output buffer of the encrypted text string</td>
</tr>
<tr>
<td>valSize</td>
<td>size of the output buffer</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_NOTFOUND</td>
<td>value not found</td>
</tr>
</tbody>
</table>

Errors

Invalid parameter

Example

```c
DSIHANDLE hApp;
DSIHANDLE hInstance;
char outValue ??(DSI_MAXVALUESIZE ??);

hApp=DSIInit();
hInstance=DSIInitInstance( hApp );
DSIEncryptValueEx(hInstance, inValue, outValue, sizeoff(outValue), DSIENCODING_UTF_8);
```

.

.

.
DSIErrorMessage

Use this function to add an error message to an attachment.

Syntax

```
long DSIErrorMessage(DSIHANDLE hInstance, long iQueue, char* pszCode, ...);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue attachment to which message should be added</td>
</tr>
<tr>
<td>pszCode</td>
<td>error code</td>
</tr>
<tr>
<td>...</td>
<td>error parameter name/value pairs, terminated by NULL</td>
</tr>
</tbody>
</table>

The variable arguments must be in this format:

```
<ERR.MSG>,<ParameterName><ParameterValue>
<ParameterName><ParameterValue>
...NULL
```

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_MEMORY</td>
<td>out of memory</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

Errors

Invalid parameter

Memory error

Example

```c
DSIErrorMessage(    hInstance,
DSI_OUTPUTQUEUE,
"IRL0023", /* error code */
"FILE",    /* error parameter name */
szFile,    /* error parameter value */
NULL );    /* NULL terminator */
```
DSIErrorMsg

Use this function to add an error message to an attachment. This function serves as a replacement for the DSIErrorMessage function in situations where a variable number of arguments is not supported, such as with languages other than C and C++.

Syntax

```c
long DSIErrorMsg ( DSIHANDLE hdsi, long iQueue, long lLevel, char **pszCode);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hdsi</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue attachment to which message should be added</td>
</tr>
<tr>
<td>lLevel</td>
<td>DSI_ERROPT_ value, level of the error. Valid values are: DSI_ERROPT_INFO, DSI_ERROPT_WARNING, DSI_ERROPT_SEVERE (not currently implemented and is ignored).</td>
</tr>
<tr>
<td>pszCode</td>
<td>pointer to the array of strings, the last string has to be NULL, the first string is the error code. The strings are in NAME/VALUE pairs.</td>
</tr>
</tbody>
</table>

To add the error message to the attachment, pass to it this array of strings:

```
"XXX0001",  /* error code
"FILENAME",  /* error parameter name
"C:\docser\file.dat",  /* error parameter value
NULL        /* NULL terminator
```

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARAM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_INTERNAL</td>
<td>internal error</td>
</tr>
</tbody>
</table>

Errors

Invalid parameter

Internal error

Example

```c
char *err ??(??) =
{
    "XXX0023",  /* error code
    "FILE",    /* error parameter name
    "C:\\docser\\file.dat",  /* error parameter value
    NULL        /* NULL terminator
};
DSIErrorMsg( hInstance,
            DSI_OUTPUTQUEUE,
            err );
```
DSIFindInQueue

Use this function to search for a record in a queue.

Syntax

```c
long DSIFindInQueue(DSIHANDLE hInstance, long iQueue, char* pszId);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue in which to search</td>
</tr>
<tr>
<td>pszId</td>
<td>unique record identifier. Use DSISetQField( ..., DSIQSET_UNIQUE_ID, ... ) to place this value in the queue record</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_EOF</td>
<td>record not found</td>
</tr>
<tr>
<td>DSIERR_MEMORY</td>
<td>out of memory</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
<tr>
<td>DSIERR_QERR</td>
<td>uninitialized queue</td>
</tr>
<tr>
<td>DSIERR_IOERR</td>
<td>end of file</td>
</tr>
</tbody>
</table>

Errors

Invalid parameter

Lower level API failure-call DSIGetQError

Example

```c
char szId [ 11 ];
DSIGetUniqueString( hInstance, szId, sizeof( szId ) );
DSISetQField( hInstance, DSI_OUTPUTQUEUE, DSIQSET_UNIQUE_ID, szId, sizeof( szId ) );
DSIAddToQueue( hInstance, DSI_OUTPUTQUEUE );
/* wait for server to process */
DosSleep( 5000 );
DSIFindInQueue( hInstance, DSI_INPUTQUEUE, szId );
```
DSIGetFirstFromQueue

Use this function to get the first record in a queue.

**Syntax**

```c
long DSIGetFirstFromQueue(DSIHANDLE hInstance, long iQueue);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue from which to retrieve</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARAM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_MEMORY</td>
<td>out of memory</td>
</tr>
<tr>
<td>DSIERR_EOF</td>
<td>no elements in the list</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
<tr>
<td>DSIERR_QERR</td>
<td>uninitialized queue</td>
</tr>
<tr>
<td>DSIERR_IOERR</td>
<td>end of file</td>
</tr>
</tbody>
</table>

**Errors**

- Invalid parameter
- Lower level API failure-call DSIGetQError

**Example**

```c
DSIGetFirstFromQueue( hInstance, DSI_INPUTQUEUE );
```
DSIGetSOAPMessage

Use this rule to retrieve an IDS message as an XML file in memory.

Syntax

```c
long DSIGetSOAPMessage ( DSIHANDLE hdsi, long IQueue, long szXMLBuffer, long szXMLBuffer, long stBuffSize, long IOptions;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hdsi</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue attachment</td>
</tr>
<tr>
<td>szXMLBuffer</td>
<td>buffer into which the XML is to be unloaded</td>
</tr>
<tr>
<td>stBuffSize</td>
<td>size of buffer in szXMLBuffer including the zero (0) terminator</td>
</tr>
<tr>
<td>IOptions</td>
<td>RFU, currently not used</td>
</tr>
</tbody>
</table>

Returns

DSIERR_SUCCESS
DSIERR_INVPARM

Errors

Invalid parameter

Example

```c
char *buf;
size_t size;

DSIGetSOAPMessageSize(hdsi,DSI_INPUT,&size,0);
buf = malloc(size);
DSIGetSOAPMessage(hdsi,DSI_INPUT,buf,size,0);

... use buffer here

free(buf);
```
**DSIGetSOAPMessageSize**

Use this rule to get the size of an IDS message as an XML file in memory.

**Syntax**

```c
long DSIGetSOAPMessageSize ( DSIHANDLE hdsi, long iQueue, long pstBuffSize, long lOptions;
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hdsi</td>
<td>handle to instance returned by DSInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue attachment</td>
</tr>
<tr>
<td>pstBuffSize</td>
<td>size of buffer in szXMLBuffer including the zero (0) terminator</td>
</tr>
<tr>
<td>lOptions</td>
<td>RFU, currently not used</td>
</tr>
</tbody>
</table>

**Returns**

DSIERR_SUCCESS

DSIERR_INVParm

**Errors**

Invalid parameter

**Example**

```c
char *buf;
size_t size;

DSIGetSOAPMessageSize(hdsi, DSI_INPUT, &size, 0);
buf = malloc(size);
DSIGetSOAPMessage(hdsi, DSI_INPUT, buf, size, 0);

... use buffer here

free(buf);
```
**DSIGetQError**

Use this function to get the last queue error from a queue.

**Syntax**

```c
long DSIGetQError(DSIHANDLE hInstance, long iQueue);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue from which to retrieve error</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_MEMORY</td>
<td>out of memory</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
<tr>
<td>DSIERR_QERR</td>
<td>uninitialized queue</td>
</tr>
</tbody>
</table>

**Errors**

Invalid parameter

**Example**

```c
long QErr;
if( DSIGetFirstFromQueue( hInstance, DSI_INPUTQUEUE, 0L )
   != DSIERR_SUCCESS )
{
    QErr = DSIGetQError( hInstance, DSI_INPUTQUEUE );
}
```
**DSIGetQField**

Use this function to retrieve the value of a queue field.

**NOTE:** Since each field has a different length which may vary from one release to the next, the system queries the length before it allocates memory and performs this function.

### Syntax

```c
long DSIGetQField(DSIHANDLE hInstance, long iQueue, long iField, void* pvValue, size_t cbValue);
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue to which operation applies</td>
</tr>
<tr>
<td>iField</td>
<td>DSIQSET_* field identifier. For example: REQTYPE (must be three characters in length) STATUS INTIME OUTTIME USERID PRIORITY UNIQUE_ID ATTACHMENT</td>
</tr>
<tr>
<td>pvValue</td>
<td>buffer in which the data should be placed</td>
</tr>
<tr>
<td>cbValue</td>
<td>length of the buffer</td>
</tr>
</tbody>
</table>

### Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_MEMORY</td>
<td>out of memory</td>
</tr>
<tr>
<td>DSIERR_EOF</td>
<td>queue record not found</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
<tr>
<td>DSIERR_QERR</td>
<td>uninitialized queue</td>
</tr>
<tr>
<td>DSIERR_IOERR</td>
<td>end of file</td>
</tr>
</tbody>
</table>

### Errors

- Invalid parameter
- Low level API failure-call DSIGetQError
See also DSISetQField on page 156

Example

```c
char szRequest[8];
DSIGetQField(hInstance,
DSI_INPUTQUEUE,
DSIQSET_REQTYPE,
szRequest,
szRequest);  // Use sizeof(szRequest)
if(!strcmp(szRequest,"LGN"))
{
  //...
}
```
**DSIGetQFieldLength**

Use this function to get the length of one of the pre-defined fields in a queue.

**Syntax**

```c
long DSIGetQFieldLength(DSIHANDLE hInstance, long iQueue, long iField);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue from which to retrieve data</td>
</tr>
<tr>
<td>iField</td>
<td>DSIQSET_&quot; field identifier. For example: REQTYPE (must be three characters in length) STATUS INTIME OUTTIME USERID PRIORITY UNIQUE_ID ATTACHMENT</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>error</td>
</tr>
<tr>
<td>0</td>
<td>length of field</td>
</tr>
</tbody>
</table>

**Errors**

Invalid parameter

Lower level API failure—Call DSIGetQError

**Example**

```c
void *pvAttach;
long cbField;

cbField = DSIGetQFieldLength(   hInstance,
                             DSI_INPUTQUEUE,
                             DSIQSET_ATTACHMENT );

if( cbField > 0 )
{
    DosAllocMem(    ( PPVOID )&pvAttach,
                    cbField,
                    PAG_READ | PAG_WRITE | PAG_COMMIT );
}
```
### DSIGetQueueRec

Use this function to search for a record in a queue.

**Syntax**

```c
long DSIGetQueueRec(DSIHANDLE hInstance, long iQueue, char* pszId, long lWait, long lTimeOut);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue in which to search</td>
</tr>
<tr>
<td>pszId</td>
<td>unique record identifier. Use DSISetQField(..., DSIQSET_UNIQUE_ID, ...) to place this value in the queue record</td>
</tr>
<tr>
<td>lWait</td>
<td>number of milliseconds to wait between retries, zero (0) is invalid for this parameter and is replaced with 1000.</td>
</tr>
<tr>
<td>lTimeOut</td>
<td>number of milliseconds to keep trying, if zero (0) the system does not retry</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_MEMORY</td>
<td>out of memory</td>
</tr>
<tr>
<td>DSIERR_EOF</td>
<td>record set not found</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
<tr>
<td>DSIERR_QERR</td>
<td>uninitialized queue</td>
</tr>
<tr>
<td>DSIERR_IOERR</td>
<td>end of file</td>
</tr>
</tbody>
</table>

**Errors**

Invalid parameter

Lower level API failure-call DSIGetQError

**Example**

```c
char szId[11];
DSIGetUniqueString( hInstance, szId, sizeof(szId) );
DSISetQField( hInstance,
    DSI_OUTPUTQUEUE,
    DSIQSET_UNIQUE_ID,
    szId,
    sizeof(szId) );
DSIAddToQueue( hInstance, DSI_OUTPUTQUEUE );
/* wait for server to process */
DSIGetQueueRec( hInstance, DSI_INPUTQUEUE, szId, 1000L, 10000L );
/* tries every second for 10 seconds */
```
DSIGetUniqueString

Use this function to fill the buffer pointed to by pszString with a unique string. You can use this function to generate unique file names. The buffer is filled with characters of the size specified by the cbSize parameter less one. So, if you need to generate an 8-character unique file name, specify a buffer size of 9. The output string is unique for the current instance of the Internet Document Server.

Syntax

long DSIGetUniqueString(DSIHANDLE hInstance, char* pszString, size_t cbSize);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSInitInstance</td>
</tr>
<tr>
<td>pszString</td>
<td>pointer to the output buffer</td>
</tr>
<tr>
<td>cbSize</td>
<td>size of buffer in pszString</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVARM</td>
<td>Invalid parameter</td>
</tr>
<tr>
<td></td>
<td>hInstance is NULL</td>
</tr>
<tr>
<td></td>
<td>pszString is NULL</td>
</tr>
<tr>
<td></td>
<td>cbSize is 0</td>
</tr>
<tr>
<td>DSIERR_MEMORY</td>
<td>memory errors</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

Errors

Invalid parameters

Memory errors

Example

char szFileName DIM { 9 };
if ( DSIGetUniqueString ( hInstance, szFileName, sizeof(szFileName) ) != DSIERR_SUCCESS )
{
    Some code to display error message
}
DSIInit

Use this function to initialize the systems and structures necessary for DSI calls. This should be called by the application only once.

This rule loads the DSI.INI file, which you can use to store DSI internal INI options, such as queue names. If the INI does not exist, no error is given.

Syntax

`DSIHANDLE DSIInit();`

Parameters

None

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIHANDLE</td>
<td>handle to application data to be used for subsequent calls to DSIInitInstance and DSITerm</td>
</tr>
<tr>
<td>DSINULLHANDLE</td>
<td>on failure</td>
</tr>
</tbody>
</table>

Errors

Memory error

See also

`DSITerm` on page 159

Example

```c
DSIHANDLE hApp;
if( ( hApp = DSIInit() ) == DSINULLHANDLE )
{
    return( FALSE );
}
```
DSIInitInstance

Use this function to initialize the structures necessary for DSI calls. This should be called once per thread.

Syntax

```c
DSIHANDLE DSIInitInstance(DSIHANDLE hApp);
```

Parameters

- **hApp**: handle of application data returned by a prior call to DSIInit

Return values

- **DSIHANDLE hInstance**: handle to instance data, returns 0 on error
- **DSINULLHANDLE**: returns on failure

Errors

- Memory allocation failure
- Rule data initialization failure

See also

- DSIInit on page 136
- DSITermInstance on page 160

Example

```c
DSIHANDLE hApp;
DSIHANDLE hInstance;
hApp = DSIInit();
hInstance = DSIInitInstance( hApp );
DoSomeStuff( hInstance, andSomeOtherParameters );
DSITermInstance( hInstance );
DSITerm( hApp );
return( -10368 );
```
DSIInitQueue

Use this function to initialize a queue.

**Syntax**

```c
long DSIInitQueue(DSIHANDLE hInstance, long iQueue, char* pszQName);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue to initialize</td>
</tr>
<tr>
<td>pszQName</td>
<td>name of queue to initialize.</td>
</tr>
</tbody>
</table>

The iQueue parameter tells the system whether to initialize the request (REQUESTQ) or result (RESULTQ) queue. If the pszQName parameter is NULL, the rule uses the NameINI option in the REQUESTQ or RESULTQ control group. If found, it will use this name for the output (or input) queue name. These names have default values which are used when the name passed in is NULL and no INI option is specified in the DSI.INI file. The default names are REQUESTQ for output and RESULTQ for input queues.

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_MEMORY</td>
<td>out of memory</td>
</tr>
<tr>
<td>DSIERR_EOF</td>
<td>record not found</td>
</tr>
<tr>
<td>DSIERR_NOTFOUND</td>
<td>no more elements in the list</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
<tr>
<td>DSIERR_QERR</td>
<td>uninitialized queue</td>
</tr>
<tr>
<td>DSIERR_IOERR</td>
<td>end of file</td>
</tr>
</tbody>
</table>

**Errors**

Invalid parameter

Lower level API failure-call DSIGetError

**See also**

DSITermQueue on page 161

**Example**

```c
long rc;
if( DSIInitQueue( hInstance, DSI_INPUTQUEUE, "InputQ" )
    != DSIERR_SUCCESS )
{
    rc = DSIGetQError( hInstance, DSI_INPUTQUEUE );
}
```
Use this function to locate an attachment variable. You must call the DSIParseAttachment function before you use this function.

**Syntax**

```c
long DSILocateAttachVar(DSIHANDLE hInstance, long iQueue, char* szName,
                        char* szValue, size_t cbValSize);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue attachment in which variable is to be found</td>
</tr>
<tr>
<td>szName</td>
<td>name of the variable to locate</td>
</tr>
<tr>
<td>szValue</td>
<td>buffer for the variable</td>
</tr>
<tr>
<td>cbValSize</td>
<td>size of buffer in szValue</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_NOTFOUND</td>
<td>variable not found</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

**Errors**

Invalid parameter

Variable not found in the attachment

**See also**

DSIAddAttachVar on page 84
DSIDeleteAttachVar on page 120
DSIParseAttachment on page 144

**Example**

```c
char szVar[32];
DSILocateAttachVar( hInstance, DSI_INPUTQUEUE, "FileName", szVar, sizeof(szVar) );
```
**DSILocateAttachVarEx**

Use this function to locate an attachment variable. You must call the DSIParseAttachment function *before* you use this function.

**Syntax**

```c
long DSILocateAttachVarEx(DSIHANDLE hdsi, long iQueue, char* szName, char* szValue, size_t cbValSize, long IEncoding);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>encoding</td>
<td>DSIENTCENCODING_SINGLE_BYTE or DSIENTCENCODING_UTF_8. DSIENTCENCODING_SINGLE_BYTE uses codepage 1252 encoding, similar to ASCII but is compatible with Documaker RP handles Euro characters and others. DSIENTCENCODING_UTF_8 translates Unicode into a format compatible with null-terminated C language strings.</td>
</tr>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue attachment in which variable is to be found</td>
</tr>
<tr>
<td>szName</td>
<td>name of the variable to locate</td>
</tr>
<tr>
<td>szValue</td>
<td>buffer for the variable</td>
</tr>
<tr>
<td>cbValSize</td>
<td>size of buffer in szValue</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIER_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIER_INVPARM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIER_NOTFOUND</td>
<td>variable not found</td>
</tr>
<tr>
<td>DSIER_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

**Errors**

Invalid parameter

Variable not found in the attachment

**See also**

- DSIAddAttachVar on page 84
- DSIDeleteAttachVar on page 120
- DSIParseAttachment on page 144

**Example**

```c
char szVar [ 32 ];
DSILocateAttachVarEx( hInstance,
    DS_INPUTQUEUE,
    "FileName",
    szVar,
    sizeof( szVar ),
    DSIENTCENCODING_UTF_8);
```
DSILocateValue

Use this function to locate a persistent DSI variable.

**Syntax**

```c
long DSILocateValue(DSIHANDLE hInstance, char* szName, void* pvValue, size_t cbValueSize);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>szName</td>
<td>name of the variable to locate</td>
</tr>
<tr>
<td>pvValue</td>
<td>buffer in which to place the data</td>
</tr>
<tr>
<td>cbValueSize</td>
<td>size of buffer</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_NOTFOUND</td>
<td>named value not found</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

**Errors**

- Invalid parameter
- Variable not found

**See also**

- DSICreateValue on page 119
- DSIDestroyValue on page 121
- DSIQueryValueSize on page 146

**Example**

```c
char szFile [ CCHMAXPATH ];
DSILocateValue( hInstance, "FILENAME", szFile, sizeof( szFile ) );
```
**DSIOpenAttachCursor**

Use this function to open a cursor into the attachment list for the specified queue.

**Syntax**

```c
DSIHANDLE DSIOpenAttachCursor(DSIHANDLE hInstance, long iQ);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance data initialized by a prior call to DSIInitInstance</td>
</tr>
<tr>
<td>iQ</td>
<td>queue identifier</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIHANDLE</td>
<td>handle to cursor which you can use for subsequent calls to the DSIAttachCursorFirst, DSIAttachCursorNext, DSIAttachCursorPrev and DSICloseAttachCursor functions.</td>
</tr>
<tr>
<td>DSINULLHANDLE</td>
<td>on failure</td>
</tr>
</tbody>
</table>

**Errors**

Invalid parameter

Memory allocation failure

Uninitialized queue

See also

- DSIAttachCursorFirst on page 90
- DSIAttachCursorNext on page 100
- DSIAttachCursorLast on page 94
- DSIAttachCursorPrev on page 104
- DSIAttachCursorName on page 98
- DSIAttachCursorValue on page 108
- DSICloseAttachCursor on page 116
- DSIParseAttachment on page 144

**Example**

```c
DSIHANDLE hApp;
DSIHANDLE hInstance;
DSIHANDLE hCursor;
char szName[DSI_MAXNAMESIZE];
char szValue[DSI_MAXVALUESIZE];
```
hApp = DSILogInit();
hInstance = DSILogInitInstance( hApp );
hCursor = DSILogOpenAttachCursor( hInstance,
DSI_INPUTQUEUE );

if ( DSILogAttachCursorFirst( hCursor,
   szName,
   sizeof(szName),
   szValue,
   sizeof(szValue)) == DSIERR_SUCCESS )
{
    printf( "The first element is: %s = %s", szName, szValue );
}


DSIParseAttachment

Use this function to parse the attachment field in the queue record into an internal attachment list of name/value pairs.

Syntax

```
long DSIParseAttachment(DSIHANDLE hInstance, long iQueue);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue in which the attachment is to be parsed</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARAM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_MEMORY</td>
<td>out of memory</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

Errors

Invalid parameter

Uninitialized attachment list

See also

DSIStoreAttachment on page 158

Example

```
DSIParseAttachment( hInstance, DSI_INPUTQUEUE );
```
DSIQueryEnvOptions

Use this function to return DSI-specific environment options via DSIENV_* flags. You can use this function to determine if a rule is running on the client or on the server.

Syntax

```c
_DSIXPORT long _DSIAPI DSIQueryEnvOptions ( DSINHANDLE hInstance, long *plOptions );
```

These flags are currently available:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Available on the...</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIENV_SERVER</td>
<td>server</td>
</tr>
<tr>
<td>DSIENV_CLIENT</td>
<td>client</td>
</tr>
<tr>
<td>DSIENV_SERVICE</td>
<td>server as an NT service</td>
</tr>
</tbody>
</table>

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSInitInstance</td>
</tr>
<tr>
<td>plOptions</td>
<td>pointer to a long for returning the DSIENV_* values.</td>
</tr>
</tbody>
</table>

Return values

DSIERR_SUCCESS or an error code

Error conditions

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_INVPARAM</td>
<td>invalid DSI instance handle or plOptions is NULL</td>
</tr>
<tr>
<td>DSIERR_INTERNAL</td>
<td>internal error</td>
</tr>
</tbody>
</table>

Example

```c
long lOpt;
if ( DSIQueryEnvOptions(hInstance,&lOpt) != DSIERR_SUCCESS ) {
    ... display error message
}
if ( lOpt & DSIENV_SERVER )
    {
    printf("Running on the server\n");
}
if ( lOpt & DSIENV_CLIENT )
    {
    printf("Running on the client\n");
}
```
DSIQueryValueSize

Use this function to find the length of a persistent DSI variable.

Syntax

```c
size_t DSIQueryValueSize(DSIHANDLE hInstance, char* szName);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>szName</td>
<td>name of the variable to locate</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>error</td>
</tr>
<tr>
<td>0</td>
<td>variable size</td>
</tr>
</tbody>
</table>

Error conditions

- Invalid parameter
- variable not found

See also

- DSICreateValue on page 119
- DSIDestroyValue on page 121
- DSILocateValue on page 141

Example

```c
size_t cbVar;
cbVar = DSIQueryValueSize( hInstance, "FILENAME" );
```
DSIReceiveFile

Use this function to get a file from an attachment and write that file to disk. This function supports text (such as XML or RTF) and binary files. The size of file is limited to the queue message size. Use this function with the DSISendFile function.

NOTE: XML files can have very long lines. If the line length is over 1K, use the binary file send/receive option. The binary send/receive works with any file, including XML and other text files.

**Syntax**

```c
DSIReceiveFile(hdsi, iQueue, pszFileName, pszAttachName, iOptions );
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hdsi</td>
<td>The handle to the instance returned by DSInitInstance.</td>
</tr>
<tr>
<td>iQueue</td>
<td>The queue attachment to which the file was added by the DSISendFile function.</td>
</tr>
<tr>
<td>pszFileName</td>
<td>The full name of the output file you want to create.</td>
</tr>
<tr>
<td>pszAttachName</td>
<td>The name of the attachment variable to find file data.</td>
</tr>
<tr>
<td>iOptions</td>
<td>Currently supported options are DSIFILE_TEXT and DSIFILE_BINARY. These options are mutually exclusive. This value should be the same as was used with the DSISendFile function.</td>
</tr>
</tbody>
</table>

**Return values**

- DSIERR_SUCCESS
- DSIERR_INVVPARM
- DSIERR_IOERR

**Error conditions**

- Invalid parameter
- Memory error
- IO error

**Example**

```c
DSIReceiveFile( hdsi,
    DSI_INPUTQUEUE,
    "c:\docserv\a.txt", /* file name
    "FILESEND",       /* attachment variable name
    DSIFILE_TEXT );    /* option, file is text file
```
DSIReceiveFileAsBuffer

Use this function to get a file from an attachment and copy it into a passed in buffer. This function supports both text and binary files. The size of file is limited to the one queue message size. You must use this function with the DSISendFile function.

**Syntax**

```c
DSIReceiveFileAsBuffer (hdsi, iQueue, pszFileName, pszAttachName, pBuffer, cbSize, iOptions );
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hdsi</td>
<td>handle to instance returned by DSInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue attachment to which the file was added by DSISendFile</td>
</tr>
<tr>
<td>pszAttachName</td>
<td>name of the attachment variable to find file data</td>
</tr>
<tr>
<td>pBuffer</td>
<td>output, the buffer to receive file data, buffer should be large enough to hold the whole file data. Use the DSIReceiveFileAsBufferSize function to determine the size.</td>
</tr>
<tr>
<td>cbSize</td>
<td>allocated size of buffer in pBuffer</td>
</tr>
<tr>
<td>iOptions</td>
<td>RFU, currently not used</td>
</tr>
</tbody>
</table>

**Return values**

- DSIERR_SUCCESS
- DSIERR_INVVPARM

**Errors**

- Invalid parameter
- Memory error

**Example**

Here is an example:

```c
size_t size;
char *buffer;

if ( DSIReceiveFileAsBufferSize(hdsi,
    DSI_INPUTQUEUE,
    "FILESEND",
    &size,
    0) != DSIERR_SUCCESS )
{
    printf("Error in DSIReceiveFileAsBufferSize\n");
    return -1;
}

buffer = malloc(size); /* allocate the right size
if ( buffer == NULL )
{
    printf("Cannot allocate buffer\n");
```
if ( DSIReceiveFileAsBuffer(hdsi, DSI_INPUTQUEUE, "FILESEND", buffer, size, 0) != DSIERR_SUCCESS )
{
    printf("ReceiveFile failed\n");
}
.. here application can do whatever is needed with the buffer ..
free(buffer); /* free the buffer
DSIReceiveFileAsBufferSize

Use this function to get the actual size of file from an attachment. This function supports both text and binary files. The size of file is limited to the one queue message size. You must use this function with the DSISendFile function.

Syntax

```c
DSIReceiveFileAsBufferSize(hdsi, iQueue, pszAttachName, pstSize, iOptions);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hdsi</td>
<td>handle to instance returned by DSInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue attachment to which the file was added by DSISendFile</td>
</tr>
<tr>
<td>pszAttachName</td>
<td>name of the attachment variable to find file data,</td>
</tr>
<tr>
<td>pstSize</td>
<td>output, the size of file data in attachment</td>
</tr>
<tr>
<td>iOptions</td>
<td>RFU, currently not used</td>
</tr>
</tbody>
</table>

Return values

- DSIERR_SUCCESS
- DSIERR_INVtparam

Errors

- Invalid parameter
- Memory error

Example

```c
Here is an example:

```c

```c
size_t size;
char *buffer;

if ( DSIReceiveFileAsBufferSize(hdsi, DSI_INPUTQUEUE, "FILESEND", &size, 0) != DSIERR_SUCCESS )
{
    printf("Error in DSIReceiveFileAsBufferSize\n");
    return -1;
}
buffer = malloc(size); /* allocate the right size
if ( buffer == NULL )
{
    printf("Cannot allocate buffer\n");
}
if ( DSIReceiveFileAsBuffer(hdsi, DSI_INPUTQUEUE, "FILESEND", buffer, size, 0) != DSIERR_SUCCESS )
{
```
printf("ReceiveFile failed\n");
}
.. here application can do whatever is needed with the buffer ..

free(buffer); /* free the buffer

On the ASP side, you can use this code:

buff = DSI.ReceiveFileAsBuffer ( "ZZLPDF" )
Response.ContentType = "application/PDF"
Response.BinaryWrite buff

Where ZZLPDF is the name used in the ATCSendFile rule in DOCSERV configuration file.
**DSIRowset2XML**

Use this function to get a row set back as XML in memory. A row set is a collection of attachment variables created using the DSIAAddRecord and DSIAAddToRecord functions.

**Syntax**

```c
DSIRowset2XML( hdsi, iQueue, pszRowset, szXMLBuffer stBuffSize, lOptions );
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hdsi</td>
<td>handle to instance returned by DSInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue attachment to which the row set was added by DSIAAddRecord</td>
</tr>
<tr>
<td>pszRowset</td>
<td>name of the row set to get</td>
</tr>
<tr>
<td>szXMLBuffer</td>
<td>buffer into which the XML is to be unloaded</td>
</tr>
<tr>
<td>stBuffSize</td>
<td>size of buffer in szXMLBuffer including the zero terminator</td>
</tr>
<tr>
<td>lOptions</td>
<td>RFU, currently not used</td>
</tr>
</tbody>
</table>

**Returns**

- DSIERR_SUCCESS
- DSIERR_NOTFOUND
- DSIERR_INVPARM

**Errors**

- Invalid parameter
- Variable cannot be located or is not a row set

**See also**

- DSIRowset2XMLSize on page 153

**Example**

```c
char *buf;
size_t size;

DSIRowset2XMLSize(hdsi, DSI_INPUT, "LIBRARIES", &size, 0);
buf = malloc(size);
DSIRowset2XML(hdsi, DSI_INPUT, "LIBRARIES", buf, size, 0);

... use buffer here

free(buf);
```
DSIRowset2XMLSize

Use this function to get the size of row set back as XML in memory. A row set is a collection of attachment variables created using the DSIAddRecord and DSIAddToRecord functions.

Syntax

```
DSIRowset2XMLSize(hdsi, iQueue, pszRowset, pstSize, iOptions);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hdsi</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue attachment to which the row set was added by DSIAddRecord</td>
</tr>
<tr>
<td>pszRowset</td>
<td>name of the row set to get</td>
</tr>
<tr>
<td>pstSize</td>
<td>output, the size of row set in XML format</td>
</tr>
<tr>
<td>iOptions</td>
<td>RFU, currently not used</td>
</tr>
</tbody>
</table>

Returns

- DSIERR_SUCCESS
- DSIERR_NOTFOUND
- DSIERR_INVVPARM

Errors

Invalid parameter

Variable cannot be located or is not a row set

See also

DSIRowset2XML on page 152

Example

```c
char *buf;
size_t size;

DSIRowset2XMLSize(hdsi, DSI_INPUT, "LIBRARIES", &size, 0);
buf = malloc(size);
DSIRowset2XML(hdsi, DSI_INPUT, "LIBRARIES", buf, size, 0);

... use buffer here

free(buf);
```
DSISendBuffer

Use this function to add a file to an attachment so it can be received on the other end. This function supports text and binary files. The size of file is limited to the one queue message size.

The file being sent is provided to this API as a buffer in memory. It can be used when the data is in memory to eliminate unnecessary IO operation.

When text buffer is used, the new line character is the delimiter for each line. For text, send the lines delimited only by the new line character. Do not use carriage returns. If the line is longer than 1024 bytes, use the binary send method.

Syntax

DSISendBuffer(hdsi, iQueue, pszAttachName, pBuffer, cbSize, iOptions);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hdsi</td>
<td>The handle to the instance returned by DSIInitInstance.</td>
</tr>
<tr>
<td>iQueue</td>
<td>The queue attachment to which the file should be added, usually output.</td>
</tr>
<tr>
<td>pszAttachName</td>
<td>The name of the attachment variable to use for the file data. This name is</td>
</tr>
<tr>
<td></td>
<td>used on the receiving end to retrieve file data from the queue.</td>
</tr>
<tr>
<td>pBuffer</td>
<td>The buffer with file data.</td>
</tr>
<tr>
<td>cbSize</td>
<td>The size of data in pBuffer, if text is being sent the size does not need</td>
</tr>
<tr>
<td></td>
<td>to include the null terminator character.</td>
</tr>
<tr>
<td>iOptions</td>
<td>Currently supported options are DSIFILE_TEXT and DSIFILE_BINARY. These</td>
</tr>
<tr>
<td></td>
<td>options are mutually exclusive.</td>
</tr>
</tbody>
</table>

Returns

DSIERR_SUCCESS
DSIERR_INVPARM
DSIERR_MEMORY

Errors

Invalid parameter
Memory error

Example

Here is an example:

```c
DSISendBuffer( hdsi,
    DSI_OUTPUTQUEUE,
    "FILESEND",      /* attachment variable name */
    buffer,          /* file data */
    strlen(buffer),  /* length of file data */
    DSIFILE_TEXT );  /* option, file is text file */
```
DSISendFile

Use this function to add a file to an attachment so it can be received on the other end. This function supports text (such as XML or RTF) and binary files. The size of file is limited to the queue message size.

**NOTE:** XML files can have very long lines. If the line length is over 1K, use the binary file send/receive option. The binary send/receive works with any file, including XML and other text files.

### Syntax

```
DSISendFile( hdsi, iQueue, pszFileName, pszAttachName, iOptions);
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hdsi</td>
<td>The handle to the instance returned by DSInitInstance.</td>
</tr>
<tr>
<td>iQueue</td>
<td>The queue attachment to which the file should be added.</td>
</tr>
<tr>
<td>pszFileName</td>
<td>The full name of the output file you want to send.</td>
</tr>
<tr>
<td>pszAttachName</td>
<td>The name of the attachment variable to use for file data. You must use this same name in the DSIReceiveFile rule to get the file.</td>
</tr>
<tr>
<td>iOptions</td>
<td>Currently supported options are DSIFILE_TEXT and DSIFILE_BINARY. These options are mutually exclusive.</td>
</tr>
</tbody>
</table>

### Return values

- **DSIERR_SUCCESS**
- **DSIERR_INVOPARM**
- **DSIERR_IOERR**

### Error conditions

- Invalid parameter
- Memory error
- IO error

### Example

```
DSISendFile(    hdsi,
DSI_OUTPUTQUEUE,
"c:\\docserv\\a.txt",  /* file name
"FILESEND",            /* attachment variable name
DSIFILE_TEXT );        /* option, file is text file
```
DSISetQField

Use this function to set a queue field. The system includes several pre-defined queue fields (see IQueue in the table below) which you can set and retrieve. These fields are used by the standard rules and the rule engine.

Syntax

```c
long DSISetQField(DSIHANDLE hInstance, long iQueue, long iField,
                   void* pvValue, size_t cbValue);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue to which operation applies</td>
</tr>
<tr>
<td>iField</td>
<td>DSIQSET_* field identifier. For example: REQTYPE (must be three characters in length) STATUS INTIME OUTTIME USERID PRIORITY UNIQUE_ID</td>
</tr>
<tr>
<td>pvValue</td>
<td>data to copy into queue field</td>
</tr>
<tr>
<td>cbValue</td>
<td>length of pvValue parameter (including the trailing null)</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INV Parm</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
<tr>
<td>DSIERR_MEMORY</td>
<td>out of memory</td>
</tr>
<tr>
<td>DSIERR_EOF</td>
<td>record not found</td>
</tr>
<tr>
<td>DSIERR_QERR</td>
<td>uninitialized queue</td>
</tr>
<tr>
<td>DSIERR_IOERR</td>
<td>end of file</td>
</tr>
</tbody>
</table>

Errors

Invalid parameter

Lower level API failure-call DSIGetQError (This is often the name of a record identified in the DOCSERV configuration file.)

See also  [DSIGetQField on page 131](#)
Example

```
DSISetQField(hInstance,
DSI_OUTPUTQUEUE,
DSIQSET_REQTYPE,
"LGN",
sizeof( "LGN" ) );
```
DSIStoreAttachment

Use this function to update the attachment field in the queue record from the internal attachment list. The system does not clear the internal attachment list.

Use this function after you use the DSIAddAttachVar function to move your additions to the attachment list.

Syntax

```c
long DSIStoreAttachment(DSIHANDLE hInstance, long iQueue);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue in which the attachment is to be updated</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARAM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_MEMORY</td>
<td>out of memory</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

Errors

Invalid parameter

Uninitialized attachment list

See also

- DSIParseAttachment on page 144
- DSIAddAttachVar on page 84
- DSIAddAttachRec on page 83
- DSIAddToAttachRec on page 86
- DSISetQField on page 156

Example

```c
DSIStoreAttachment( hInstance, DSI_OUTPUTQUEUE );
```
**DSITerm**

Use this function to terminate DSI use. This should be called by the application only once.

**Syntax**

```c
long DSITerm(DSIHANDLE hApp);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hApp</td>
<td>handle to application data returned by a prior call to DSInit</td>
</tr>
</tbody>
</table>

**Return values**

DSIERR_SUCCESS

**Errors**

None

**See also**

DSInit on page 136

**Example**

```c
DSIHANDLE hApp;
DSIHANDLE hInstance;
hApp = DSInit();
hInstance = DSInitInstance( hApp );
DoSomeStuff( hInstance, andSomeOtherParameters );
DSITermInstance( hInstance );
DSITerm( hApp );
return( -10368 );
```
DSITermInstance

Use this function to terminate instance data.

Syntax

```c
long DSITermInstance(DSIHANDLE hInstance);
```

Parameters

Parameter | Description
-----------|------------------
hInstance | handle of instance data previously initialized by a call to DSInitInstance

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
</tbody>
</table>

Errors

Invalid parameter

See also

DSInitInstance on page 137

Example

```c
DSIHANDLE hApp = DSIni();
DSIHANDLE hInstance = DSInitInstance( hApp );
DoSomeStuff( hApp, SomeOtherParameters );
DSITermInstance( hInstance );
DSITerm( hApp );
return( 22 );
```
DSITermQueue

Use this function to terminate the usage of a queue.

**Syntax**

```c
long DSITermQueue(DSIHANDLE hInstance, long iQueue);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>handle to instance returned by DSIInitInstance</td>
</tr>
<tr>
<td>iQueue</td>
<td>queue to terminate</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIERR_SUCCESS</td>
<td>no error</td>
</tr>
<tr>
<td>DSIERR_INVPARAM</td>
<td>invalid parameter</td>
</tr>
<tr>
<td>DSIERR_UNKNOWN</td>
<td>unknown error</td>
</tr>
<tr>
<td>DSIERR_MEMORY</td>
<td>out of memory</td>
</tr>
<tr>
<td>DSIERR_EOF</td>
<td>record not found</td>
</tr>
<tr>
<td>DSIERR_QERR</td>
<td>uninitialized queue</td>
</tr>
<tr>
<td>DSIERR_IOERR</td>
<td>end of file</td>
</tr>
</tbody>
</table>

**Errors**

- Invalid parameter
- Lower level API failure—call DSIGetQError

**See also**

- DSIInitQueue on page 138

**Example**

```c
DSITermQueue( hInstance, DSI_INPUTQUEUE );
```
LDAPGetErrorCode

Use this function to return the last LDAP error code.

Returns
An integer value that represents the last LDAP error code.

Example
Here is an example:

```c
char *args = "ldap.host=localhost, ldap.port=389, ldap.timeout=5000";
char *file = "c:\docserv\openldap.properties";
char *userid = "demo1";
VMMHANDLE listH = VMMNULLHANDLE;
void *ldap = NULL;

if ((ldap = LDAPInit(args, file)) != NULL){

    listH = LDAPSearchDirectory(userid, ldap);
    if (listH == VMMNULLHANDLE || VMMCountList(listH) == 0){
        UTLLogTrace("LDAP Error Code: %d \n "
                    "LDAP Error Message: %s",
                    LDAPGetErrorCode(ldap),
                    LDAPGetErrorMessage(ldap));
        LDAPTerm(ldap);
    }
    LDAPTerm(ldap);
}
```

See also
LDAPGetErrorMessage on page 163
LDAPGetErrorMessage

Use this function to return the last error message.

Returns
A character pointer to the last LDAP error message.

Example
Here is an example:

```c
char *args = "ldap.host=localhost,ldap.port=389,ldap.timeout=5000";
char *file = "c:\docserv\openldap.properties";
char *userid = "demo1";
VMMHANDLE listH = VMMNULLHANDLE;
void *ldap = NULL;

if ((ldap = LDAPInit(args,
    file)) != NULL){

    listH = LDAPSearchDirectory(userid,
        ldap);
    if (listH == VMMNULLHANDLE ||
        VMMCountList(listH) == 0){
        UTLLLogTrace("LDAP Error Code: %d\n"
            "LDAP Error Message: %s",
            LDAPGetErrorCode(ldap),
            LDAPGetErrorMessage(ldap));
        LDAPTerm(ldap);
    }
    LDAPTerm(ldap);
}
```

See also
LDAPGetErrorCode on page 162
LDAPInit

Use this function to initialize and start an SSL or non-SSL connection to an LDAP server. This function reads the connection and search options from a comma-delimited list of arguments, a properties file, an INI file, or from input message variables/GVMs, in that order.

The options found in more than one location override the previous one. Option names are not case sensitive. This function supports option values encrypted through the cryrun program. Precede encrypted option values with the keyword ~ENCRYPTED and a space.

Be sure to call this function before calling the LDAPSearchDirectory function to set the connection and search options and to establish a connection session to an LDAP server.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDAP.HOST</td>
<td>(Optional) The host name or IP address of the LDAP server. The default is localhost.</td>
</tr>
<tr>
<td>LDAP.PORT</td>
<td>(Optional) The port in which the LDAP server is listening on. The default is 389 when SSL is not used, 636 otherwise (see the LDAP.USE.SSL option).</td>
</tr>
<tr>
<td>LDAP.URL</td>
<td>(Optional) The URL the LDAP server is listening on. If a value is specified for this property, it overrides the values specified for LDAP.HOST and LDAP.PORT.</td>
</tr>
<tr>
<td>LDAP.UID</td>
<td>(Optional) The user ID for logging onto the LDAP server. If this value is provided and LDAP.USER is not provided, the user ID is derived from this value and the value provided for LDAP.DOMAIN option, such as <a href="mailto:Administrator@pd.com">Administrator@pd.com</a>.</td>
</tr>
<tr>
<td>LDAP.USER</td>
<td>(Optional) An explicit value to use for the user ID for the purpose of login into the LDAP server. Define this option to override the behavior used to determine the user ID when LDAP.UID and LDAP.DOMAIN are defined - see LDAP.DOMAIN.</td>
</tr>
<tr>
<td>LDAP.PWD</td>
<td>(Optional) The password used to login into the LDAP server.</td>
</tr>
<tr>
<td>LDAP.AUTHENTICATION.MODE</td>
<td>(Optional) The method of authentication used to login into the LDAP server. Acceptable values are (simple) which provides clear-text password authentication and (none) which provides anonymous authentication. The default is (simple).</td>
</tr>
<tr>
<td>LDAP.TIMEOUT</td>
<td>(Optional) The amount of time (in milliseconds) after which a connection attempt or query should expire. The default is 10000 (10 seconds).</td>
</tr>
<tr>
<td>LDAP.SEARCH.BASE</td>
<td>(Optional) The base of the search in the DIT (Directory Information Tree). This is the starting point (node location) of a search in the DIT. If you omit this property, the system looks for the LDAP.DOMAIN option and builds a search base from it.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LDAP.DOMAIN</td>
<td>(Optional) This is the domain of the LDAP server. It is used to build the user ID for login into the LDAP server by appending the at symbol (@) plus the value of this option to the LDAP.UID value. The value of LDAP.DOMAIN is further parsed into domain components which are used as the default value for LDAP.SEARCH.BASE, if not already defined.</td>
</tr>
<tr>
<td>LDAP.OBJECTS</td>
<td>(Optional) A semicolon-delimited filter list of object classes to search in the LDAP server. If defined, it overrides the default filter list of object classes to search: group and groupOfNames.</td>
</tr>
</tbody>
</table>
| LDAP.OBJECTS.SEARCH.STRING     | (Optional) An explicit string value used as the filter of object classes to search. If defined, it overrides any value provided for LDAP.OBJECTS option. The value provided for this option must be specified in the appropriate LDAP protocol filter format. Also, if the search filter contains a question mark (?), the system replaces it with the user ID passed in as an argument to this function. Here are some examples:  
  
```
(&(objectClass=group)(objectClass=groupOfNames)).
Cn=*
```
| LDAP.OBJECT.ATTRIBUTES         | (Optional) The name of the attributes to retrieve for each object class which contain a value used to determine a match for USERID specified. The default values are member and cn (cn is always included).            |
| LDAP.MATCH.ATTRIBUTES          | (Optional) The name of one or more attributes contained within the value returned by a search for the LDAP.OBJECT.ATTRIBUTES option. This is the name of an attribute whose value is used to compare as opposed to the USERID specified to determine a match.  
For example, if LDAP.OBJECTS contains a value of groupOfUniqueNames and LDAP.OBJECT.ATTRIBUTES contains a value of uniqueMember and the value returned for the uniqueMember attribute of groupOfUniqueNames object class is uid=admin,ou=people,dc=mycompany,dc=com and you want to match the USERID value with the value for uid, you would supply a value of uid for this option. The default is cn. |
| LDAP.SEARCH.SCOPE              | (Optional) The scope of the search. Acceptable values are: (base) - search only the named context (one) - search one level below the named context but not the named context (sub) - search the entire subtree, including the named context. The default is (sub). |
| LDAP.DEREF.LINK                | (Optional) Enter Yes or No to indicate whether or not to remove reference links to other nodes during a search. The default is No. |
### Property Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
</table>
| LDAP.VERSION          | (Optional) An integer value that indicates the LDAP protocol version to use. You can choose from:  
2 - Version 2  
3 - Version 3  
The default is three (3). |
| LDAP.SEARCH.LEVEL     | (Optional) An integer value that indicates the search level. You can choose from:  
1 - User type objects  
2 - Group type objects  
3 - Any objects  
The default is one (1), user type objects. |
| LDAP.DN.IDENTIFIER    | (Optional) The value for this property is used in the following ways:  
1) In cases where LDAP.SEARCH.LEVEL is equal to 1 (USER) and there is no LDAP.OBJECTS.SEARCH.STRING value specified, the system generates a default search filter of the format identifier=userid, where identifier is the value of this property and userid is the user ID passed in as an argument to this function.  
2) In cases where LDAP.SEARCH.LEVEL is equal to 2 (GROUPS) and there is no LDAP.OBJECTS.SEARCH.STRING value specified, the system generates a default search filter from LDAP.OBJECTS and LDAP.OBJECT.ATTRIBUTES, where each attribute value in the search filter is an asterisk (*), which tells the system to match any value for the attributes specified. If the LDAP.RDNDS property is also provided, the asterisk (*) is replaced with identifier=userid, followed by a comma and the LDAP.RDNDS value to fine tune the search, where identifier is the value for this property and userid is the user ID passed in as an argument to this function. Here is an example of a default search filter:  
```
(&(objectClass=groupOfNames)(member=*))
```
If a value of 'CN=Users,DC=PDDC,DC=DOCUCORP,DC=COM' is specified for LDAP.RDNDS and this property contains a value of 'CN', the search filter generated would look like this:  
```
(&(objectClass=groupOfNames)(member=CN=Administrator, 
CN=Users,DC=PDDC,DC=DOCUCORP,DC=COM))
```
3) The default is 'CN'. |
Here is an example of a properties file:

ldap.host=localhost
ldap.port=389
ldap.timeout=5000
ldap.uid=cn=Administrator, dc=pdldap, dc=com
ldap.pwd=marks99
ldap.authentication.mode=simple
ldap.objects=groupOfNames;group
ldap.search.base=dc=pdldap, dc=com
ldap.object.attributes=member
ldap.match.attributes=cn
ldap.search.scope=sub
ldap.version=3
ldap.deref.link=Yes
ldap.debug=yes

Here is an example of an INI file:

```
[LDAP]
ldap.host=PDDC.pd.com
ldap.port=389
```
ldap.uid=jroberts
ldap.pwd=~ENCRYPTED 251U0jhIqWh8Gnr7o2Yq5A000
ldap.authentication.mode=simple
ldap.domain=PDDC.pd.com
ldap.objects=group
ldap.debug=yes
ldap.object.attributes=member
ldap.match.attributes=cn

Returns
An LDAP error code.

Example
Here is an example:

```c
char *args = "ldap.host=localhost,ldap.port=389,ldap.timeout=5000";
char *file = "c:\docserv\openldap.properties";
char *userid = "demo1";
VMMHANDLE listH = VMMNULLHANDLE;
void *ldap = NULL;

if ((ldap = LDAPInit(args,
                   file)) != NULL){

    listH = LDAPSearchDirectory(userid,
                                 ldap);
    if (listH == VMMNULLHANDLE ||
        VMMCountList(listH) == 0){
        UTLLogTrace("LDAP Error Code: %d\n" 
                     "LDAP Error Message: %s",
                     LDAPGetErrorCode(ldap),
                     LDAPGetErrorMessage(ldap));
        LDAPTerm(ldap);
    }

    LDAPTerm(ldap);
}
```

See also
LDAPTerm on page 170
LDAPSearchDirectory on page 169
LDAPSearchDirectory

Use this function to search a user ID for group or role membership in an LDAP server DIT (Directory Information Tree).

Call this function after the LDAPInit function, followed by the LDAPTerm function when the session is no longer needed. This function supports encrypted communications through an SSL channel (see the LDAP.USE.SSL property in the LDAPInit function) and encrypted option values.

Returns

A VMMHANDLE to a VMLList of string values corresponding to each group or role the user ID belongs to.

Example

Here is an example:

```c
char *args = "ldap.host=localhost,ldap.port=389,ldap.timeout=5000";
char *file = "c:\docserv\openldap.properties";
char *userid = "demo1";
VMMHANDLE listH = VMMNULLHANDLE;
void *ldap = NULL;

if ((ldap = LDAPInit(args,  
    file)) != NULL){

    listH = LDAPSearchDirectory(userid,  
        ldap);
    if ((listH == VMMNULLHANDLE ||  
        VMMCountList(listH) == 0){
        UTLLogTrace("LDAP Error Code: \%d\n"  
            "LDAP Error Message: \%s",  
            LDAPGetErrorCode(ldap),  
            LDAPGetErrorMessage(ldap));  
        LDAPTerm(ldap);
    }  

    LDAPTerm(ldap);
}
```

See also

LDAPTerm on page 170
LDAPInit on page 164
LDAPTerm

Use this function to terminate a connection to an LDAP server.

Example

Here is an example:

```c
char *args = "ldap.host=localhost,ldap.port=389,ldap.timeout=5000";
char *file = "c:\docserv\openldap.properties"
char *userid = "demo1"
VMMHANDLE listH = VMMNULLHANDLE;
void *ldap = NULL;

if ((ldap = LDAPInit(args,
        file)) != NULL){

        listH = LDAPSearchDirectory(userid,
              ldap);
        if (listH == VMMNULLHANDLE ||
            VMMCountList(listH) == 0){
            UTLLogTrace("LDAP Error Code: %d
" 
            "LDAP Error Message: %s",
               LDAPGetErrorCode(ldap),
               LDAPGetErrorMessage(ldap));
             LDAPTerm(ldap);
        }

        LDAPTerm(ldap);
}

See also LDAPInit on page 164
LDAPSearchDirectory on page 169
Chapter 3

DSI Java APIs

This chapter provides a reference to the Document Server Interface (DSI) Java APIs you can use to create applications to interface with Oracle Insurance's Internet Document Server.

This information will help you build either a proprietary client interface or a custom set of rules which will interact with the Internet Document Server.

The DSI Java API provides the DSI API. Since Java is an object-oriented language, the API is implemented as three classes:

- Class DSIJSession
- Class DSIJException
- Class DSIJQueue

These classes provide access to the Internet Document Server. All three classes are in a single package, com.Docucorp.DIS.util, which should be imported into any Java source file.

**NOTE:** The DSI API includes multiple interfaces (APIs). This lets you choose the language to build custom rules and applications. You will also find sample clients written in each language, which serve as a reference when building your own solution.

The topic, Java API Classes on page 182 provides a list of all Java methods, grouped by class. Each method is then discussed in alphabetical order, by class.

You will also find information on using the included JavaBean component in the topic, Using JavaBean Components on page 172.
USING JAVA BEAN COMPONENTS

com.docucorp.ids.jsp.dsi is a JavaBean component which lets you create an interface between Java server pages (JSPs) and IDS rules.

The request name/value string from the browser is passed to JavaBean using these methods:

- AddRequest(Object name, Object value)
- AddAllRequest(javax.servlet.ServletRequest request)

AddRequest adds one request name/value at a time. AddAllRequest adds all name/values from the http request object.

This illustration shows how it works:
The name *REQTYPE* is reserved for the request type to the IDS rule. Once the request name/value has been passed to the dsi JavaBean, ProcessRequest is called to send the name/value and request type to the IDS rules.

After the IDS rule is processed, ProcessRequest returns the name/value records from the IDS rules as a HashMap object. `setWaittime()` sets the retry time to read the return records from IDS. `setTimeout()` sets the timeout period to read the return records from IDS.

`debug_on(javax.servlet.ServletResponse response)` sets a flag to send the request name/value and return name/value from IDS to the passing response object and then calls the AddRequest and ProcessRequest methods.
Here is a summary of how the methods work:

<table>
<thead>
<tr>
<th>Step</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>void debug_on(javax.servlet.ServletResponse response)</td>
<td>Sets the flag to send the request name/value and return name/value from IDS to the passing response object. Then calls the AddRequest and ProcessRequest methods.</td>
</tr>
<tr>
<td>2</td>
<td>void debug_off()</td>
<td>Clears the debug flag.</td>
</tr>
<tr>
<td>3</td>
<td>int getWaittime()</td>
<td>Gets the amount of retry time to read the IDS return record.</td>
</tr>
<tr>
<td>4</td>
<td>int setWaittime(int waittime)</td>
<td>Sets the retry time (in milliseconds) to read the IDS return record.</td>
</tr>
<tr>
<td>5</td>
<td>int getTimeout()</td>
<td>Gets the timeout to read the IDS return record.</td>
</tr>
<tr>
<td>6</td>
<td>int setTimeout(int timeout)</td>
<td>Sets the timeout (in milliseconds) to read the IDS return record.</td>
</tr>
<tr>
<td>7</td>
<td>void AddRequest(Object key, Object value)</td>
<td>Adds the name/value field to the record to send to the IDS rule.</td>
</tr>
<tr>
<td>8</td>
<td>void AddAllRequest(javax.servlet.ServletResponse request)</td>
<td>Adds all name/value fields from the request object to the records to send to the IDS rule.</td>
</tr>
<tr>
<td>9</td>
<td>HashMap ProcessRequest()</td>
<td>Sends all name/values and request types to the IDS rules. Processes the IDS rule and gets the return records from the IDS rule. Returns the record as type HashMap.</td>
</tr>
<tr>
<td>10</td>
<td>String GetResult(Object key)</td>
<td>Gets the return record value from the IDS rule index using the key from internal result.</td>
</tr>
<tr>
<td>11</td>
<td>void ClearRequest()</td>
<td>Clears the JavaBean internal request object.</td>
</tr>
<tr>
<td>12</td>
<td>void ClearResult()</td>
<td>Clears the JavaBean internal result object.</td>
</tr>
</tbody>
</table>
echotest.jsp

Here is an example:

```html
<html>
<!--
 Copyright (c) 2001 Docucorp International. All rights reserved.
 -->

<%@ page language="java"%>
<body bgcolor="white">
<jsp:useBean id='dsi' scope='page' class='com.docucorp.ids.jsp.dsi'/>
<%
    dsi.setTimeout(20000);    //Set Timeout
%>
<font size=4>
<ul>
<li>WaitTime: is  <jsp:getProperty name="dsi" property="waittime"/>
<li>TimeOut: is  <jsp:getProperty name="dsi" property="timeout"/>
</ul>
</font>
</body>
</html>
```

This JSP calls an echo rule in IDS and pass two name/value pairs.
RETURNING A RECORDSET OBJECT

The processRequest method in dsimsg class returns a user-defined RecordSet object for requests that execute SQL queries through the SQLQueryDB rule. The RecordSet object is built from the output message XML rowsets: RECORDS and SELECTIONFIELDS.

Use this capability with the SQLQueryDB rule, which adds the rowsets RECORDS and SELECTIONFIELDS to the result message. This lets you process queries with dsimsg class instead of using idssql package — and a RecordSet object can still be returned. The RecordSet object is identical to the idsrs object in the idssql package, so all method definitions and calls are the same.

Here is a sample JSP page:

```jsp
<%@ page language="java" import="java.util.*, java.net.*, java.io.*" %>

<jsp:useBean id='dsi' scope='page' class='com.docucorp.ids.jsp.dsimsg'/>
<jsp:useBean id='rs' scope='page' class='com.docucorp.ids.jsp.RecordSet'/>

<%

/***always call at the beginning of a jsp page
***when calling processRequest more than
***once with the same dsimsg bean instance.*/
dsi.initInstance();

for (int x = 0; x < 20; x++){

dsi.setTimeOut(30000);
//dsi.debugOn(response);

dsi.addRequest("REQTYPE", "TEST3");
dsi.addRequest("USERID", "FORMAKER");
dsi.addRequest("PASSWORD", "FORMAKER");
dsi.addRequest("PROCNAME", "YYZ ");
dsi.addRequest("INSTANCE", String.valueOf(x));

String record = "SQLPARAMETERS";
String rec = dsi.addAttachRec(record);
if (rec != null){

dsi.addToAttachRec(rec, "PARAM1", "PASSWORD");
dsi.addToAttachRec(rec, "PARAM2", "USERID ");
dsi.addToAttachRec(rec, "PARAM3", "SERVERTIMESPENT");
dsi.addToAttachRec(rec, "PARAM4", "TRANLOG20030602");
dsi.addToAttachRec(rec, "PARAM5", "FORMAKER");
dsi.addToAttachRec(rec, "PARAM6", "FORMAKER");
}

rs = dsi.processRequest();
```
Using JavaBean Components

if (rs == null){
out.println("rs == null");
}
else{
out.println("<BR><b>INSTANCE:" + String.valueOf(x) + "</b><BR>");
for(int i=1; i<= rs.getRecordCount();i++){
out.println("===========" + "<BR>");
out.println("RECORD " + i + ":" + "<BR>");
out.println("===========" + "<BR>");
for (int j=1;j<= rs.getColumnCount();j++){
out.println(rs.getColumn(j) + ":" + rs.getString(j) + "<BR>");
}
rs.next();
}
}
/***always call in between requests to reset / clear the messages in
the
***queues.
*/
dsi.resetInstance();
}
/***always call at the end of a jsp page
***when calling processRequest more than
***once with the same dsimsg bean instance.
*/
dsi.termInstance();

%>

177


USING IDSJSP IN A JSP CONTAINER

Here is an example JSP page that uses IDSJSP to send an SSS request type using the message bus properties in the dsimsgclient.properties file:

```jsp
<%@ page language="java" import="java.util.*, java.net.*, java.io.*" %>
<%@ page language="java" import="java.util.*, java.net.*, java.io.*" %>
<jsp:useBean id='dsi' scope='page' class='com.docucorp.ids.jsp.dsi'/> 
<%
    dsi.setTimeout(30000);
    dsi.debugOn(response);
    dsi.AddRequest("REQTYPE", "SSS");
    HashMap Rst = dsi.ProcessRequest();
    if (Rst.get("RESULTS") == null){
        out.println("No response from server");
    }
%>
```

Alternatively, you can specify the properties in the JSP page, in which case the dsimsgclient.properties file is not needed. Here is an example JSP page that uses the HTTP message bus properties to send an SSS request type to IDS:

```jsp
<%@ page language="java" import="java.util.*, java.net.*, java.io.*" %>
<%@ page language="java" import="java.util.*, java.net.*, java.io.*" %>
<jsp:useBean id='dsi' scope='page' class='com.docucorp.ids.jsp.dsi'/> 
<%
    Properties props = new Properties();
    props.put("queuefactory.class", "com.docucorp.messaging.http.DSIHTTPMessageQueueFactory");
    props.put("marshaller.class", "com.docucorp.messaging.data.marshaller.SOAPMIMEDSIMessageMarshaller");
    props.put("http.url", "http://localhost:49152");
    dsi.debugOn(response);
    dsi.AddRequest("REQTYPE", "SSS");
    HashMap Rst = dsi.ProcessRequest(props);
    if (Rst.get("RESULTS") == null){
        out.println("No response from server");
    }
%>
```

DSI BEAN APIs

Please refer to the docs/com/docucorp/ids/jsp/dsi.html documentation that is shipped with the Java SDK for a description of the methods available in the dsi bean.
If you are deploying a Java client application you can use the DSI Java messaging library, DSIJavaMsg.jar. This library provides the same functionality as the DSI Java APIs but uses only Java code. The DSI Java APIs use native code related to the DSI C APIs.

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

By using only Java code, the DSI Java messaging library lets you have Java client applications wherever you have a Java runtime so you do not need to port Document Server Interface code to your target platform.

The DSI Java messaging library only works with IBM's MQSeries as the messaging service. It cannot be used with Java rules for the Internet Document Server.

**NOTE:** If you are running the DSI Java Messaging Library inside a Java 2 Enterprise Edition (J2EE) Application Server, such as IBM's WebSphere or BEA's WebLogic, the JavaMail API and Javabeans Activation Framework are already installed as a part of the application server.

The DSI Java messaging library also requires XML processing libraries from the Apache group, xerces.jar and xalan.jar. These libraries are included. Copy these libraries into the same directory as DocucorpMsg.jar.
PASSING JVM OPTIONS TO DSILIB

DSILIB uses Java through JNI (Java Native Interface) and as such it creates a Java Virtual Machine (JVM) at runtime. DSILIB lets you pass JVM options before the JVM is created, so you can fine-tune what is created.

For instance, you can specify the size of memory for the JVM. This is helpful, for example, if you need to set memory higher to handle large files transmitted via the message bus (queue).

To pass JVM options, use the `dsi_extended_properties` environment variable. This environment variable should contain a comma-delimited list of additional JVM options to pass during creation of a JVM.

Here is an example of how you would set the environment variable from a command prompt:

Windows

```
set dsi_extended_properties=-Xmx256m,-
    Dlog4j.configuration=logclientconf.xml
```

UNIX

```
export dsi_extended_properties=-Xmx256m,-
    Dlog4j.configuration=logclientconf.xml
```

Examples of client-based applications that use DSILIB include:

- ASP pages using IDSASP.DLL
- JSP pages using IDSJSP.jar
- DSIJava.jar files, which use the C code (DSILIB)
- The DSICOTB.EXE, DSITEST.EXE, and DSIEX.EXE test programs
IDS supports the following log4j categories and appenders which you can use in a log4j client configuration file to produce debugging output for client requests:

```xml
<category name="Receive-Message">
    <priority value="DEBUG"/>
    <appender-ref ref="receive-message"/>
</category>

<category name="Send-Message">
    <priority value="DEBUG"/>
    <appender-ref ref="send-message"/>
</category>

<appender class="com.docucorp.util.logging.IDSFileAppender"
    name="receive-message">
    <param value="false" name="Append"/>
    <param value="client-receive.msg" name="File"/>
    <param value="true" name="Close"/>
    <param value="ISO-8859-1" name="Encoding"/>
    <layout class="org.apache.log4j.PatternLayout">
        <param value="%m" name="ConversionPattern"/>
    </layout>
</appender>

<appender class="com.docucorp.util.logging.IDSFileAppender"
    name="send-message">
    <param value="false" name="Append"/>
    <param value="client-send.msg" name="File"/>
    <param value="true" name="Close"/>
    <param value="ISO-8859-1" name="Encoding"/>
    <layout class="org.apache.log4j.PatternLayout">
        <param value="%m" name="ConversionPattern"/>
    </layout>
</appender>
```

**NOTE:** See the logclientconf.xml file for an example.
Here are the methods you can use with Java, grouped into these classes:

- **DSIJession**
  Refer to the dsidocs/com/Docucorp/DSI/util/DSIJession.html documentation shipped with the Java SDK for a description of the methods that are available.

- **DSIJQueue**
  Refer to the dsidocs/com/Docucorp/DSI/util/DSIJession.html documentation shipped with the Java SDK for a description of the methods that are available.

- **DSIJException**
  Refer to the dsidocs/com/Docucorp/DSI/util/DSIJession.html documentation shipped with the Java SDK for a description of the methods that are available.
Chapter 4

DSI Processing Rules

The Internet Document Server includes processing rules you can use to control what happens to data. These rules are divided into the following groups and explained in this chapter.

• Server Rules on page 184
• Client Rules on page 225

Within each group, the rules are listed in alphabetical order.

These rules run on all supported platforms except where noted.

NOTE: The rule names are case sensitive.
Server Rules

These rules may only be run on the Internet Document Server.

With version 2.0, the built-in server rules in IDS were replaced with Java rules. When IDS finds a mention of an IDS 1.x server rule, it is automatically replaced with the corresponding IDS Java rule.

Here is a list of the IDS 1.x rules that have Java substitutes. All Java classes mentioned are in the com.docucorp.ids.rules package.

<table>
<thead>
<tr>
<th>Version 1.x rule</th>
<th>Version 2.x rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATCSendFile on page 233</td>
<td>AttachmentFilerRule.sendFile on page 209.</td>
</tr>
<tr>
<td>ATCReceiveFile on page 230</td>
<td>AttachmentFilerRule.receiveFile on page 209.</td>
</tr>
<tr>
<td>ATCLogTransaction on page 228</td>
<td>LogTransactionRule.logTransaction on page 223.</td>
</tr>
<tr>
<td>ATCUUnloadAttachment on page 236</td>
<td>IDSTransactionRule.reportTimes on page 222.</td>
</tr>
<tr>
<td>IRLInitFTP on page 202</td>
<td>FTPRule.setupConnections on page 214.</td>
</tr>
<tr>
<td>IRLFileFTP on page 195</td>
<td>FTPRule.transferFiles on page 214.</td>
</tr>
<tr>
<td>IRLCleanDirectory on page 189</td>
<td></td>
</tr>
<tr>
<td>IRLCopyAttachment on page 192</td>
<td>CopyDataRule.copyData on page 213.</td>
</tr>
<tr>
<td>IRLInit on page 194</td>
<td>IDSInitRule.init on page 221.</td>
</tr>
<tr>
<td>IRLLog on page 203</td>
<td></td>
</tr>
<tr>
<td>IRLPurgeCache on page 204</td>
<td>BLPPurgeRule.purge on page 211.</td>
</tr>
<tr>
<td>IRLSearch on page 205</td>
<td></td>
</tr>
<tr>
<td>IRLSendVersion on page 206</td>
<td></td>
</tr>
<tr>
<td>IRLStatistics on page 208</td>
<td>BLPStatisticsRule.addStatistics on page 212.</td>
</tr>
<tr>
<td>IRLDecryptValue on page 193</td>
<td>IDSEncryptionRule.cryptVariables on page 220.</td>
</tr>
</tbody>
</table>

Note: Both the old and new rules are discussed in this chapter. In future releases, documentation on the old rules will be removed.

You can run these rules in IDS:

- AttachmentFilerRule
- BLPPurgeRule
- BLPStatisticsRule
- CopyDataRule
- FTPRule
- IDSEncryptionRule
- IDSInitRule
- IDSTransactionRule
- LogTransactionRule
FTPRule

Use this rule to handle FTP file transfers. This rule is a Java class that implements an IDS rule for this purpose. The FTPRule rule is a server rule which runs on both Windows and Solaris, as opposed to the IRLInitFTP and IRLFileFTP rules which run only on Windows.

Because the FTPRule rule tracks all FTP connections made across transactions, you should run it using global scope.

There are two methods in FTPRule you must use:

- setupMethod
- transferMethod

setupMethod

Use this method in the INI request type. This method creates the data needed to run multiple FTP transfers in the DSI_MSGINIT message and destroys the data in the DSI_MSGTERM message.

Add these lines into your INI request group:

```
function = dsijrule->JavaInitRule
function = dsijrule->JavaRunRule,;com/docucorp/ids/rules/
FTPRule;JAVAFTP;global;setupMethod;
```

Like all Java rules, the FTPRule rule requires that JavaInitRule be run first in the INI request group. In the second function description, you have these parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com/docucorp/ids/rules/FTPRule</td>
<td>Identifies the FTPRule class with full package naming required for JNI loading.</td>
</tr>
<tr>
<td>JAVAFTP</td>
<td>An example name for a named object with global scope; any name would suffice here.</td>
</tr>
<tr>
<td>global</td>
<td>Indicates that JavaRunRule will create an object with global scope and that can be used in other transactions.</td>
</tr>
</tbody>
</table>

In the JavaRule control group in DOCSERV configuration file, make sure the following Java Archive (JAR) files are in your class path via the UserClassPath option:

- DSIIJava.jar
- NetComponents.jar
- DocucorpUtil.jar
- IDSRules.jar
Use this method in your transaction control group to do the actual file transfer via FTP. It gets files from the FTP server in the DSI_MSGRUNF message and puts them onto the FTP server in the DSI_MSGRUNR message.

Add these lines into your transaction’s request group:

```
function = irlw32->IRLJavaFTPSetup
function = dsijrule->JavaRunRule,;com/docucorp/ids/rules/FTPRule;JAVAFTP;global;transferMethod;
function = dsijrule->JavaRunRule,;com/docucorp/ids/rules/FTPRule;JAVAFTP;global;transferMethod;,FTPUTLOCALFILE->FTPRRC2FILE
```

The IRLJavaFTPSetup rule must be run before JavaRunRule with FTPRule. IRLJavaFTPSetup reads the INI settings for the IRLFileFTP rule and creates attachment variables that can be understood by FTPRule. For more information on which parameters, attachment variables and INI options to use with the FTPRule rule, see IRLFileFTP on page 195.

In addition to the options for IRLFileFTP, you can use the JavaLogFileName option in the FTP control group to specify a file for logging FTPRule’s debugging messages when the Debug option is set to Yes. If you omit this option, the system uses the name, FTPRULE.LOG.

### PUTTING AND GETTING MULTIPLE FILES

Before version 2.1, FTPRule used a message variable to hold the name of a file to get or put, such as `GETFILEREMOTE`. In version 2.1 and later, if the message variable listed ends with an asterisk (*), IDS scans all message variables for variables that begin with that name. For example, if you set up FTPRule with these parameters:

```
<entry name="function">irlw32->IRLFileFTP,GETFILEREMOTE*-GETFILELOCAL*,</entry>
```

IDS matches the message variables `GETFILEREMOTEA`, `GETFILEREMOTEB`, `GETFILEREMOTEC`, and so on.

When a match is found on the first parameter, IDS looks for a corresponding match on the second parameter with the same suffix. For example, for `GETFILEREMOTEA`, `GETFILEREMOTE` is the matching prefix and `A` is the suffix, so IDS will look for a message variable named `GETFILELOCALA`.  

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com/docucorp/ids/rules/FTPRule</td>
<td>Identifies the FTPRule class with full package naming required for JNI loading.</td>
</tr>
<tr>
<td>JAVAFTP</td>
<td>An example name for a named object with global scope; use the same name for the object that you used with setupMethod.</td>
</tr>
<tr>
<td>global</td>
<td>Indicates that this rule is using an object with global scope, the same object used when running setupMethod.</td>
</tr>
<tr>
<td>transferMethod</td>
<td>The method in the FTPRule class that does the actual file transfers. The argument after the method name follows the same convention as the arguments for the IRLFileFTP rule. For more information, see IRLFileFTP on page 195.</td>
</tr>
</tbody>
</table>
Assuming all the message variables are there, this would be the same as running the FTPRule three times, as shown here:

GETFILEREMOTEA -> GETFILELOCALA
GETFILEREMOTEB -> GETFILELOCALB
GETFILEREMOTEC -> GETFILELOCALC

This also works when you are putting files. Here is an example:

<entry name="function">irlw32->IRLFileFTP,,PUTFILELOCAL*->PUTFILEREMOTE*</entry>

This would be the same as (with the message variables set up):

PUTFILELOCALA -> PUTFILEREMOTEA
PUTFILELOCALB -> PUTFILEREMOTEB
PUTFILELOCALC -> PUTFILEREMOTEC

If a variable for a second parameter is missing, a unique name is generated and stored in that variable, as happened previously.

The FTPRule now also reports its own results in the output, separate from the RESULTS variable. If FTPRule is getting files from a remote FTP site, the results are placed in the FTPGETRESULTS variable; for putting to a remote site, the results are placed in the FTPPUTRESULTS variable. The variable will have either SUCCESS or ERROR. Error messages in the output can be checked for specific errors. For multiple file rule setups, all files must be successfully gotten or put to be reported as SUCCESS.
**IRLCleanDirectory**

Use the IRLCleanDirectory rule to remove expired files from a directory. To determine if a file has expired, the operating system’s local time is compared against a file's last modified time plus the expiration time supplied.

**Syntax**

```c
long _DSIAPI IRLCleanDirectory ( DSIHANDLE hInstance,
    char * pszParms,
    unsigned long  ulMsg,
    unsigned long  ulOptions )
```

**Attachment inputs**

The input attachment variables for this rule are:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIR</td>
<td>(Optional) The name of the directory you want cleaned up. If this attachment variable is present, it overrides any value specified as a rule argument. If a DIR value is omitted as an attachment variable or as a rule argument, the rule sets the RESULTS output attachment variable with a value of FAILURE and then exits.</td>
</tr>
<tr>
<td>EXPTIME</td>
<td>(Optional) The expiration time in minutes after which files should be removed. If this attachment variable is present, it overrides any value specified as a rule argument. If an EXPTIME value is omitted as an attachment variable or rule argument, the rule sets the RESULTS output attachment variable with a value of FAILURE and then exits.</td>
</tr>
<tr>
<td>DEBUG</td>
<td>(Optional) Enter Yes if you want the rule to output debug information. If this attachment variable is present, it overrides any value specified as a rule argument.</td>
</tr>
</tbody>
</table>

**Attachment outputs**

The output message variables are:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESULTS</td>
<td>Contains SUCCESS or FAILURE.</td>
</tr>
</tbody>
</table>

**Parameters**

The rule parameters are:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIR</td>
<td>(Optional) The name of the directory you want to clean up. If a DIR value is neither specified as a rule argument nor present as an attachment variable, the rule sets the RESULTS output attachment variable with a value of FAILURE and then exits.</td>
</tr>
<tr>
<td>EXPTIME</td>
<td>(Optional) The expiration time in minutes after which files should be removed. If an EXPTIME value is neither specified as a rule argument nor present as an attachment variable, the rule sets the RESULTS output attachment variable with a value of FAILURE and then exits.</td>
</tr>
<tr>
<td>DEBUG</td>
<td>(Optional) Enter Yes if you want the rule to output debug information.</td>
</tr>
</tbody>
</table>
Errors

You may encounter these errors:

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRV0004</td>
<td>The specified directory does not exist.</td>
</tr>
<tr>
<td>SRV0005</td>
<td>There is an invalid value for a rule parameter.</td>
</tr>
</tbody>
</table>

Example

Here is an example of a request type:

```
<section name="ReqType:TEST_REMOVE">
  <entry name="function">atcw32->ATCLoadAttachment</entry>
  <entry name="function">atcw32->ATCUnloadAttachment</entry>
  ;IRLCleanDirectory,DIR=c:\temp,EXPTIME=10,DEBUG=T</section>
```
IRLClearLog

Use this rule to remove all records from the server access log or error log files.

Syntax

```c
long _DSI_API IRLClearLog ( DSIHANDLE hInstance,
    char * pszParms,
    unsigned long ulMsg,
    unsigned long ulOptions )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIHANDLE hInstance</td>
<td>DSI instance handle</td>
</tr>
<tr>
<td>char * pszParms</td>
<td>Pointer to rule parameter string</td>
</tr>
<tr>
<td>unsigned long ulMsg</td>
<td>DSI_MSG???? message, such as DSI_MSGRUNF</td>
</tr>
<tr>
<td>unsigned long ulOptions</td>
<td>Options</td>
</tr>
</tbody>
</table>

The default DOCSEerv configuration file sets this rule with these INI settings.

```ini
< ReqType:CLF >
    Function = irlw32->IRLClearLog
```

Returns

Success or failure

Errors

You may encounter these errors:

- Missing necessary attachment variable
- Failed to locate records
- Failed to append to attachment
IRLCopyAttachment

Use this rule to copy attachment variables from the input queue to the output queue on the DSI_MSGRUNR message.

Syntax

```c
long _DSI_API IRLCopyAttachment ( DSIHANDLE hInstance,
        char * pszParms,
        unsigned long  ulMsg,
        unsigned long  ulOptions )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIHANDLE hInstance</td>
<td>DSI instance handle</td>
</tr>
<tr>
<td>char * pszParms</td>
<td>Pointer to rule parameter string</td>
</tr>
<tr>
<td>unsigned long ulMsg</td>
<td>DSI_MSGRUNR message, such as DSI_MSGRUNF</td>
</tr>
<tr>
<td>unsigned long ulOptions</td>
<td>Options</td>
</tr>
</tbody>
</table>

Returns

Success or failure

Errors

You may encounter these errors:

- Missing a necessary attachment variable
- Missing a necessary configuration file value
- Failed to write out configuration file
IRLDecryptValue

Use this rule to encrypt the attachment variables for use in the web browser and decrypt them back for IDS on the next request.

For example, on initial login request you can use this rule to encrypt the POLICYNUM in the output attachment. On the subsequent requests this rule will decrypt the POLICYNUM value in the input attachment so any other IDS rule that needs this value will be able to access it.

On the client side, POLICYNUM will be encrypted and not easy to change to point to some other policy in archive. If the system cannot locate the attachment variable, or if the encryption process fails, processing continues and no error is generated.

Syntax

long _DSIAPI IRLDecryptValue ( DSIHANDLE hInstance,
                      char * pszParms,
                      unsigned long  ulMsg,
                      unsigned long  ulOptions )

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIHANDLE hInstance</td>
<td>DSI instance handle</td>
</tr>
<tr>
<td>char * pszParms</td>
<td>Pointer to rule parameter string</td>
</tr>
<tr>
<td>unsigned long ulMsg</td>
<td>DSI_MSG???? message, such as DSI_MSGRUNF</td>
</tr>
<tr>
<td>unsigned long ulOptions</td>
<td>Options</td>
</tr>
</tbody>
</table>

The system supports wild cards, such as

abc*xyz, *xyz, or abc*

This rule works with attachment variables in a case insensitive manner.
IRLInit

Use this rule to initialize the server file cache and access log tables on the DSI_MSGINIT message. This rule also terminates them on the DSI_MSGTERM message. This rule is used on the REQTYPE INI, which means it has to run every time you start the server.

Syntax

```c
long _DSIAPI IRLInit ( DSIHANDLE hInstance,
                        char * pszParms,
                        unsigned long  ulMsg,
                        unsigned long  ulOptions )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIHANDLE hInstance</td>
<td>DSI instance handle</td>
</tr>
<tr>
<td>char * pszParms</td>
<td>Pointer to rule parameter string</td>
</tr>
<tr>
<td>unsigned long ulMsg</td>
<td>DSI_MSG???? message, such as DSI_MSGRUNF</td>
</tr>
<tr>
<td>unsigned long ulOptions</td>
<td>Options</td>
</tr>
</tbody>
</table>

Uses the following INI values to specify the locations (full file name) of the server cache and access log tables.

```ini
< DocSrvr >
CacheTbl = SRVCACHE
LogTable = SRVLOG
```

The default DOCSERV configuration file sets this rule with these INI settings.

```ini
< ReqType:INI >
Function = irlw32->IRLInit
```

Returns

Success or failure

Errors

You may encounter these errors:

- Failed to initialize cache
- Failed to initialize log
IRLFileFTP

Use this rule to get a file from the remote FTP server on the DSI_MSGRUNF and put another file back on the DSI_MSGRUNR.

**NOTE:** To use the IRLFileFTP rule, you must first run the IRLInitFTP rule. Be sure to place the IRLInitFTP rule on the INI rules list to run it.

**Syntax**

```c
long _DSI_API IRLFileFTP ( DSIHANDLE hInstance,
                           char * pszParms,
                           unsigned long ulMsg,
                           unsigned long ulOptions )
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIHANDLE hInstance</td>
<td>DSI instance handle</td>
</tr>
<tr>
<td>char * pszParms</td>
<td>Pointer to rule parameter string</td>
</tr>
<tr>
<td>unsigned long ulMsg</td>
<td>DSI_MSG???? message, such as DSI_MSGRUNF</td>
</tr>
<tr>
<td>unsigned long ulOptions</td>
<td>Options</td>
</tr>
</tbody>
</table>

You must register this rule using an INI request. Here is an example:

```
< ReqType:INI >
  function = irlw32->IRLInitFTP
< ReqType:FTPTest >
  function = irlw32->IRLFileFTP,GetFileRemote->GetFileLocal,
             PutFileLocal ->PutFileRemote
```

The following rule arguments are used in the following way:

- GetFileRemote and GetFileLocal rule arguments are used to look up the path and file name of the remote and local files for the GET operation. They are looked up in the following manner:
  
  Look first in the input attachment and if not found look in the output attachment.
  
  The rule argument names are just a representation and could be any other user-defined names, but there must be matching names in the input or output attachment.

- PutFileLocal and PutFileRemote rule arguments are used to look up the path and file name of the local and remote put files for the PUT operation. They are looked up in the following manner:
  
  Look first in the output attachment and if not found look in the input attachment.
  
  The rule argument names are just a representation and could be any other user-defined names, but there must be matching names in the input or output attachment.
### DSI Processing Rules

#### Chapter 4

If the `Get` names are missing, no FTP `Get` action is performed by this rule and no error message is generated. If the `Put` names are missing, no FTP `Put` action is performed by this rule and no error message is generated.

You can register multiple IRLFileFTP rules on the same request type if you need to FTP multiple files. This rule maintains the list of open FTP connections and reuses connections when possible.

Here is an example:

To transfer a file named FILE.DAT from the incoming directory on the FTP server to the d:/temp directory and rename it to MYFILE.DAT on the IDS server, you could set up the IRLFileFTP rule on a rules list in the DOCSERV configuration file as follows:

```plaintext
Function = irlw32::IRLFileFTP,GetRem->GetLoc,
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetFileRemote</td>
<td>The name of the attachment variable which contains the name of the file to get via FTP from the FTP server. This name is not a URL, it is the name of a file and, optionally, an FTP directory name. For instance, for <code>ftp://servername/incoming/file.dat</code> you would enter the name <code>incoming/file.dat</code>.</td>
</tr>
<tr>
<td>GetFileLocal</td>
<td>The name of the attachment variable which contains the name of the destination file (to be written locally to the IDS machine). If this value is not found, the rule generates a unique name and sets the value of the variable to the generated name. See FTPGetFilePath, below, for information on how to prefix this name with a path. The generated name is a long file name, so your file system has to support long file names. If the file exists when the GET operation is executed, it is overwritten. If the GET operation is successful and a unique file name is generated, the file name is added an output attachment variable.</td>
</tr>
<tr>
<td>PutFileLocal</td>
<td>The name of the attachment variable which contains the name of the local (to IDS) source file to be put via FTP onto the FTP server.</td>
</tr>
<tr>
<td>PutFileRemote</td>
<td>The name of the attachment variable which contains the name under which the destination file is to be written to the FTP server. If you supply this variable, bear in mind that the name it holds is not a URL, it is the name of a file and, optionally, an FTP directory name. For instance, for <code>ftp://servername/incoming/file.dat</code> you would enter <code>incoming/file.dat</code>. If this value is not found, this rule generates a unique name and sets the value of the variable to the generated name. See FTPPutFilePath, below, for information on how to prefix this name with an FTP directory. The generated name is a long file name, so your file system has to support long file names. If the file exists when the PUT operation is executed, the file will be overwritten. If the PUT operation is successful and a unique file name is generated, the file name is added an output attachment variable.</td>
</tr>
</tbody>
</table>
In this case, you put two variables on the input attachment: one named `GETREM` with the value `/INCOMING/FILE.DAT`, and one named `GETLOC` with the value `/d:/temp/ MYFILE.DAT`. Notice that parameters for putting a file are omitted, so no PUT operation occurs for this call to the IRLFileFTP rule.

Here is another example:

To transfer a file named `FILE.DAT` from the `d:/temp` directory on the IDS server, and let the IRLFileFTP rule generate the name under which it will be written to the FTP server, you could set up the IRLFileFTP rule on a rules list in the DOCSERV configuration file as follows:

```
Function = irlw32->IRLFileFTP,,PutLoc->PutRem,
```

In this case, you would put one variable named `PUTLOC` with the value `/d:/temp/ FILE.DAT` on the output attachment. You would not create a variable named `PUTREM`. The IRLFileFTP rule would automatically generate a file name, write the file to the FTP server using that name, create a variable named `PutRem` on the output attachment, and put the generated file name into the variable. Notice that since the parameters for getting a file were omitted, no GET operation occurs for this call to the IRLFileFTP rule.

**NOTE:** Keep in mind the FTP directories do not have drive letters.

If a connection is dropped, this rule reopens it. The default timeout value on an FTP server is 900 seconds, so the connection will stay open for at least this amount of time before it is dropped.

### Input options

These options are looked up in the following manner:

**GET OPERATIONS.** Look for each option in the input attachment and then in the output attachment using the value `FTP` value prefixed to the option name, such as `FTPDEBUG`. Then look for the options in the FTP:ReqType control group, where ReqType is the value of the REQTYPE input attachment variable and in the FTP control group. Each search occurs in the order listed and stops when an option is found. GET operations do not look up or use the RemoveOnPut or PutFilePath options.

**PUT OPERATIONS.** Look for each option in the output attachment and then in the input attachment using the value `FTP` value prefixed to the option name, such as `FTPDEBUG`. Then look for the options in the FTP:ReqType control group, where ReqType is the value of the REQTYPE input attachment variable and in the FTP control group. Each search occurs in the order listed and stops when an option is found. PUT operations do not look up or use the RemoveOnGet, GetFilePath, or CacheGetFile options.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>The server name or IP address for the FTP connection.</td>
</tr>
<tr>
<td>UserID</td>
<td>The user ID for the FTP connection.</td>
</tr>
<tr>
<td>Password</td>
<td>The password for the FTP connection.</td>
</tr>
<tr>
<td>Port</td>
<td>The server port for the FTP connection.</td>
</tr>
</tbody>
</table>
Here is an example of the INI options:

```
< FTP:ReqType>
    Server =
    UserID =
    Password =
    Port =
    GetFilePath =
    PutFilePath =
< FTP >
    Server =
    UserID =
    Password =
    Port =
    GetFilePath =
    PutFilePath =
    RemoveOnGet =
    RemoveOnPut =
    Debug =
    CacheGetFile =
< Attachment >
    Path =
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetFilePath</td>
<td>The path to be prefixed to the unique name IRLFileFTP generates when the variable for GetFileLocal is not found as an attachment variable. For example, <code>d:\temp</code> causes local names such as <code>d:\temp\0abcdefg.ext</code> to be generated.</td>
</tr>
<tr>
<td>PutFilePath</td>
<td>The FTP directory path (omit the drive specifier) to be prefixed to the unique name IRLFileFTP generates when the variable for PutFileRemote is not found as an attachment variable. For example, <code>incoming\datafiles</code> causes FTP names such as <code>incoming\datafiles\0abcdefg.ext</code> to be generated.</td>
</tr>
<tr>
<td>RemoveOnGet</td>
<td>If set to Yes, the rule issues the FTP command to remove the remote source file after getting it—if the user ID used can remove files from the FTP site. This is done to allow clean up activities. The default is No, which helps when you are debugging.</td>
</tr>
<tr>
<td>RemoveOnPut</td>
<td>If set to Yes, the local source file is removed as soon as the Put operation is complete. This reduces the number of temporary files. The file is removed even if the Put operation failed. The default is No, which helps when you are debugging.</td>
</tr>
<tr>
<td>Debug</td>
<td>Determines if the rule logs its actions to the DSRVTRC.LOG file. Set this option to Yes for debugging purposes, but be sure to change the option to No when you are ready to use the system in a production environment. The default is No. See the Sample debug log on page 201 for an example.</td>
</tr>
<tr>
<td>CacheGetFile</td>
<td>Enter the number of seconds the rule should store the file it got from the remote FTP server using the IDS file cache. The default is 3600 (1 hour). See also IRLPurgeCache on page 204.</td>
</tr>
</tbody>
</table>
If you omit the user ID and password in either the attachment or in the configuration file, the system makes an anonymous connection. Keep in mind that if you set up your FTP server to allow anonymous connections, anyone can FTP in and see your files and anyone can put files in. You can solve this problem by setting the FTP server to refuse all connections except those from specified IP addresses.

Both the configuration file options and the attachment variables can provide all of the needed information for FTP operations (server address, user ID, password, port), so the same IDS setup can FTP to different FTP servers, if needed.

The web application is responsible for removing any file sent to it via FTP. For example, when IDS FTPs the file to the web application, IDS removes the local file it created. The web application must remove the file it got via FTP from IDS. IDS can also remove the remote file it got via the FTP using the RemoveOnGet option.

**NOTE:** You can use multiple IRLFileFTP rules on the same request type with different rule parameters if necessary for getting or putting multiple files.

Here is another example:

In this example, on DSI_MSGRUNR, you want to transfer a file called MYFILE.DOC from the incoming directory on an FTP server called testftp into the local directory called e:\temp and you want IRLFileFTP to generate a name for the local destination file.

Additionally, on DS_MSGRUNR, you want to transfer a file called MYFILE.PDF from the local directory called e:\temp into the incoming directory on the FTP server and you want IRLFileFTP to generate a name for the remote destination file. Assume you are using anonymous FTP. Here's one way you would could set this up:

First, add these INI options in your DOCSERV configuration file:

```
< ReqType:PRT >
  ...
  Function = irlw32->IRLFileFTP,GETREM->GETLOC,PUTLOC->PUTREM
  ...
< FTP:PRT >
  GetFilePath = e:\temp
< FTP >
  Server = testftp
```
PutFilePath = incoming
Debug = Yes

Then set these attachment variables:

- Input attachment: GETREM = incoming\myfile.doc
- Output attachment: PUTLOC = e:\temp\myfile.pdf

When running a transaction with these settings, IRLFileFTP creates the variable GETLOC on the input attachment and will fill it with a temporary name such as e:\temp\E0A79110D30D11D2AA2600104BD359C8.doc. It also creates the variable PUTREM on the output attachment and fills it with a temporary name such as incoming\E0A79111D30D11D2AA2600104BD359C8.pdf.

See the sample debug log for the results of running a transaction with the settings in this example.

### Attachment outputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTPGETRESULTS</td>
<td>A value of SUCCESS or ERROR.</td>
</tr>
<tr>
<td>FTPPUTRESULTS</td>
<td>A value of SUCCESS or ERROR.</td>
</tr>
<tr>
<td>RESULTS</td>
<td>A value of SUCCESS, if the GET and PUT operations succeeded, otherwise the last error code returned.</td>
</tr>
<tr>
<td>RemotePutFile</td>
<td>Where RemotePutFile represents the rule argument name for the remote put file. This is only present if the rule generated a unique file name for the remote file in a PUT operation.</td>
</tr>
<tr>
<td>LocalGetFile</td>
<td>Where LocalGetFile represents the rule argument name for the local get file. This is only present if the rule generated a unique file name for the local file in a PUT operation.</td>
</tr>
</tbody>
</table>

### Errors

You may encounter these errors:

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRL0026</td>
<td>The IRLInitFTP rule was not set up in theINI request type.</td>
</tr>
<tr>
<td>IRL0027</td>
<td>The IRLFileFTP rule does not contain any rule arguments.</td>
</tr>
<tr>
<td>IRL0028</td>
<td>Indicates a missing server option.</td>
</tr>
<tr>
<td>IRL0029</td>
<td>Unable to connect to the FTP server. Check the Server, UserID and Password options.</td>
</tr>
<tr>
<td>IRL0030</td>
<td>Indicates a missing attachment variable. The name of the missing attachment variable is also indicated.</td>
</tr>
<tr>
<td>IRL0031</td>
<td>An unexpected error occurred. The error code and description is also included.</td>
</tr>
</tbody>
</table>

### Returns

Success or failure
Sample debug log

Here is a sample debug log produced if you use the Debug option in the FTP control group. This debug log is based on the example above.

1. IRLFileFTP after parsing using: <GETREM> for GetFileRemote, <GETLOC> for GetFileLocal, <PUTLOC> for PutFileLocal, <PUTREM> for PutFileRemote
2. Attachment value FTPUSERID is not found. Looking for INI value <FTP:PRT> UserID =
3. INI value is not found. Looking for INI value <FTP> UserID =
4. USERID is not found.
5. Attachment value FTPPASSWORD is not found. Looking for INI value <FTP:PRT> Password =
6. INI value is not found. Looking for INI value <FTP> Password =
7. PASSWORD is not found.
8. Attachment value FTPSERVER is not found. Looking for INI value <FTP:PRT> Server =
9. INI value is not found. Looking for INI value <FTP> Server =
10. Attachment value FTPSERVERPORT is not found. Looking for INI value <FTP:PRT> Port =
11. INI value is not found. Looking for INI value <FTP> Port =
16. Created new FTP connection
17. Successful get current directory ///</>
18. Did not find <GETLOC> in the attachment. Generated name: e:\temp\E0A7911D30D11D2AA2600104BD359C8.DOC>
19. Did not find <PUTREM> in the attachment. Generated name: <e:\temp\incoming\E0A7911D30D11D2AA2600104BD359C8.PDF>
20. Successful GetFile
22. Attachment value FTPUSERID is not found. Looking for INI value <FTP:PRT> UserID =
23. INI value is not found. Looking for INI value <FTP> UserID =
24. USERID is not found.
25. Attachment value FTPPASSWORD is not found. Looking for INI value <FTP:PRT> Password =
26. INI value is not found. Looking for INI value <FTP> Password =
27. PASSWORD is not found.
28. Attachment value FTPSERVER is not found. Looking for INI value <FTP:PRT> Server =
29. INI value is not found. Looking for INI value <FTP> Server =
30. Attachment value FTPSERVERPORT is not found. Looking for INI value <FTP:PRT> Port =
31. INI value is not found. Looking for INI value <FTP> Port =
32. Using FTP UserID <>.
33. Using FTP Password <>.
34. Using FTP Server <testftp>.
36. Found existing FTP connection
37. Successful get current directory ///</>
38. Successful PutFile.
IRLInitFTP

Use this rule to create and destroy an InternetSession object. This rule creates and destroys two global DSI variables: INTERNETSESSION and FTPCONNECTIONS.

Syntax

```c
long __DSI_API IRLInitFTP ( DSIHANDLE hInstance,
                          char * pszParms,
                          unsigned long ulMsg,
                          unsigned long ulOptions )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIHANDLE hInstance</td>
<td>DSI instance handle</td>
</tr>
<tr>
<td>char * pszParms</td>
<td>Pointer to rule parameter string</td>
</tr>
<tr>
<td>unsigned long ulMsg</td>
<td>DSI_MSG???? message, such as DSI_MSGRUNF</td>
</tr>
<tr>
<td>unsigned long ulOptions</td>
<td>Options</td>
</tr>
</tbody>
</table>

You must register this rule using an INI request. Here is an example:

```
< ReqType:INI >
  Function = irlw32->IRLInitFTP
```

Returns

Success or failure

Errors

None
IRLLog

Use this rule to return records from server access log or error log files.

Syntax

```c
long _DSIAPI IRLLog ( DSIHANDLE hInstance,
                     char * pszParms,
                     unsigned long ulMsg,
                     unsigned long ulOptions )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIHANDLE hInstance</td>
<td>DSI instance handle</td>
</tr>
<tr>
<td>char * pszParms</td>
<td>Pointer to rule parameter string</td>
</tr>
<tr>
<td>unsigned long ulMsg</td>
<td>DSI_MSG???? message, such as DSI_MSGRUNF</td>
</tr>
<tr>
<td>unsigned long ulOptions</td>
<td>Options</td>
</tr>
</tbody>
</table>

The error log report is created in this format:

- `REQTYP`E
- `TIME`
- `USERID`
- `RESULT`
- `REASON`
- `AREA`

The access log includes these fields:

- USERID
- REM_ADDR
- REQTYP
- STATUS
- RESULT
- INTIME

The default DOCSERV configuration file sets this rule with this INI option:

```
< ReqType:VLF >
  Function = irlw32->IRLLog
```

Returns

Success or failure

Errors

You may encounter these errors:

- Missing necessary attachment variable
- Failed to locate records
- Failed to append to attachment
IRLPurgeCache

Use this rule to remove expired files. The rule runs on the timer (SAR) request and removes all files registered in the server cache table after the specified time has expired.

Syntax

```c
long _DSIAPI IRLPurgeCache ( DSIHANDLE hInstance,
                        char * pszParms,
                        unsigned long  ulMsg,
                        unsigned long  ulOptions )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIHANDLE hInstance</td>
<td>DSI instance handle</td>
</tr>
<tr>
<td>char * pszParms</td>
<td>Pointer to rule parameter string</td>
</tr>
<tr>
<td>unsigned long ulMsg</td>
<td>DSI_MSG??? message, such as DSI_MSGRUNF</td>
</tr>
<tr>
<td>unsigned long ulOptions</td>
<td>Options</td>
</tr>
</tbody>
</table>

This rule uses the following INI option to remove records from the result queue which were not picked up by a front-end client.

```
<DOCSRVR>
  ExpireTransactions = 86400

The default value is 86400 seconds, which is 24 hours. With this setting, all records in the result queue with an in time older than 24 hours will be removed.

The default DOCSERV configuration file sets this rule with these INI settings.

```
< ReqType:SAR >
  Function = irlw32->IRLPurgeCache
```

Returns

Success or failure

Errors

None
IRLSearch

Use this rule to return a list of matching table records.

Syntax

```c
long _DSIAPI IRLSearch ( DSIHANDLE hInstance,
                         char * pszParms,
                         unsigned long ulMsg,
                         unsigned long ulOptions )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIHANDLE hInstance</td>
<td>DSI instance handle</td>
</tr>
<tr>
<td>char * pszParms</td>
<td>Pointer to rule parameter string</td>
</tr>
<tr>
<td>unsigned long ulMsg</td>
<td>DSI_MSG??? message, such as DSI_MSGRUNF</td>
</tr>
<tr>
<td>unsigned long ulOptions</td>
<td>Options</td>
</tr>
</tbody>
</table>

You can use this INI option with this rule:

```ini
< ArcRet >
MaxRecords = 100
```

Returns

Success or failure

Errors

You may encounter these errors:

- Missing necessary attachment variable
- Failed to locate records
- Failed to append to attachment
IRLSendVersion

Use this rule to report DLL version information.

Syntax

```c
long _DSIAPI IRLSendVersion ( DSIHANDLE hInstance,
    char * pszParms,
    unsigned long  ulMsg,
    unsigned long  ulOptions )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIHANDLE hInstance</td>
<td>DSI instance handle</td>
</tr>
<tr>
<td>char * pszParms</td>
<td>Pointer to rule parameter string</td>
</tr>
<tr>
<td>unsigned long ulMsg</td>
<td>DSI_MSGRUNF message, such as DSI_MSGRUNF</td>
</tr>
<tr>
<td>unsigned long ulOptions</td>
<td>Options</td>
</tr>
</tbody>
</table>

For each of the following DLLs, this rule creates attachment variables on the DSI_MSGRUNF message.

- IRL
- IRP
- DQM
- IBASE
- DCB
- ATC
- DSJI

Here is a list of the variables:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Tells you the...</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>name of the DLL</td>
</tr>
<tr>
<td>VERSION</td>
<td>version of the DLL, such as 100.012.XXX</td>
</tr>
<tr>
<td>DATE</td>
<td>date of the last compile in MMM DD YYYY format</td>
</tr>
<tr>
<td>TIME</td>
<td>time of the last compile in HH:MM:SS format</td>
</tr>
</tbody>
</table>

These values only change when you upgrade to a newer version.

The default DOCSERV configuration file sets this rule with this INI option.

```
< ReqType:SSS >
    Function = irlw32->IRLSendVersion
```

Returns

Success or failure
Errors
None

See also
IRCSendVersion on page 242
IRLStatistics

Use this rule to compile server statistics.

**Syntax**

```c
long _DSI_API IRLStatistics ( DSIHANDLE hInstance,
                            char * pszParms,
                            unsigned long ulMsg,
                            unsigned long ulOptions )
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIHANDLE hInstance</td>
<td>DSI instance handle</td>
</tr>
<tr>
<td>char * pszParms</td>
<td>Pointer to rule parameter string</td>
</tr>
<tr>
<td>unsigned long ulMsg</td>
<td>DSI_MSG???? message, such as DSI_MSGRUNF</td>
</tr>
<tr>
<td>unsigned long ulOptions</td>
<td>Options</td>
</tr>
</tbody>
</table>

This rule creates the following values in the attachment:

<table>
<thead>
<tr>
<th>Value</th>
<th>Tells you the...</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPTIME</td>
<td>time the server started, in format: Mon Dec 22 15:37:31 1999</td>
</tr>
<tr>
<td>SUCCESSCOUNT</td>
<td>number of successful transactions</td>
</tr>
<tr>
<td>ERRORCOUNT</td>
<td>number of transactions in error</td>
</tr>
<tr>
<td>ALLOCCOUNT</td>
<td>number of memory allocations—used for debugging purposes</td>
</tr>
<tr>
<td>RESTARTCOUNT</td>
<td>number of times the Internet Document Server been restarted</td>
</tr>
<tr>
<td>LASTRESTART</td>
<td>time of the last restart, in format: Mon Dec 22 15:37:31 1999</td>
</tr>
<tr>
<td>FREECOUNT</td>
<td>number of memory deallocations—used for debugging purposes</td>
</tr>
</tbody>
</table>

The default DOCSEV configuration file sets this rule with these INI settings.

```
< ReqType:SSS >
    Function = irlw32->IRLStatistics
```

**Returns**

Success or failure

**Errors**

None
public class com.docucorp.ids.rules.AttachmentFilerRule

This class contains rule functions that send and receive files through attachments in DSIMessages. The files can be binary or text. Create objects of this class with transaction scope since receiveFile uses information in the object in both the MSG_RUNF and MSG_RUNR messages.

This class implements the substitution for these IDS 1.x rules:

- ATCSendFile
- ATCReceiveFile

<table>
<thead>
<tr>
<th>Constructors</th>
<th>public AttachmentFilerRule()</th>
</tr>
</thead>
</table>

| Methods     | public int sendFile(          |
|             | RequestState requestState,   |
|             | String arg,                  |
|             | int msg)                     |

Use this rule to read a file from disk in binary or text format and put it in an attachment in the output DSIMessage to be sent back to the client application.

In the MSG_RUNR message this rule will read three parameters from arg, separated by commas. The three parameters are attachment name, file name message variable, and file type.

Attachment name is the name that the file data is stored in the output DSIMessage's attachments.

File name message variable is the name of the message variable that has the file name in it. The file type is either TEXT or BINARY, specifying the type of file to be read. For example, if the rule is specified in the configuration as:

```
java;com.docucorp.ids.rules.AttachmentFilerRule;;transaction;sendFile;ZZZ,IMPORTFILE,TEXT
```

and the message variable IMPORTFILE contains '/home/docserv/client/test.txt,' then the file 'test.txt' is added to the DSIMessage in a text attachment named 'ZZZ'.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestState</td>
<td>Object that holds the current running state of the request at this point of execution.</td>
</tr>
<tr>
<td>arg</td>
<td>Arguments from the rule line of the configuration file.</td>
</tr>
<tr>
<td>msg</td>
<td>Message that is currently being run, either MSG_INIT, MSG_RUNF, MSG_RUNR or MSG_TERM.</td>
</tr>
</tbody>
</table>

| Returns | |
|---------|RET_SUCCESS if successful, else RET_FAIL, usually caused by the file not being found, missing message variable, and so on. |
Use this rule to write a file to disk in binary or text format from an attachment in the input DSIMessage, usually sent from a client application.

In the MSG_RUNF message this rule reads these parameters from arg: attachment name, file name attachment variable, file name, and disposition. The parameters should be separated by commas.

Attachment name is the name that the file data is stored in the input DSIMessage's attachments. The file type, text or binary, is stored in the attachment and the file is written in the proper mode.

File name message variable is the name of the message variable that will have the file name stored in it.

File name is the name of the file to write. If it is a regular file name the file is overwritten each time the rule is run. If the file name has an asterisk (*) in it, the asterisk is replaced with a unique string, causing different files to be written each time the rule is run. In either case the file name that is used is stored in the file name message variable.

Disposition determines if the file is erased during the MSG_RUNR message. If disposition is set to **KEEP** then the file is kept, otherwise it is erased.

For example, if the rule is specified in the configuration as:

```
java;com.docucorp.ids.rules.AttachmentFilerRule;transaction;receiveFile;ZZZ,IMPORTFILE,/home/docserv/client/test.txt,KEEP
```

then the file named test.txt is written to disk with data in the ZZZ attachment and the file name is stored in the message variable IMPORTFILE.

If the file name was instead `/home/docserv/client/*.txt`, then a unique file name ending with `.txt` would be generated and that would be stored in IMPORTFILE.

In the MSG_RUNR message the rule will erase the file written in the MSG_RUNF message, unless the disposition was set to **KEEP**.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestState</td>
<td>Object that holds the current running state of the request at this point of execution.</td>
</tr>
<tr>
<td>arg</td>
<td>Arguments from the rule line of the configuration file.</td>
</tr>
<tr>
<td>msg</td>
<td>Message that is currently being run, either MSG_INIT, MSG_RUNF, MSG_RUNR or MSG_TERM.</td>
</tr>
</tbody>
</table>

### Returns

RET_SUCCESS if successful, else RET_FAIL, an invalid or empty parameter in arg.
public class com.docucorp.ids.rules.BLPPurgeRule

extends com.docucorp.ids.rules.AbstractIDSJavaRule

Use this class to delete files in the file cache when the file's expiration time has passed. This class implements the substitution for the IDS 1.x rule IRLPurgeCache.

<table>
<thead>
<tr>
<th>Constructors</th>
<th>public BLPPurgeRule()</th>
</tr>
</thead>
</table>

| Methods       | public int purge(       |
|               | RequestState requestState, |
|               | String arg,             |
|               | int msg)                |

During the MSG_RUNR message this rule calls a function that checks the files that have been cached to see if any of the file lifetimes have expired, and if they have then deletes the files. No arguments are expected from the function line.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestState</td>
<td>Object that holds the current running state of the request at this point of execution.</td>
</tr>
<tr>
<td>arg</td>
<td>Arguments from the rule line of the configuration file.</td>
</tr>
<tr>
<td>msg</td>
<td>Message that is currently being run, either MSG_INIT, MSG_RUNF, MSG_RUNR or MSG_TERM.</td>
</tr>
</tbody>
</table>

Returns RET_SUCCESS.
public class com.docucorp.ids.rules.BLPStatisticsRule

extends com.docucorp.ids.rules.AbstractIDSJavaRule

Use the rule in this class to add statistical information to the output attachment. This is usually called as part of a SSS request.

This class implements the substitution of the IDS 1.x rule IRLStatistics.

<table>
<thead>
<tr>
<th>Constructors</th>
<th>public BLPStatisticsRule()</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
<th>public int addStatistics( RequestState requestState, String arg, int msg)</th>
</tr>
</thead>
</table>

During the MSG_RUNF message add statistical information to the output DSIMessage. Currently includes number of successful transactions, number of errors, number of restarts, time when BLP was started and time of the last restart. No arguments are expected from the function line.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestState</td>
<td>Object that holds the current running state of the request at this point of execution.</td>
</tr>
<tr>
<td>arg</td>
<td>Arguments from the rule line of the configuration file.</td>
</tr>
<tr>
<td>msg</td>
<td>Message that is currently being run, either MSG_INIT, MSG_RUNF, MSG_RUNR or MSG_TERM.</td>
</tr>
</tbody>
</table>

| Returns | RET_SUCCESS if successful, else RET_FAIL. |
public class `com.docucorp.ids.rules.CopyDataRule`

extends `com.docucorp.ids.rules.AbstractIDSJavaRule`

Use the rule in this class to copy message variables and attachments from the input DSIMessage to the output DSIMessage.

This class implements the substitution of the IDS 1.x rule IRLCopyAttachment.

<table>
<thead>
<tr>
<th>Constructors</th>
<th>public CopyDataRule()</th>
</tr>
</thead>
</table>

| Methods | public int copyData(  
|         | RequestState requestState,  
|         | String arg,  
|         | int msg) |

During the MSG_RUNR message copy all message variables and attachments from the input DSIMessage to the output DSIMessage. No arguments are expected from the function line.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestState</td>
<td>Object that holds the current running state of the request at this point of execution.</td>
</tr>
<tr>
<td>arg</td>
<td>Arguments from the rule line of the configuration file.</td>
</tr>
<tr>
<td>msg</td>
<td>Message that is currently being run, either MSG_INIT, MSG_RUNF, MSG_RUNR or MSG_TERM.</td>
</tr>
</tbody>
</table>

| Returns | RET_SUCCESS if successful, else RET_FAIL. |
public class com.docucorp.ids.rules.FTPRule

extends com.docucorp.ids.rules.AbstractIDSRule

Use the rules in this class to transfer files back and forth over FTP connections. There are two sets of rules in the class. One set is used in IDS 2.x, the other is used for IDS 1.x Java rule compatibility. Each method is marked as to how it should be used.

There is a method that is run in the INI request that stores and caches FTP connections and a method that does the actual file transfer. This class implements the substitution for these IDS 1.x rules:

- IRLInitFTP
- IRLFileFTP

### Constructors

| public FTPRule() |

### Methods

All of these methods are used for IDS 1.x compatibility.

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>public int setupMethod(int dsih, String arg, int ulMsg, int ulOptions)</td>
</tr>
<tr>
<td>public int convertParameter=Description(RequestState requestState, String arg, int msg)</td>
</tr>
<tr>
<td>public int transferMethod(int dsih, String arg, int ulMsg, int ulOptions)</td>
</tr>
<tr>
<td>public int setupConnections(RequestState requestState, String arg, int msg)</td>
</tr>
</tbody>
</table>

Use these methods to create data to run multiple FTP transfers in the MSG_INIT message and destroy the data in the MSG_TERM message. Use this rule in the INI request type.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestState</td>
<td>Object that holds the current running state of the request.</td>
</tr>
<tr>
<td>arg</td>
<td>Arguments from the rule line of the configuration file.</td>
</tr>
<tr>
<td>msg</td>
<td>Message currently being run, either MSG_INIT, MSG_RUNF, MSG_RUNR, or MSG_TERM.</td>
</tr>
</tbody>
</table>
Returns
RET_SUCCESS if successful, otherwise RET_FAIL

Methods

| public int transferFiles( |
| RequestState requestState, |
| String arg, |
| int msg) |

Use this rule to do the actual file transfers through FTP. Files are retrieved during the MSG_RUNF message and sent during the MSG_RUNR message. For example, if the rule is specified in the configuration as:

```
java;com.docucorp.ids.rules.FTPRule;;transaction;transferFiles;GetFileRemote->PutFileRemote
```

GetFileRemote is the name of the message variable which contains the name of the file to get via FTP from the FTP server. This variable must be in the input attachment. This name is not a URL, it is the name of a file and, optionally, an FTP directory name. For instance, for

```
ftp://servername/incoming/file.dat
```

you would enter the name

```
incoming/file.dat
```

GetFileLocal is the name of the message variable which contains the name of the destination file (to be written locally to the IDS machine). If this variable exists, it must be in the input DSIMessage. If this variable is not found, the rule generates a unique name, adds the message variable to the input attachment, and sets the value of the variable to the generated name. See FTPGetFilePath, below, for information on how to prefix this name with a path. If the file exists when the GET operation is executed, it is overwritten.

PutFileLocal is the name of the message variable which contains the name of the local (to IDS) source file to be put via FTP onto the FTP server. This variable must be in the output DSIMessage.

PutFileRemote The name of the message variable which contains the name under which the destination file is to be written to the FTP server. If this variable exists, it must be in the output DSIMessage. If you supply this variable, bear in mind that the name it holds is not a URL, it is the name of a file and, optionally, an FTP directory name. For instance, for

```
ftp://servername/incoming/file.dat
```

you would enter incoming/file.dat

If this variable is not found in the output DSIMessage, this rule generates a unique name, adds the variable to the output DSIMessage, and sets the value of the variable to the generated name. See FTPPutFilePath, below, for information on how to prefix this name with an FTP directory. If the file exists when the PUT operation is executed, the file is overwritten.

If the Get names are missing, no FTP Get action is performed by this rule and no error message is generated. If the Put names are missing, no FTP Put action is performed by this rule and no error message is generated.

You can register multiple FTPRule rules on the same request type if you need more than one file FTP. This rule maintains the list of open FTP connections and reuses connections when possible. For example, if the rule is specified in the configuration as:

```
java;com.docucorp.ids.rules.FTPRule;;transaction;transferFiles;GetFileRemote->GetFileLocal,PutFileLocal->PutFileRemote
```
GetFileRemote is the name of the message variable that contains the name of the file to get via FTP from the FTP server. This variable must be in the input attachment. This name is not a URL, it is the name of a file and, optionally, an FTP directory name.

For instance, for

```
ftp://servername/incoming/file.dat
```
you would enter the name

```
incoming/file.dat.
```

GetFileLocal is the name of the message variable that contains the name of the destination file (to be written locally to the IDS machine). If this variable exists, it must be in the input DSIMessage. If this variable is not found, the rule generates a unique name, adds the message variable to the input attachment, and sets the value of the variable to the generated name. See below, for information on how to prefix this name with a path. If the file exists when the GET operation is executed, it is overwritten.

PutFileLocal is the name of the message variable that contains the name of the local (to IDS) source file to be put via FTP onto the FTP server. This variable must be in the output DSIMessage.

PutFileRemote is the name of the message variable that contains the name under which the destination file is to be written to the FTP server. If this variable exists, it must be in the output DSIMessage. If you supply this variable, bear in mind that the name it holds is not a URL, it is the name of a file and, optionally, an FTP directory name.

For instance, for

```
ftp://servername/incoming/file.dat
```
you would enter

```
incoming/file.dat
```

If this variable is not found in the output DSIMessage, this rule generates a unique name, adds the variable to the output DSIMessage, and sets the value of the variable to the generated name. See below, for information on how to prefix this name with an FTP directory. If the file exists when the PUT operation is executed, the file is overwritten.

If the Get names are missing, no FTP Get action is performed by this rule and no error message is generated. If the Put names are missing, no FTP Put action is performed by this rule and no error message is generated.

You can register multiple FTPRule rules on the same request type if you need more than one file FTP. This rule maintains the list of open FTP connections and reuses connections when possible.

If a connection is dropped, this rule reopens it. The default timeout value on an FTP server is 900 seconds, so the connection will stay open for at least this amount of time before it is dropped.

There are several FTP setup parameters required to transfer files, for example the Internet address of the remote machine. There are multiple ways to specify these parameters, first through message variables then through configuration options. This is also the order in which the parameters are searched. For example, if the remote machine is specified through a message variable this overrides any parameters in the configuration.
There are several optional message variables which you can use with this rule. For instance, you can set the values represented by these message variables in the configuration. If, however, the message variable is present, its value will override any corresponding value in the configuration.

You must specify the server through the FTPServer attachment variable or by using a configuration option. You can omit any of the variables you do not need.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTPServer</td>
<td>The server name or IP address for the FTP connection.</td>
</tr>
<tr>
<td>FTPUserID</td>
<td>The user ID for the FTP connection.</td>
</tr>
<tr>
<td>FTPPassword</td>
<td>The password for the FTP Connection</td>
</tr>
<tr>
<td>FTPServerPort</td>
<td>The server’s FTP port.</td>
</tr>
<tr>
<td>FTPGetFilePath</td>
<td>The path to be prefixed to the unique name transferFiles generates when the value for GetFileLocal does not exist on the input attachment. For example, /home/temp causes local names such as /home/temp/0abcdefg.ext to be generated.</td>
</tr>
<tr>
<td>FTPPutFilePath</td>
<td>The FTP directory path (omit the drive specifier) to be prefixed to the unique name transferFiles generates when the variable for PutFileRemote does not exist on the output attachment. For example, incoming/datafiles causes FTP names such as incoming/datafiles/0abcdefg.ext to be generated.</td>
</tr>
</tbody>
</table>

You must specify the server through the FTPServer message variable or by using a configuration option. You can omit any configuration option you do not need. The transferFiles rule searches for each value that can be specified in the optional message variables in this order:

1. First search the input DSIMessage for a message variable that contains the value
2. If not found, search the FTP:ReqType section for the corresponding value
3. If not found, search the FTP control section for the corresponding value
4. For get and put paths, if not found search the Attachment section

This search order lets you have unique values for a given transaction and unique values for any given request type, or have the same values for all transactions and request types. For example, you may have several request types that use the transferFiles rule. One request type could be set up with a section that provides unique values, while all other request types could use the values defined in the FTP section.

Here is an example of the configuration options:

```
<section name="FTP:ReqType">
  <entry name="Server">ftp.yourcompany.com</entry>
  <entry name="UserID">customer</entry>
  <entry name="Password">password</entry>
  <entry name="RemoveOnGet">No</entry>
  <entry name="RemoveOnPut">No</entry>
  <entry name="CacheGetFile">10</entry>
</section>
```
The options for the FTP:ReqType section are:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>The server name or IP address for the FTP connection. Corresponds to message variable FTPServer.</td>
</tr>
<tr>
<td>UserID</td>
<td>The user ID for the FTP connection. Corresponds to message variable FTPUserID.</td>
</tr>
<tr>
<td>Password</td>
<td>The password for the FTP Connection. Corresponds to message variable FTPPassword.</td>
</tr>
<tr>
<td>ServerPort</td>
<td>The server’s FTP port. Corresponds to message variable FTPServerPort.</td>
</tr>
<tr>
<td>GetFilePath</td>
<td>The path to be prefixed to the unique name transferFiles generates when the variable for GetFileLocal does not exist on the input attachment. Corresponds to message variable FTPGetFilePath.</td>
</tr>
<tr>
<td>PutFilePath</td>
<td>The FTP directory path (omit the drive specifier) to be prefixed to the unique name transferFiles generates when the variable for PutFileRemote does not exist on the output attachment. Corresponds to message variable FTPServer.</td>
</tr>
</tbody>
</table>

The options for the FTP section are:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RemoveOnGet</td>
<td>If set to Yes, the rule issues the FTP command to remove the remote source file after getting it, if the user ID used can remove files from the FTP site. This is done to allow clean up activities. The default is Yes. Enter No for debugging purposes.</td>
</tr>
<tr>
<td>RemoveOnPut</td>
<td>If set to Yes, the local source file is removed as soon as the Put operation is complete. This reduces the number of temporary files. The default is Yes. The file is removed even if the Put operation failed. Enter No for debugging purposes.</td>
</tr>
<tr>
<td>CacheGetFile</td>
<td>Enter the number of seconds the rule should store the file it got from the remote FTP server using the IDS file cache. The default is 3600 (1 hour). See also BLPPurgeRule.purge.</td>
</tr>
</tbody>
</table>

The options for the Attachment section are:
If you omit the user ID and password in either the message variable or in the configuration, the system makes an anonymous connection. Keep in mind that if you set up your FTP server to allow anonymous connections, anyone can FTP in and see your files and anyone can put files in. You can solve this problem by setting the FTP server to refuse all connections except those from specified IP addresses. Both the configuration options and the message variables can provide all of the needed information for FTP operations (server address, user ID, password, port), so the same IDS setup can FTP to different FTP servers, if needed.

The web application is responsible for removing any file sent to it via FTP. For example, when IDS FTPs the file to the web application, IDS removes the local file it created. The web application must remove the file it got via FTP from IDS. IDS can also remove the remote file it got via the FTP using the RemoveOnGet option.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestState</td>
<td>Object that holds the current running state of the request at this point of execution.</td>
</tr>
<tr>
<td>arg</td>
<td>Arguments from the rule line of the configuration file.</td>
</tr>
<tr>
<td>msg</td>
<td>Message that is currently being run, either MSG_INIT, MSG_RUNF, MSG_RUNR or MSG_TERM.</td>
</tr>
</tbody>
</table>

### Returns

RET_SUCCESS if successful, else RET_FAIL.
public class com.docucorp.ids.rules.IDSEncryptionRule

Use the rule in this class to decrypt and encrypt message variables. All of the functions in the class are static, so invoke the rule with static scope. All functions are thread-safe. This class implements the substitution for the IDS 1.x IRLDecryptValue rule.

| Constructors | public IDSEncryptionRule() |

| Methods | public static int cryptVariables( RequestState requestState, String arg, int msg) |

Use this rule to decrypt and encrypt message variables. The argument is a comma-delimited list of message variables to work on.

On MSG_RUNF the variables are taken from the input message, decrypted, and put back in the input message.

On MSG_RUNR the variables are taken from the output message, encrypted, and put back in the output message.

If a message variable is not found in the message a warning is generated but processing continues on the other variables.

The rule also supports wildcard message variable names by putting an asterisk (*) in the message variable name. The asterisk can go at the beginning, middle, or end of a message variable name.
public class com.docucorp.ids.rules.IDSInitRule

extends com.docucorp.ids.rules.AbstractIDSJavaRule

Use the rule in this class to start IDS server utilities, such as those used for purging files and logging transactions. This class implements the substitution for the IDS 1.x IRLInit rule.

Use this rule to initialize and terminate IDS server-wide utilities.

In the MSG_INIT message this rule will do initialization for the server-wide file cache and transaction log. In the MSG_TERM message this rule will terminate the file cache and transaction log.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>requestState</td>
</tr>
<tr>
<td>arg</td>
</tr>
<tr>
<td>msg</td>
</tr>
</tbody>
</table>

Returns RET_SUCCESS
public class com.docucorp.ids.rules.IDSTransactionRule

Use the rule in this class to report transaction times to IDS clients.

This class implements the substitution of the non-attachment part of the IDS 1.x rule ATCUnloadAttachment.

<table>
<thead>
<tr>
<th>Constructors</th>
<th>public IDSTransactionRule()</th>
</tr>
</thead>
</table>

| Methods | public static int reportTimes(RequestState requestState, String arg, int msg) |

Use this rule to report the amount of time a request takes to run on the server. The IDS 1.x rule ATCUnloadAttachment would do this in addition to other functions now built into IDS.

In the MSG_RUNR message this rule adds a message variable SERVERTIMESPENT to the output DSIMessage listing the time spent on the transaction in seconds. If the argument is INCLUDEMS then this rule also adds a message variable SERVERTIMESPENTMS which lists the time in milliseconds. SERVERTIMESPENTMS is useful if IDS is logging transactions since it is easier to sort by time spent in this format.

If using this rule it should be the first rule in the request, or the second if also logging transactions.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestState</td>
<td>Object that holds the current running state of the request at this point of execution.</td>
</tr>
<tr>
<td>arg</td>
<td>Arguments from the rule line of the configuration file.</td>
</tr>
<tr>
<td>msg</td>
<td>Message that is currently being run, either MSG_INIT, MSG_RUNF, MSG_RUNR or MSG_TERM.</td>
</tr>
</tbody>
</table>

| Returns | RET_SUCCESS |
public class com.docucorp.ids.rules.LogTransactionRule

Use the rules in this class to control the logging of transactions in databases. The rules log message variables in a database specified in the configuration and purge expired database tables.

All rule methods in this class should be called with static scope.

This class implements the substitution of the non-attachment part of these IDS 1.x rules:

- ATCLogTransaction
- IRLClearLog

### Constructors

| public LogTransactionRule() |

| Methods | public static int logTransaction( RequestState requestState, String arg, int msg) |

Use this rule to store message variables in a database table set up in the IDS configuration. In the MSG_RUNR message this rule will add a message variables from the output DSIMessage to a database that can be browsed by other applications.

If using this rule it should be the first rule in the request.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestState</td>
<td>Object that holds the current running state of the request at this point of execution.</td>
</tr>
<tr>
<td>arg</td>
<td>Arguments from the rule line of the configuration file.</td>
</tr>
<tr>
<td>msg</td>
<td>Message that is currently being run, either MSG_INIT, MSG_RUNF, MSG_RUNR or MSG_TERM.</td>
</tr>
</tbody>
</table>

| Returns | RET_SUCCESS. |

| Methods | public static int purgeOldTransactionTables( RequestState requestState, String arg, int msg) |

Use this rule to delete database tables that have expired. The expiration time is set up in the IDS configuration. In the MSG_RUNR message this rule will drop database tables that are no longer needed.
## DSI Processing Rules

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestState</td>
<td>Object that holds the current running state of the request at this point of execution.</td>
</tr>
<tr>
<td>arg</td>
<td>Arguments from the rule line of the configuration file.</td>
</tr>
<tr>
<td>msg</td>
<td>Message that is currently being run, either MSG_INIT, MSG_RUNF, MSG_RUNR or MSG_TERM.</td>
</tr>
</tbody>
</table>

### Returns

RET_SUCCESS.
CLIENT RULES

These rules may only be run in the front-end client. The rules are listed in alphabetical order, as shown below:

- ATCAppend2Attachment on page 226
- ATCLoadAttachment on page 227
- ATCLogTransaction on page 228
- ATCReceiveFile on page 230
- ATCSendFile on page 233
- ATCSendMultipleFiles on page 235
- ATCUnloadAttachment on page 236
- IRCInit on page 238
- IRCPrint on page 239
- IRCRequest on page 240
- IRCResult on page 241
- IRLSendVersion on page 206
- IRCUnloadPage on page 244
ATCAppend2Attachment

Use this rule to append values from an INI file to the queue attachment.

Syntax

```c
long _DSI_API ATCAppend2Attachment ( DSIHANDLE hInstance,
char * pszParms,
unsigned long  ulMsg,
unsigned long  ulOptions )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIHANDLE hInstance</td>
<td>DSI instance handle</td>
</tr>
<tr>
<td>char * pszParms</td>
<td>Pointer to rule parameter string</td>
</tr>
<tr>
<td>unsigned long ulMsg</td>
<td>DSI_MSG?? message, such as DSI_MSGRUNF</td>
</tr>
<tr>
<td>unsigned long ulOptions</td>
<td>Options</td>
</tr>
</tbody>
</table>

Use these optional INI control groups when REQTYPE is the REQUEST type.

- `< ATTACH:Default >`
- `< ATTACH:REQTYPE >`

All of the VALUE=OPTION pairs from the ATTACH:REQTYPE control group are appended to the input queue attachment, followed by the ATTACH:Default control group.

The default DOCCLNT.INI file sets this rule with these INI settings.

- `< ResType:Default >`
  - `Function = atcw32->ATCAppend2Attachment`

Returns

Success or failure

Errors

None
ATCLoadAttachment

Use this rule to parse the attachment from the input queue into the internal format of the DSI_MSGRUNF message. You can then access the attachment via DSI APIs, such as DSILocateAttachVar. This rule frees allocated memory for the internal format in the input queue on the DSI_MSGTERM message.

**Syntax**

```c
long _DSI_API ATCLoadAttachment ( DSIHANDLE hInstance,
char * pszParms,
unsigned long ulMsg,
unsigned long ulOptions )
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIHANDLE hInstance</td>
<td>DSI instance handle</td>
</tr>
<tr>
<td>char * pszParms</td>
<td>pointer to rule parameter string</td>
</tr>
<tr>
<td>unsigned long ulMsg</td>
<td>DSI_MSGRUNF message, such as DSI_MSGRUNF</td>
</tr>
<tr>
<td>unsigned long ulOptions</td>
<td>options</td>
</tr>
</tbody>
</table>

Use these INI settings to specify the location of the attachments.

```
< RequestQ >
  AttachmentPath =
```

The default DOCLNT.INI file sets this rule with these INI settings.

```
< ResType:Default >
  Function = atcw32->ATCLoadAttachment
```

The default DOCSERV configuration file sets this rule with these INI settings.

```
< ReqType:SSS >
  Function = atcw32->ATCLoadAttachment
```

**Returns**

Success or failure

**Errors**

Memory error

**See also**

ATCUnloadAttachment on page 236
**ATCLogTransaction**

Use this rule to write transaction information to log file.

**Syntax**

```c
long _DSIAPI ATCLogTransaction ( DSIHANDLE hInstance,
            char * pszParms,
            unsigned long  ulMsg,
            unsigned long  ulOptions )
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIHANDLE hInstance</td>
<td>DSI instance handle</td>
</tr>
<tr>
<td>char * pszParms</td>
<td>Pointer to rule parameter string</td>
</tr>
<tr>
<td>unsigned long ulMsg</td>
<td>DSI_MSG?? message, such as DSI_MSGRUNF</td>
</tr>
<tr>
<td>unsigned long ulOptions</td>
<td>Options</td>
</tr>
</tbody>
</table>

The default DOCSERV configuration file sets this rule with these INI settings:

```
< ReqType:SSS >
    Function = atcw32->ATCLogTransaction
```

The layout of the server log file is as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Userid</td>
<td>Character</td>
<td>127</td>
</tr>
<tr>
<td>Rem_addr</td>
<td>Character</td>
<td>15</td>
</tr>
<tr>
<td>Rem_host</td>
<td>Character</td>
<td>127</td>
</tr>
<tr>
<td>Rem_user</td>
<td>Character</td>
<td>32</td>
</tr>
<tr>
<td>Reqtype</td>
<td>Character</td>
<td>25</td>
</tr>
<tr>
<td>Status</td>
<td>Character</td>
<td>1</td>
</tr>
<tr>
<td>Result</td>
<td>Character</td>
<td>8</td>
</tr>
<tr>
<td>Intime</td>
<td>Numeric</td>
<td>10</td>
</tr>
<tr>
<td>Sloginfo</td>
<td>Character</td>
<td>127</td>
</tr>
</tbody>
</table>

This rule runs on the RUNR message. It looks looking in the input attachment to get these values. The rule locates the values with the same name as field name in the attachment and puts those values into the record in the LOG table.

The Intime field is supplied by the rule. The Sloginfo field is available for application use. If you want to use it, just add the value to the attachment using the name `Sloginfo`.

**Returns**

Success or failure
Errors

- Log file not initialized
- Memory error
- Database failure
ATCReceiveFile

Use this rule to merge a series of attachment variables into a file and write that file to disk. Generally, this rule is used to re-assemble a file that has been posted in segments to an IDS queue by the ATCSendFile rule. The file that is received can be either a binary or text file.

Syntax

```c
long _DSIAPI ATCReceiveFile ( DSIHANDLE hInstance,
                                char * pszParms,
                                unsigned long ulMsg,
                                unsigned long ulOptions )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIHANDLE hInstance</td>
<td>DSI instance handle</td>
</tr>
<tr>
<td>char * pszParms</td>
<td>Pointer to rule parameter string</td>
</tr>
<tr>
<td>unsigned long ulMsg</td>
<td>DSI_MSG message, such as DSI_MSGRUNF</td>
</tr>
<tr>
<td>unsigned long ulOptions</td>
<td>Options</td>
</tr>
</tbody>
</table>

This rule accepts four parameters (Prefix, AttachmentVariable, FileName, and Disposition) delimited with commas and specified immediately after the rule in the INI file.

The file name you specified (see note below) is opened in write mode in the appropriate manner (binary or text).

After the data is written into the file, the file is closed and the name of the disk file is saved into the attachment variable indicated by the AttachmentVariable parameter. To keep the file on disk after the IDS rules for this transaction have terminated, specify `Keep` in the Disposition parameter. Otherwise, the file is deleted.
NOTE: The file name used in the FileName parameter can be specified as a constant file name or as a dynamically generated file name. To use a constant file name, use a name such as:

```
c:\docserv\testr.txt
```

With a constant file name, each time the ATCReceiveFile rule runs, it will replace the contents of this file with the file that is re-assembled from the attachments. This approach is useful when developing or debugging.

To indicate that you want the rule to generate a unique name each time the rule is run, specify an asterisk (*) in the path name. The rule then generates a 45-character unique name and replaces the asterisk with that name. For example, if you specify a dynamically generated file name such as this:

```
c:\docserv\*.txt
```

the ATCReceiveFile rule generates a file name similar to this:

```
c:\docserv\0ypCmGu3koAfeD7E-ia_8yYxgfB1aybcSBIYihTqManZ.txt
```

To debug the receiving of files as attachments, use this INI option:

```
< Debug >
Attachments = Yes
```

The debug or trace information produced by specifying the Attachment option looks something like this:

```
286. ATCReceiveFile: entered,
pszParms=<ZZZ,IMPORTFILE,c:\docserv\testr.txt,keep>
287. ATCReceiveFile: Constructed filename=<c:\docserv\testr.txt>
288. ATCAttachment2File: entered,
pszFileName=<c:\docserv\testr.txt>, pszAttachName=<ZZZ>, ulOptions=<TEXT>
289. ATCAttachment2File: For attachment <ZZZ>,szFileType=<TEXT>, szNumRecs=<3>
290. ATCAttachment2File: Successful, created <c:\docserv\testr.txt>
291. ATCReceiveFile: Successful, Attachment <IMPORTFILE=c:\docserv\testr.txt> added to Attachment List.
...
```

Because it degrades performance, be sure to turn off the Attachments option after you finish debugging attachment processing.

Example

Here is an example:

```
< ReqType:T1 >
function = atcw32->ATCReceiveFile,ZZZ,IMPORTFILE,c:\docserv\testr.txt,KEEP
```

The specified file name (c:\docserv\testr.txt) is opened for write mode and text format. Once the rule writes the contents of the three attachment variables to the file, it closes the file.
Additionally, the file name is placed into the attachment variable you specified in the AttachmentVariable parameter. If you specify the Disposition parameter *Keep*, the file is kept on disk even after the rules for this transaction have terminated. This option can be useful for debugging.

**Returns**  
Success or fail

**Errors**  
Various

**See also**  
ATCSendFile on page 233
ATCSendFile

Use this rule to post a file in segments to the output attachment and send it over the IDS queue. The ATCReceiveFile rule or the DSIReceiveFile API can then re-assemble the file from the input attachment and save it. The file can be binary or text.

NOTE: Each IDS rule has a run forward and a run reverse step. The run forward step usually contains most of the functionality. The run reverse step usually re-initializes variables in preparation for the next request. The ATCSendFile function, however, does more in its reverse run than in its forward run, including sending the file.

When a request is used in IDS, all the forward run code runs (from the first rule in the list until the last); then, the reverse run takes place — all functions are considered again for any back out procedures. So, the reverse run for the ATCSendFile takes place after the forward run or RunRP rules.

Syntax

```c
long _DSIAPI ATCSendFile ( DSIHANDLE hInstance,
    char * pszParms,
    unsigned long ulMsg,
    unsigned long ulOptions )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIHANDLE hInstance</td>
<td>DSI instance handle</td>
</tr>
<tr>
<td>char * pszParms</td>
<td>Pointer to rule parameter string</td>
</tr>
<tr>
<td>unsigned long ulMsg</td>
<td>DSI_MSG message, such as DSI_MSGRUNF</td>
</tr>
<tr>
<td>unsigned long ulOptions</td>
<td>Options</td>
</tr>
</tbody>
</table>

This rule accepts three parameters (Prefix, Attachment Variable, and FileType), delimited with commas, and specified immediately following the rule in the INI file.

The file name indicated in the Attachment Variable parameter is opened in read mode based on the FileType parameter (text or binary).

You can use the ATCReceiveFile rule to write the file to disk.

Keep in mind that this rule removes the attachment variable named in its second parameter and does not work with the default queues.

The prefix name is an important parameter and it has to match when the file is being received. The format of the message and how the file data is stored in the message is described in the message layout chapter.

If you need to debug the sending of files as attachments, include this INI option:

```
< Debug >
Attachments = Yes
```

The debug or trace information produced by the Attachments option will look something like this:
Chapter 4

DSI Processing Rules

... 9. ATCSendFile: entered, pszParms=<ZZZ,IMPORTFILE,TEXT>

10. ATCFile2Attachment: entered, pszFileName=<c:\docserv\client\test.txt>, pszAttachName=<ZZZ>, ulOptions=<TEXT>

11. ATCFile2Attachment: Successful, added Attachment Variable <ZZZ=;TEXT;3;>

12. ATCSendFile: Successful, Attachment Variable <IMPORTFILE> removed from Attachment List.

...  

Because it degrades performance, be sure to turn off the Attachments option after you finish debugging attachment processing.

Example
< ReqType:T1 >
    function = atcw32->ATCSendFile,ZZZ,IMPORTFILE,TEXT

In this example, suppose the attachment variable named IMPORTFILE contains this value:
    c:\docserv\client\test.txt

This file is added to the IDS message for later use for posting to the IDS queue.

Returns  Success or fail

Errors    Various

See also  ATCReceiveFile on page 230
ATCSendMultipleFiles

Use the ATCSendMultipleFiles rule to send multiple files as queue attachments.

Syntax

```c
long _DSIAPI ATCSendMultipleFiles ( DSIHANDLE hInstance,
                                   char * pszParms,
                                   unsigned long ulMsg,
                                   unsigned long ulOptions )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIHANDLE hInstance</td>
<td>DSI instance handle</td>
</tr>
<tr>
<td>char * pszParms</td>
<td>Pointer to rule parameter string</td>
</tr>
<tr>
<td>unsigned long ulMsg</td>
<td>DSI_MSG???? message, such as DSI_MSGRUNF</td>
</tr>
<tr>
<td>unsigned long ulOptions</td>
<td>Options</td>
</tr>
</tbody>
</table>

This rule accepts the name of the attachment variable that contains the name of the file you want to send. The system uses partial name matching so if this parameter is provided as FILETOSEND, the following attachment variables will be used to find the file names to send:

```c
FILETOSEND, FILETOSEND1, FILETOSENDABC
```

The name of the file without an extension and path is used as the attachment delimiter.

The rule also accepts the type (binary or text) to use for sending all files. No individual file type can be provided, as all are handled as the same type. The default is binary because this rule is used to send multiple PNG/JPG files created during HTML generation.

This rule does not remove the attachment variables with original file names.

This rule is executed on the RUNR message.

Example

Here is an example:

```c
function=atcw32->ATCSendMultipleFiles,FILETOSEND
```
ATCUnloadAttachment

Use this rule to convert the attachment from internal format into the queue attachment format in the output queue on the DSI_MSGRUNR message. This rule makes sure the attachment name is present in the queue record. If this name is empty, this rule fills it in with the unique name on the DSI_MSGINIT message. Use this rule to free allocated memory for the internal format in the output queue on the DSI_MSGTERM message. The reserved request type DEFAULT sets this rule.

Syntax
long _DSIAPI ATCUnloadAttachment ( DSIHANDLE hInstance,
char * pszParms,
unsigned long ulMsg,
unsigned long ulOptions )

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIHANDLE hInstance</td>
<td>DSI instance handle</td>
</tr>
<tr>
<td>char * pszParms</td>
<td>Pointer to rule parameter string</td>
</tr>
<tr>
<td>unsigned long ulMsg</td>
<td>DSI_MSG???? message, such as DSI_MSGRUNF</td>
</tr>
<tr>
<td>unsigned long ulOptions</td>
<td>Options</td>
</tr>
</tbody>
</table>

This rule uses these INI options to determine the location of attachments:

< ResultQ >
AttachmentPath =

The default DOCCLNT.INI file sets this rule with these INI options:

< ReqType: Default >
Function = atcw32->ATCUnloadAttachment

The default DOC:

< ReqType: SCS >
Function = atcw32->ATCUnloadAttachment

The default DOCSERV configuration file sets this rule with these INI options:

< ReqType: SSS >
Function = atcw32->ATCLoadAttachment
NOTE: To calculate the time spent in the queue, IDS returns the ServerTimeSpent attachment variable on every transaction. The value returned is in a form of seconds.milliseconds.

The difference between this value and the TotalTimeSpent attachment variable created by the client is the **queuing latency**, which gives you an indication of how much time a transaction spent in the queue.

The ATCUnloadAttachment rule creates the attachment to be sent back, so the ServerTimeSpent value is put into that attachment. If there are any rules in the list executed after the ATCUnloadAttachment rule on RUNR message, their time is not included. Nor is the time spent on the TERM message included. The rules executed after the ATCUnloadAttachment rule on the RUNR message are the rules listed before this rule in the DOCSERE configuration file.

<table>
<thead>
<tr>
<th>Returns</th>
<th>Success or failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Errors</td>
<td>Memory error</td>
</tr>
<tr>
<td>See also</td>
<td>ATCLoadAttachment on page 227</td>
</tr>
</tbody>
</table>
IRCIInit

Use this rule to initialize a client.

**Syntax**

```c
long _DSIAPI IRCInit ( DSIHANDLE hInstance,
                    char * pszParms,
                    unsigned long  ulMsg,
                    unsigned long  ulOptions )
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIHANDLE hInstance</td>
<td>DSI instance handle</td>
</tr>
<tr>
<td>char * pszParms</td>
<td>Pointer to rule parameter string</td>
</tr>
<tr>
<td>unsigned long ulMsg</td>
<td>DSI_MSG???? message, such as DSI_MSGRUNF</td>
</tr>
<tr>
<td>unsigned long ulOptions</td>
<td>Options</td>
</tr>
</tbody>
</table>

The default DOCCNT.INI file sets this rule with this INI option.

```
< ReqType:INI >
  Function = ircltw32->IRCInit
```

**Returns**

Success or failure

**Errors**

None
**IRCPrint**

Use this rule to locate the print file created by the Internet Document Server.

**Syntax**

```c
long _DSIAPI IRCPrint ( DSIHANDLE hInstance,
                      char * pszParms,
                      unsigned long  ulMsg,
                      unsigned long  ulOptions )
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIHANDLE hInstance</td>
<td>DSI instance handle</td>
</tr>
<tr>
<td>char * pszParms</td>
<td>Pointer to rule parameter string</td>
</tr>
<tr>
<td>unsigned long ulMsg</td>
<td>DSI_MSG???? message, such as DSI_MSGRUNF</td>
</tr>
<tr>
<td>unsigned long ulOptions</td>
<td>Options</td>
</tr>
</tbody>
</table>

This rule gets the REMOTEPRINTFILE attachment variable and creates a PRINTFILE attachment variable. The rule mainly translates the file name from the file name on the server, to the file name for a front-end client.

**Returns**

Success or failure

**Errors**

Failed to locate necessary attachment variable
IRCRequest

Use this rule to prepare a request for the Internet Document Server.

Syntax

```c
long _DSI_API IRCRequest ( DSIHANDLE hInstance,
    char * pszParms,
    unsigned long ulMsg,
    unsigned long ulOptions )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIHANDLE hInstance</td>
<td>DSI instance handle</td>
</tr>
<tr>
<td>char * pszParms</td>
<td>Pointer to rule parameter string</td>
</tr>
<tr>
<td>unsigned long ulMsg</td>
<td>DSI_MSG??? message, such as DSI_MSGRUNF</td>
</tr>
<tr>
<td>unsigned long ulOptions</td>
<td>Options</td>
</tr>
</tbody>
</table>

This rule checks for the REQTYPE and USERID in the attachment and sets the fields into the request (output) queue. This rule also fills in the unique name in the request queue.

This rule only responds to the DSI_MSGRUNF message.

The default DOCCNT.INI file sets this rule with these INI settings.

```
< ReqType: Default >
    Function = ircltw32->IRCRquest
```

Returns  
Success or failure

Errors

- Necessary attachment variable missing
- DQM failure

See also  
IRCRresult on page 241
IRCResult

Use this rule to retrieve a result for the Internet Document Server and prepares the result for the client.

**Syntax**

```c
long __DSI_API IRCResult ( DSIHANDLE hInstance,
                           char * pszParms,
                           unsigned long ulMsg,
                           unsigned long ulOptions )
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIHANDLE hInstance</td>
<td>DSI instance handle</td>
</tr>
<tr>
<td>char * pszParms</td>
<td>Pointer to rule parameter string</td>
</tr>
<tr>
<td>unsigned long ulMsg</td>
<td>DSI_MSG???? message, such as DSI_MSGRUNF</td>
</tr>
<tr>
<td>unsigned long ulOptions</td>
<td>Options</td>
</tr>
</tbody>
</table>

This rule checks results returned by the server. It checks for the RESULTS attachment value in the result (input) queue attachment. If this value is not found or is not equal to SUCCESS, the rule creates an attachment variable called ERROR and a value that matches the value of the RESULTS variable. This lets you work with the ERRORS.HTM template.

**NOTE:** If you have created your own rules and are using only the IRCUnloadPage base rule, which processes the HTML template, you do not need this rule in the rule list.

This rule only responds to the DSI_MSGRUNF message.

The default DOCCNT.INI file sets this rule with these INI settings.

```ini
< ResType:Default >
   Function = ircltw32->IRCResult
</ResType:Default>
```

**Returns**

Success or failure

**Errors**

Necessary attachment variable missing

**See also**

IRCRequest on page 240
IRCSendVersion

Use this rule to report DLL version information.

Syntax

```c
long _DSIAPI IRCSendVersion ( DSIHANDLE hInstance,
                          char * pszParms,
                          unsigned long  ulMsg,
                          unsigned long  ulOptions )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIHANDLE hInstance</td>
<td>DSI instance handle</td>
</tr>
<tr>
<td>char * pszParms</td>
<td>Pointer to rule parameter string</td>
</tr>
<tr>
<td>unsigned long ulMsg</td>
<td>DSI_MSG???? message, such as DSI_MSGRUNF</td>
</tr>
<tr>
<td>unsigned long ulOptions</td>
<td>Options</td>
</tr>
</tbody>
</table>

For each of the following DLLs, this rule creates attachment variables on the DSI_MSGRUNF message.

- ATC
- DCB
- IRP
- DQM
- IBASE
- DSI
- DSIJ

Here is a list of the variables:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Tells you the...</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>name of the DLL</td>
</tr>
<tr>
<td>VERSION</td>
<td>version of the DLL, such as 100.012.XXX</td>
</tr>
<tr>
<td>DATE</td>
<td>date of the last compile in MMM DD YYYY format</td>
</tr>
<tr>
<td>TIME</td>
<td>time of the last compile in HH:MM:SS format</td>
</tr>
</tbody>
</table>

These values only change when you upgrade to a newer version.

The default DOCCNT.INI file sets this rule with these INI settings.

```ini
< ReqType:SCS >
  Function = ircltw32->IRCSendVersion
</ ReqType:SCS >
```

Returns  
Success or failure
Errors | None
---|---
See also | [IRLSendVersion on page 206](#)
IRCUnloadPage

Use this rule to unload an HTML page.

Syntax

```c
long _DSIAPI IRCUnloadPage ( DSIHANDLE hInstance,
   char * pszParms,
   unsigned long  ulMsg,
   unsigned long  ulOptions )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIHANDLE hInstance</td>
<td>DSI instance handle</td>
</tr>
<tr>
<td>char * pszParms</td>
<td>Pointer to rule parameter string</td>
</tr>
<tr>
<td>unsigned long ulMsg</td>
<td>DSI_MSG???? message, such as DSI_MSGRUNF</td>
</tr>
<tr>
<td>unsigned long ulOptions</td>
<td>Options</td>
</tr>
</tbody>
</table>

Uses HTML setting in DOCCLNT.INI file for configuration settings. Refer to Chapter 3 in the Internet Document Server Guide for an explanation of template variables and their replacement by attachment variables.

The default DOCCNT.INI file sets this rule with these INI settings.

```ini
< RestType:Default >
   Function = ircltw32->IRCUnloadPage
```

Returns

Success or failure

Errors

Missing INI values
File I/O error
Memory error
Chapter 5

DSI Visual Basic APIs

Users of the DSI Visual Basic (VB) API are expected to fall into one of these groups:

- Fat client applications written in VB or VBA
- ASP ActiveX components
- VB rules

Fat clients should start with a call to InitSession and end with a call to TermSession. The general work flow will be to build a request into one or more attachment lists which are submitted to IDS by a call to Submit.

When the server has completed its work the results will be processed with calls to GetAttachmentAll, GetAttachRecSet, GetAttachVarSet, or (occasionally) LocateAttachVar. Testing and debugging will be easier with DSICoTB than the IDE because the attachment lists can be changed with the click of a mouse and the edit/compile/test cycle is minimized.

ASP ActiveX components are structured differently. The Visual Basic object should be created in a GLOBAL.ASA file and not be new’d in the ActiveX component. InitSession either should be called in OnStartPage and the instance handle returned by InitSession, either...

- Kept in the Session object or
- TermSession called in OnEndPage.

The instance handle should not be kept in the application object as IIS multi-threads every session and the instance handle must be thread-specific.

VB rules are subject to the same conditions as other rules. Certain methods should not be called, such as InitSession, and the rules should be stateless.
**NOTE:** COM and ActiveX are designed to be language independent—the VB API class can be called from Visual Basic, Visual J++, C, C++, VBA (Visual Basic for Applications), or VBScript.

Nonetheless, it is expected that most, if not all users, will be using Visual Basic. With that in mind, prototypes and examples are targeted toward these languages.

Developers using other languages such as C++ are most likely used to this kind of discrimination and know how to adapt. For instance, COM always returns an HRESULT but VB handles this silently. If there is value returned from a method, VB silently extracts it from the argument list; C++ users must handle this explicitly.

Here are some examples:

In VB

```vb
Dim lRet as long
lRet = oDSI.FindInQueue (hInstance, dsiINPUTQUEUE, "TROUT")
```

In C++

```c++
HRESULT hr;
long lRet;
hr = spDSI->FindInQueue
    (hInstance, dsiINPUTQUEUE, BSTR(L"TROUT"), &lRet);
```

**Handling Errors**

In VB

For subs, an error may be raised for any condition that prevents normal completion, so On Error routines are very important.

For methods, the return code usually indicates a not found (dsiERR_NOTFOUND) or end-of-file (dsiERR_EOF) condition and should always be checked. But for fatal errors or any condition that prevents normal completion, an error will be raised, so On Error routines are also very important.

In C++

Exceptions are not passed across COM interfaces: the HRESULT will tell you if IErrorInfo should be interrogated. If the method provides a return code, it will generally indicate an algorithmic error, such as dsiERR_NOTFOUND; in this case, the HRESULT will also have the DSI error code in the lower two bytes.
**USING THE WEB SERVICES EXAMPLE**

The system includes a web services example which uses VB 6.0 DLL (DP018.dll) to communicate with a remote IDS via MQSeries APIs and SOAP attachments built with Microsoft's Imessage Interface.

There are two versions of this DLL file, a server version for MQSeries Server and a client version for MQSeries Client.

The MQSeries and XML APIs will work on Windows NT 4.0 and Windows 2000 Server. The SOAP APIs will only work on Windows 2000 since Microsoft's Imessage interface is only supported on Windows 2000 at this time. The demo resides on a Windows 2000 Server.
Here is a list of Visual Basic methods, grouped by functional area. Following this list is a discussion of each method, listed in alphabetical order.

**NOTE:** These methods are only available on Windows 32-bit platforms.

**Client methods**

Use these methods for writing a client program:

- AddToQueue on page 256
- AttachList on page 265
- CopyQRecord on page 269
- FindInQueue on page 281
- GetAttachment on page 282
- GetAttachmentAll on page 284
- GetAttachRecSet on page 286
- GetAttachVarSet on page 288
- GetQField on page 291
- GetQFieldLength on page 292
- GetQueueRec on page 293
- Init on page 301
- InitInstance on page 302
- InitQueue on page 303
- InitSession on page 304
- ParseAttachment on page 312
- SetAttachment on page 314
- SetQField on page 317
- StoreAttachment on page 322
- Submit on page 323
- Term on page 325
- TermInstance on page 326
- TermQueue on page 327
- TermSession on page 328
- Trace on page 329
- TraceAttach on page 330
### Server methods

Use these methods for writing rules on the server:

- ErrorMessage on page 280
- GetUserID on page 300

### Common methods

Use these methods for both the client and server:

- AddAttachRec on page 251
- AddAttachVar on page 253
- AttachCursorFirst on page 257
- AttachCursorLast on page 258
- AttachCursorName on page 260
- AttachCursorNext on page 261
- AttachCursorPrev on page 262
- AttachCursorValue on page 264
- CacheFile on page 266
- CloseAttachCursor on page 267
- CopyAttachVars on page 268
- CreateValue on page 270
- CreateValueObj on page 272
- DeleteAttachVar on page 274
- DestroyValue on page 275
- DestroyValueObj on page 277
- DumpDebugInfo on page 279
- GetPriority on page 290
- GetReqType on page 295
- GetStatus on page 296
- GetUniqueID on page 297
- GetUniqueIDLength on page 298
- GetUniqueString on page 299
- LocateAttachVar on page 306
- LocateValue on page 307
- LocateValueObj on page 309
- OpenAttachCursor on page 311
- QueryValueSize on page 313
• SetPriority on page 316
• SetReqType on page 318
• SetStatus on page 319
• SetUniqueID on page 320
• SetUserID on page 321
• TraceEnableRule on page 331
• TraceList on page 332
• TraceSnapshot on page 333

Properties
You can also use these properties:
• Property Signature on page 335
• Property TraceEnable on page 336
• Property TracePath on page 337
AddAttachRec

Use this method to create a stem variable in the attachment list.

**Syntax**

```plaintext
AddAttachRec(hInstance as Long, DSIQUEUE QueueID, RecName as String, NewVarName as String)
```

IDS supports records within an attachment. For instance, the following might be returned from a rule:

- `FISH1.TYPE      BASS`
- `FISH1.SIZE      LARGE`
- `FISH1.STATUS    CAUGHT`
- `FISH1.LOCATION  BOAT`

Using `AddAttachRec`, the stem variable that can be created by this call is `FISH`. `FISH1` is returned because it is the first `FISH` record in the attachment. You do not have to do anything else to create a stem variable. The output of an SSS request is a stem variable.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either <code>dsiINPUTQUEUE</code> or <code>dsiOUTPUTQUEUE</code>.</td>
</tr>
<tr>
<td>RecName</td>
<td>The record to add the stem variable.</td>
</tr>
<tr>
<td>RecID</td>
<td>The record ID with a variable number, such as <code>RECORD2</code>.</td>
</tr>
</tbody>
</table>

**See also**

- AddToAttachRec on page 254
- GetAttachRecSet on page 286

**Example**

From the CSamAPI.cls file in the DSICoSAM example:

```plaintext
oDSI.AddAttachRec hInstance, dsiOUTPUTQUEUE, "FISH", sBuf

' Next we want to supply the values. To do this we use the add to attach record functionality. We supply the buffer returned from or earlier add attach record call.

' Add name of my DLL. SBuf should be "FISH1" at this point
oDSI.AddToAttachRec hInstance, dsiOUTPUTQUEUE, sBuf, "TYPE", "BASS"

' Add date DLL was built
oDSI.AddToAttachRec hInstance, dsiOUTPUTQUEUE, sBuf, "SIZE", "LARGE"

' Add time DLL was built
oDSI.AddToAttachRec hInstance, dsiOUTPUTQUEUE, sBuf, "STATUS", "CAUGHT"

' Add my DLL version number
oDSI.AddToAttachRec hInstance, dsiOUTPUTQUEUE, sBuf, "LOCATION", "BOAT"
```
' Put the attachment into the queue record
odSI.StoreAttachment hInstance, dsiOUTPUTQUEUE
AddAttachVar

Use this method to add name/value pair to an attachment.

**Syntax**

```
AddAttachVar(hInstance as Long, QueueID as DSIQUEUE, Name as String, Value as String)
```

**NOTE:** An empty Value is allowed. An empty Name is not.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
<tr>
<td>Name</td>
<td>A variable name.</td>
</tr>
<tr>
<td>Value</td>
<td>A variable value.</td>
</tr>
</tbody>
</table>

**See also**

- LocateAttachVar on page 306
- DeleteAttachVar on page 274
- GetAttachmentAll on page 284
- GetAttachVarSet on page 288

**Example**

From the CSamAPI.cls file in the DSICoSAM example:

```vbnet
ODSI.AddAttachVar hInstance, _
    dsiOUTPUTQUEUE, _
    "Hello", _
    "Hello World!"

ODSI.AddAttachVar hInstance, _
    dsiOUTPUTQUEUE, _
    "Good-bye", _
    "Good-bye World!"

ODSI.StoreAttachment hInstance, dsiOUTPUTQUEUE
```
AddToAttachRec

Use this method to append a value to a stem variable

Syntax
AddToAttachRec(hInstance as Long, QueueID as DSIQUEUE, RecName as String, Name as String, Value as String)

IDS supports records within an attachment. For instance, the following might be returned from a rule:

FISH1.TYPE       BASS
FISH1.SIZE       LARGE
FISH1.STATUS    CAUGHT
FISH1.LOCATION  BOAT

To add to the FISH1 record,

AddToAttachRec (hInstance, dsiOUTPUTQUEUE, "ANGLER", "Mom")

FISH1.TYPE       BASS
FISH1.SIZE       LARGE
FISH1.STATUS    CAUGHT
FISH1.LOCATION  BOAT
FISH1.ANGLER    Mom

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
<tr>
<td>RecName</td>
<td>The record to which variable should be added.</td>
</tr>
<tr>
<td>Name</td>
<td>The name of the field within the record.</td>
</tr>
<tr>
<td>Value</td>
<td>The data associated with the variable.</td>
</tr>
</tbody>
</table>

See also
AddAttachRec on page 251
GetAttachRecSet on page 286

Example
From the CSamAPI.cls file in the DSICoSAM example:

```vbscript
oDSI.AddAttachRec hInstance, dsiOUTPUTQUEUE, "LIBRARIES", sBuf

' Next we want to supply the values. To do this we use the add to attach record functionality. We supply the buffer returned from or earlier add attach record call.

' Add name of my DLL
oDSI.AddToAttachRec hInstance, dsiOUTPUTQUEUE, sBuf, "NAME", "DSRVRLVB"

' Add date DLL was built
oDSI.AddToAttachRec hInstance, dsiOUTPUTQUEUE, sBuf, "DATE", "Date"

' Add time DLL was built
```
oDSI.AddToAttachRec hInstance, dsiOUTPUTQUEUE, sBuf, "TIME", "time"

' Add my DLL version number
oDSI.AddToAttachRec hInstance, dsiOUTPUTQUEUE, sBuf, "VERSION", "1.0"

' Put the attachment into the queue record
oDSI.StoreAttachment hInstance, dsiOUTPUTQUEUE
AddToQueue

Use this method to release a record into the queue for processing. Nothing happens on the server until you make this call—or instead use the Submit method.

Syntax

AddToQueue(hInstance as Long, QueueID as DSIQUEUE)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
</tbody>
</table>

See also

GetQueueRec on page 293

StoreAttachment on page 322

Submit on page 323

Example

From the CSamAPI.cls file in the DSICoSAM example:

```vbnet
' put in our attachment
oDSI.AddAttachVar hInstance, dsiOUTPUTQUEUE, "What", "Me Worry?"

' move our attachment from the buffer into the record
oDSI.StoreAttachment hInstance, dsiOUTPUTQUEUE

' set the echo request type
oDSI.SetQField hInstance, dsiOUTPUTQUEUE, dsiQSET_REQTYPE, "ECH"

oDSI.UserID = "DocExample"

' if sUnique is empty, SetUniqueID will fill it in for us
oDSI.SetUniqueID hInstance, dsiOUTPUTQUEUE, sUnique

' release the queue record for processing
oDSI.AddToQueue hInstance, dsiOUTPUTQUEUE
```
**AttachCursorFirst**

Use this method to recover the first name/value pair in the attachment and position the cursor on the next pair.

**Syntax**

```
AttachCursorFirst(hCursor as Long, Name as String, Value as String) as Long
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCursor</td>
<td>the cursor obtained from OpenAttachCursor</td>
</tr>
<tr>
<td>Name</td>
<td>returned name</td>
</tr>
<tr>
<td>Value</td>
<td>returned value</td>
</tr>
</tbody>
</table>

**Returns**

- DSIERR_SUCCESS
- DSIERR_NOTFOUND

**See also**

- AttachCursorLast on page 258
- AttachCursorName on page 260
- AttachCursorNext on page 261
- AttachCursorPrev on page 262
- AttachCursorValue on page 264
- CloseAttachCursor on page 267
- OpenAttachCursor on page 311
- ParseAttachment on page 312

**Example**

From the CSamAPI.cls file in the DSICoSAM example:

```vbnet
ODSI.ParseAttachment hInstance, dsiINPUTQUEUE

' Open a cursor for the attachment
' This cursor will allow us to walk through the attachment serially
hCursor = oDSI.OpenAttachCursor(hInstance, dsiINPUTQUEUE)

' Position to the first element of the attachment'
lRet = oDSI.AttachCursorFirst(hCursor, sName, sValue)

' Loop through all elements of the parsed attachment printing
' the name and value pairs and put them in the right hand list box
While lRet = dsiERR_SUCCESS
    ... do something useful
    lRet = oDSI.AttachCursorNext(hCursor, sName, sValue)
Wend

' close out the cursor to free the resources
ODSI.CloseAttachCursor hCursor
```
AttachCursorLast

Use this method to recover the last name/value pair in the attachment and retard the cursor to previous name/value pair.

Syntax

AttachCursorLast(hCursor as Long, Name as String, Value as String) as Long

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCursor</td>
<td>cursor pointing into the attachment list</td>
</tr>
<tr>
<td>Name</td>
<td>returned name</td>
</tr>
<tr>
<td>Value</td>
<td>returned value</td>
</tr>
</tbody>
</table>

Returns

DSIERR_SUCCESS
DSIERR_NOTFOUND

See also

AttachCursorFirst on page 257
AttachCursorName on page 260
AttachCursorNext on page 261
AttachCursorPrev on page 262
AttachCursorValue on page 264
CloseAttachCursor on page 267
OpenAttachCursor on page 311
ParseAttachment on page 312

Example

From the CSamAPI.cls file in the DSICoSAM example:

```
' Parse and present our results.
oDSI.ParseAttachment hInstance, dsiINPUTQUEUE

' Open a cursor for the attachment
' This cursor will allow us to walk through the attachment serially
hCursor = oDSI.OpenAttachCursor(hInstance, dsiINPUTQUEUE)

' Position to the last element of the attachment
Dim sName As String, sValue As String
Dim lRet
lRet = oDSI.AttachCursorLast(hCursor, sName, sValue)

' Loop through all elements of the parsed attachment printing
' the name and value pairs and put them in the right hand list box
While lRet = dsiERR_SUCCESS
  ... do something useful ...
  lRet = oDSI.AttachCursorPrev(hCursor, sName, sValue)
Wend
```
' Close the attachment cursor'

oDSI.CloseAttachCursor hCursor
AttachCursorName

Use this method to get the name value for the current position of the cursor.

Syntax

AttachCursorName(hCursor as Long, Name as String) as Long

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCursor</td>
<td>the cursor obtained from the OpenAttachCursor method</td>
</tr>
<tr>
<td>Name</td>
<td>returned Name</td>
</tr>
</tbody>
</table>

Returns:

- DSIERR_SUCCESS
- DSIERR_NOTFOUND

See also

- AttachCursorFirst on page 257
- AttachCursorLast on page 258
- AttachCursorNext on page 261
- AttachCursorPrev on page 262
- AttachCursorValue on page 264
- CloseAttachCursor on page 267
- OpenAttachCursor on page 311
- ParseAttachment on page 312

Example

From the CSamAPI.cls file in the DSICoSAM example:

```vbscript
' Parse the attachment in the current record
doSI.ParseAttachment hInstance, dsiINPUTQUEUE

' obtain an attachment cursor
hCursor = doSI.OpenAttachCursor(hInstance, dsiINPUTQUEUE)

' get the first name/value pair
lRet = doSI.AttachCursorFirst(hCursor, sName, sValue)

' get just the name from the name/value pair
lRet = doSI.AttachCursorName(hCursor, sName)

' get the value from the name/value pair
lRet = doSI.AttachCursorValue(hCursor, sValue)

' drop the attachment cursor
doSI.CloseAttachCursor hCursor
```
**AttachCursorNext**

Use this method to retrieve the next name/value pair from the attachment list.

**Syntax**

```
AttachCursorNext(hCursor as Long, Name as String, Value as String) as Long
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCursor</td>
<td>cursor pointing into the attachment list</td>
</tr>
<tr>
<td>Name</td>
<td>returned name</td>
</tr>
<tr>
<td>Value</td>
<td>returned value</td>
</tr>
</tbody>
</table>

**Returns:**

- DSIERR_SUCCESS
- DSIERR_NOTFOUND

**See also**

- AttachCursorFirst on page 257
- AttachCursorLast on page 258
- AttachCursorName on page 260
- AttachCursorPrev on page 262
- AttachCursorValue on page 264
- CloseAttachCursor on page 267
- OpenAttachCursor on page 311
- ParseAttachment on page 312

**Example**

From the CSamAPI.cls file in the DSICoSAM example:

```
oDSI.ParseAttachment hInstance, dsiINPUTQUEUE

' Open a cursor for the attachment
' This cursor will allow us to walk through the attachment serially
hCursor = oDSI.OpenAttachCursor(hInstance, dsiINPUTQUEUE)

' Position to the first element of the attachment
lRet = oDSI.AttachCursorFirst(hCursor, sName, sValue)

' Loop through all elements of the parsed attachment printing
' the name and value pairs and put them in the right hand list box
While lRet = dsiERR_SUCCESS
    ... do something useful
    lRet = oDSI.AttachCursorNext(hCursor, sName, sValue)
Wend

' close out the cursor to free the resources
oDSI.CloseAttachCursor hCursor
```
**AttachCursorPrev**

Use this method to retrieve the next name/value pair from the attachment list.

**Syntax**

```
AttachCursorPrev(hCursor as Long, Name as String, Value as String) as Long
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCursor</td>
<td>cursor pointing into the attachment list</td>
</tr>
<tr>
<td>Name</td>
<td>returned name</td>
</tr>
<tr>
<td>Value</td>
<td>returned value</td>
</tr>
</tbody>
</table>

**Returns:**

- DSIERR_SUCCESS
- DSIERR_NOTFOUND

**See also**

- AttachCursorFirst on page 257
- AttachCursorLast on page 258
- AttachCursorName on page 260
- AttachCursorNext on page 261
- AttachCursorValue on page 264
- CloseAttachCursor on page 267
- OpenAttachCursor on page 311
- ParseAttachment on page 312

**Example**

From the CSamAPI.cls file in the DSICoSAM example:

```vba
' Parse and present our results.
oDSI.ParseAttachment hInstance, dsiINPUTQUEUE

' Open a cursor for the attachment
' This cursor will allow us to walk through the attachment serially
hCursor = oDSI.OpenAttachCursor(hInstance, dsiINPUTQUEUE)

' Position to the last element of the attachment
Dim sName As String, sValue As String
Dim lRet
lRet = oDSI.AttachCursorLast(hCursor, sName, sValue)

' Loop through all elements of the parsed attachment printing
' the name and value pairs and put them in the right hand list box
While lRet = dsiERR_SUCCESS
    ... do something useful ...
    lRet = oDSI.AttachCursorPrev(hCursor, sName, sValue)
Wend

' Close the attachment cursor'
```
oDSI.CloseAttachCursor hCursor
AttachCursorValue

Use this method to get the value of the attachment at the current cursor position.

Syntax

AttachCursorValue(hCursor as Long, Value as String)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCursor</td>
<td>the cursor obtained from the OpenAttachCursor method</td>
</tr>
<tr>
<td>Value</td>
<td>returned value</td>
</tr>
</tbody>
</table>

See also

AttachCursorFirst on page 257
AttachCursorNext on page 261
AttachCursorLast on page 258
AttachCursorPrev on page 262
CloseAttachCursor on page 267
OpenAttachCursor on page 311
ParseAttachment on page 312

Example

From the CSamAPI.cls file in the DSICoSAM example:

```vbscript
' Parse the attachment in the current record
oDSI.ParseAttachment hInstance, dsiINPUTQUEUE

' obtain an attachment cursor
hCursor = oDSI.OpenAttachCursor(hInstance, dsiINPUTQUEUE)

' get the first name/value pair
lRet = oDSI.AttachCursorFirst(hCursor, sName, sValue)

' get just the name from the name/value pair
lRet = oDSI.AttachCursorName(hCursor, sName)

' get the value from the name/value pair
lRet = oDSI.AttachCursorValue(hCursor, sValue)

' drop the attachment cursor
oDSI.CloseAttachCursor hCursor
```
AttachList

Use this method to attach the array of name/value pairs to the queue record.

**Syntax**

```
AttachList(hInstance as Long, QueueID as DSIQUEUE, List() as String)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The session/thread handle.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
<tr>
<td>List()</td>
<td>A two-dimensional string array with a set of name/value pairs.</td>
</tr>
</tbody>
</table>

**See also**

- AddAttachVar on page 253
- GetAttachmentAll on page 284
- GetAttachVarSet on page 288
- GetAttachRecSet on page 286

**Example**

From the CSamAPI.cls file in the DSICoSAM example:

```vbnet
sAttach(0, 0) = "Name0"
sAttach(0, 1) = "Value0"
sAttach(1, 0) = "Name1"
sAttach(1, 1) = "Value1"
sAttach(2, 0) = "Name2"
sAttach(2, 1) = "Value2"
sAttach(3, 0) = "Name3"
sAttach(3, 1) = "Value3"
sAttach(4, 0) = "Name4"
sAttach(4, 1) = "Value4"

' Add the list to the attachment
oDSI.AttachList hInstance, dsiOUTPUTQUEUE, sAttach

' every queue record must have a request
oDSI.SetReqType hInstance, dsiOUTPUTQUEUE, sRequest

sUnique = ""  ' make sure we get a new one this time
oDSI.SetUniqueID hInstance, dsiOUTPUTQUEUE, sUnique

' move the attachment from the local buffer to the record
oDSI.StoreAttachment hInstance, dsiOUTPUTQUEUE

' release queue record to the queue for processing
oDSI.AddToQueue hInstance, dsiOUTPUTQUEUE
```
CacheFile

Use this method to add a file name to the cache.

Syntax

```
CacheFile(hInstance as Long, FileName as String, Expire as long)
```

**NOTE:** Only for use in rules.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>FileName</td>
<td>The name and path of the file</td>
</tr>
<tr>
<td>Expire</td>
<td>The life of the file, in seconds</td>
</tr>
</tbody>
</table>

Example

```
oDSI.CacheFile hInstance,"temp.html",20000
```
CloseAttachCursor

Use this method to close an attachment cursor and free the associated resources.

Syntax

CloseAttachCursor(hCursor as Long)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCursor</td>
<td>The cursor obtained from the OpenAttachCursor method</td>
</tr>
</tbody>
</table>

See also

AttachCursorFirst on page 257
AttachCursorLast on page 258
AttachCursorName on page 260
AttachCursorNext on page 261
AttachCursorPrev on page 262
AttachCursorValue on page 264
OpenAttachCursor on page 311
ParseAttachment on page 312

Example

From the CSamAPI.cls file in the DSICoSAM example:

```vba
  oDSI.ParseAttachment hInstance, dsiINPUTQUEUE

  ' Open a cursor for the attachment
  ' This cursor will allow us to walk through the attachment serially
  hCursor = oDSI.OpenAttachCursor(hInstance, dsiINPUTQUEUE)

  ' Position to the first element of the attachment'
  lRet = oDSI.AttachCursorFirst(hCursor, sName, sValue)

  ' Loop through all elements of the parsed attachment printing
  ' the name and value pairs and put them in the right hand list box
  While lRet = dsiERR_SUCCESS
    ... do something useful
    lRet = oDSI.AttachCursorNext(hCursor, sName, sValue)
  Wend

  ' close out the cursor to free the resources
  oDSI.CloseAttachCursor hCursor
```
CopyAttachVars

Use this method to copy all attachment variables from one queue to the other.

Syntax

CopyAttachVars(hInstance as Long, QueueID as DSIQUEUE)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
</tbody>
</table>

See also

AddAttachVar on page 253
AttachList on page 265
LocateAttachVar on page 306
DeleteAttachVar on page 274

Example

From the CSamSupp.cls file in the DSICoSAM example:

```vba
Echo = dsiERR_SUCCESS
Select Case ulMsg
  Case dsiMSG_RUNF ' Forward (ie, inbound) logic
    oDSI.AddAttachVar hInstance, dsiOUTPUTQUEUE, "RESULTS", "SUCCESS"
    oDSI.CopyAttachVars hInstance, dsiINPUTQUEUE
  Case Else ' We don’t support the other messages
    Echo = dsiERR_MSGNOTFOUND
End Select
```
**CopyQRecord**

Use this method to copy a queue record from one queue to another.

**Syntax**

```
CopyQRecord(hInstance as Long, QueueID as DSQUEUE)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
</tbody>
</table>

**Example**

```
oDSI.CopyQueueRecord hInstance, dsiOUTPUTQUEUE
```
**CreateValue**

Use this method to create a DSI persistent variable.

**Syntax**

```
CreateValue(hInstance as Long, Name as String, Value as VARIANT)
```

These variables are persistent and must be destroyed by a call to DestroyValue. They are not associated with the queues or attachments and exist to aid communication or provide state information between rules and calls to rules.

Keep in mind:

- SAFEARRAY's are not supported
- Use the CreateValueObj method with objects

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>Name</td>
<td>the name of the variable to be created</td>
</tr>
<tr>
<td>Value</td>
<td>the variable to create (can be NULL)</td>
</tr>
</tbody>
</table>

**See also**

- CreateValueObj on page 272
- LocateValue on page 307
- DestroyValue on page 275
- QueryValueSize on page 313

**Example**

From the CSamAPI.cls file in the DSICom example:

```vba
' save our string
sTestValue = "Hello World"
oDSI.CreateValue hInstance, sSTRING_TAG, sTestValue

' now get it back
Dim lRet
lRet = oDSI.LocateValue(hInstance, sSTRING_TAG, sReturnedValue)
If lRet <> dsiERR_SUCCESS _
Or sReturnedValue <> sTestValue Then
    MsgBox ("Failed")
Else
    MsgBox ("Success")
End If

' we’re through with it so we destroy it
oDSI.DestroyValue hInstance, sSTRING_TAG

' now lets see how integers fare
iTestValue = 234
oDSI.CreateValue hInstance, sINT_TAG, iTestValue
```
lRet = oDSI.LocateValue(hInstance, sINT_TAG, iReturnedValue)
If lRet <> dsiERR_SUCCESS _
Or iTestValue <> iReturnedValue Then
   MsgBox ("Failed")
Else
   MsgBox ("Success")
End If

' we're through with it so we destroy it
oDSI.DestroyValue hInstance, sINT_TAG
CreateValueObj

Use this method to create a DSI persistent variable that refers to an object (ActiveX component).

Syntax

```
CreateValueObj(hInstance as Long, Name as String, Value as Object)
```

These variables are persistent and must be destroyed by a call to DestroyValueObj. They are not associated with the queues or attachments and exist to aid communication or provide state information between rules and calls to rules.

**NOTE:** ActiveX components are referenced counted and VB is very good about its record keeping so few are even aware that it is going on. If you use this method to save a reference to an object VB will take over that responsibility as much as it can. If, however, you fail to call DestroyValueObj, even in On Error handlers, you can leave a dangling reference which can tie up resources unnecessarily and even lead to a server crash.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>Name</td>
<td>the name of the variable to be created</td>
</tr>
<tr>
<td>Value</td>
<td>the object reference to save</td>
</tr>
</tbody>
</table>

See also

- CreateValue on page 270
- DestroyValueObj on page 277
- LocateValueObj on page 309

Example

From the CSamAPI.cls file in the DSICoSAM example:

```
' Test with early bound object
oDSI.CreateValueObj hInstance, "MY_OBJECT", oTestValue

' clear our reference
Set oTestValue = Nothing

' get it back
lRet = oDSI.LocateValueObj(hInstance, "MY_OBJECT", oOtherTestValue)

' use the object to make sure we got back what we sent out
MsgBox (oOtherTestValue.TestReturn("Hello World"))

' clear our reference
Set oOtherTestValue = Nothing

' we don’t want a dangling reference
oDSI.DestroyValueObj hInstance, "MY_OBJECT"
```
' Test with late bound object

Dim oObject As Object
Dim oOtherObject As Object

Set oObject = CreateObject("Docucorp_IDS_SamTObj.CSSamTObj")
oDSI.CreateValueObj hInstance, "MY_OBJECT", oObject

' clear our reference
Set oObject = Nothing

' get it back
lRet = oDSI.LocateValueObj(hInstance, "MY_OBJECT", oOtherObject)

' use the object to make sure we got back what we sent out
MsgBox("Object #2 Recovered: " + oOtherObject.TestReturn("Hello World")

' clear our reference
Set oOtherObject = Nothing

' we don't want a dangling reference
oDSI.DestroyValueObj hInstance, "MY_OBJECT"
DeleteAttachVar

Use this method to remove an attachment variable.

Syntax

\[
\text{DeleteAttachVar}(\text{hInstance as Long, QueueID as DSIQUEUE, Name as String})
\]

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
<tr>
<td>Name</td>
<td>The name of the variable you want to delete.</td>
</tr>
</tbody>
</table>

See also

- LocateAttachVar on page 306
- AddAttachVar on page 253
- GetAttachmentAll on page 284
- GetAttachVarSet on page 288
- GetAttachRecSet on page 286

Example

From the CSamAPI.cls file in the DSICoSAM example:

```
' parse the attachment into local storage
oDSI.ParseAttachment hInstance, dsiINPUTQUEUE

' delete what we do not like
oDSI.DeleteAttachVar hInstance, dsiINPUTQUEUE, "Name0"

' make sure it worked
lRet = oDSI.LocateAttachVar(hInstance, dsiINPUTQUEUE, "Name0", sValue)

If lRet <> dsiERR_SUCCESS Then
   MsgBox ("Success: didn’t find Name0")
Else
   MsgBox ("Failure: " + Hex(lRet), "data found")
End If
```
**DestroyValue**

Use this method to destroy a DSI persistent variable.

**Syntax**

```
DestroyValue(hInstance as Long, Name as String)
```

These variables are persistent and must be destroyed by a call to this method. They are not associated with the queues or attachments and exist to aid communication or retain state between rules and calls to rules.

**NOTE:** If you do not call this routine for each call to CreateValue you will create memory leaks.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>Name</td>
<td>the name of the persistent variable to be destroyed</td>
</tr>
</tbody>
</table>

**See also**

- [CreateValue on page 270](#)
- [LocateValue on page 307](#)
- [DestroyValueObj on page 277](#)
- [QueryValueSize on page 313](#)

**Example**

From the CSamAPI.cls file in the DSICoSAM example:

```
' save our string
sTestValue = "Hello World"
ODSI.CreateValue hInstance, sSTRING_TAG, sTestValue

' now get it back
Dim lRet
lRet = ODSI.LocateValue(hInstance, sSTRING_TAG, sReturnedValue)
If lRet <> dsiERR_SUCCESS _
Or sReturnedValue <> sTestValue Then
    MsgBox ("Failed")
Else
    MsgBox ("Success")
End If

' we're through with it so we destroy it
ODSI.DestroyValue hInstance, sSTRING_TAG

' now let's see how integers fare
iTestValue = 234
ODSI.CreateValue hInstance, sINT_TAG, iTestValue

lRet = ODSI.LocateValue(hInstance, sINT_TAG, iReturnedValue)
If lRet <> dsiERR_SUCCESS _
```
Or iTestValue <> iReturnedValue Then
    MsgBox ("Failed")
Else
    MsgBox ("Success")
End If

' we're through with it so we destroy it
oDSI.DestroyValue hInstance, sINT_TAG
DestroyValueObj

Use this method to destroy a DSI persistent variable that is an object (ActiveX component).

**Syntax**

```
DestroyValueObj(hInstance as Long, Name as String)
```

These variables are persistent and must be destroyed by a call to this method. They are not associated with the queues or attachments and exist to aid communication or retain state between rules and calls to rules.

**NOTE:** ActiveX and VB objects are referenced counted and VB is very good about its record keeping so few are even aware that it is going on. If you use this method to save a reference to an object VB will take over that responsibility as much as it can. If, however, you fail to call DestroyValueObj, even in On Error handlers, you can leave a dangling reference which can tie up resources unnecessarily and perhaps even crash the server.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>Name</td>
<td>the name of the persistent variable to be destroyed</td>
</tr>
</tbody>
</table>

**See also**

- `CreateValueObj on page 272`
- `LocateValueObj on page 309`
- `DestroyValue on page 275`

**Example**

From the CSamAPI.cls file in the DSICoSAM example:

```vb
' Test with early bound object
oDSI.CreateValueObj hInstance, "MY_OBJECT", oTestValue

' clear our reference
Set oTestValue = Nothing

' get it back
lRet = oDSI.LocateValueObj(hInstance, "MY_OBJECT", oOtherTestValue)

' use the object to make sure we got back what we sent out
MsgBox (oOtherTestValue.TestReturn("Hello World"))

' clear our reference
Set oOtherTestValue = Nothing

' we don’t want a dangling reference
oDSI.DestroyValueObj hInstance, "MY_OBJECT"
```
' Test with late bound object

Dim oObject As Object
Dim oOtherObject As Object

Set oObject = CreateObject("Docucorp_IDS_SamTObj.CSamTObj")
oDSI.CreateValueObj hInstance, "MY_OBJECT", oObject

' clear our reference
Set oObject = Nothing

' get it back
lRet = oDSI.LocateValueObj(hInstance, "MY_OBJECT", oOtherObject)

' use the object to make sure we got back what we sent out
MsgBox("Object #2 Recovered: " + oOtherObject.TestReturn("Hello World"))

' clear our reference
Set oOtherObject = Nothing

' we don't want a dangling reference
oDSI.DestroyValueObj hInstance, "MY_OBJECT"
DumpDebugInfo

Use this method to get the debug information as text for diagnostic purposes. This information is also placed at various locations in the VB trace file and can be forced into the VB trace file by a call to TraceSnapshot.

**Syntax**

```vbscript
DumpDebugInfo hInstance as Long, DebugInfo () as String
```

To see the output run the DSICoDiag sample project or run DEBUG.ASP from your browser.

This method is not dependent on TraceEnable.

**NOTE:** The information returned by this method is subject to change in both content and format without notice. This information is provided to aid debugging only. If you build a program around the returned contents, you will eventually get a program that does not work.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>DebugInfo()</td>
<td>A one-dimensional string array which contains diagnostic text</td>
</tr>
</tbody>
</table>

**See also**

TraceSnapshot on page 333

**Example**

From the CSamAPI.cls file in the DSICoSAM example:

```vbscript
Dim sInfo() as String
oDSI.DumpDebugInfo hInstance, sInfo

dim i
For i = 0 To UBound(sInfo, 1)
    ListBox1.Add (sInfo(i))
Next i
```
ErrorMessage

Use this method to add an error message to an attachment. It is expected that the first element will be the error number followed by the details as name/value pairs.

Syntax

```vba
ErrorMessage(hInstance as Long, QueueID as DSIQUEUE, ErrorMsg () as String)
```

This method is most commonly called in rules.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
<tr>
<td>ErrorMsg ()</td>
<td>A one-dimensional array which consists of the error message followed by name/value pairs.</td>
</tr>
</tbody>
</table>

See also  
[StoreAttachment on page 322](#)

Example

From the CSamAPI.cls file in the DSICoSAM example:

```vba
Dim sMsg(0 To 2) As String
sMsg(0) = "SAM001"
sMsg(1) = "FileName"
sMsg(2) = "lostinspace.dat"

' put our error into the queue
oDSI.ErrorMessage hInstance, dsiOUTPUTQUEUE, sMsg

' this is not necessary in a rule
oDSI.StoreAttachment hInstance, dsiOUTPUTQUEUE
```
FindInQueue

Use this method to search for a record in a queue. FindInQueue is the same as GetQueueRec except that FindInQueue does not wait.

Syntax

```
FindInQueue(hInstance as Long, QueueID as DSIQUEUE, UniqueID as String) as Long
```

If the queue record is not immediately available it will return DSIERR_NOTFOUND and you can try again at a later time.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
<tr>
<td>UniqueID</td>
<td>The search target.</td>
</tr>
</tbody>
</table>

Returns

- DSIERR_SUCCESS
- DSIERR_NOTFOUND

See also

GetQueueRec on page 293

Example

```
Dim ctLook
lRet = dsiERR_NOTFOUND
While lRet <> dsiERR_SUCCESS _
    And ctLook < 10000
        lRet = oDSI.FindInQueue(hInstance, dsiINPUTQUEUE, sUnique)
        DoEvents
        ctLook = ctLook + 1
    Wend
```
Chapter 5
DSI Visual Basic APIs

GetAttachment

Use this method to get the unparsed attachment for the current queue record. Since attachments can be quite large, expect a very long string.

**Syntax**

```
GetAttachment(hInstance as Long, QueueID as DSIQUEUE, Attachment as String)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
<tr>
<td>Attachment</td>
<td>The returned attachment.</td>
</tr>
</tbody>
</table>

**See also**

- DeleteAttachVar on page 274
- GetAttachmentAll on page 284
- GetAttachVarSet on page 288
- GetAttachRecSet on page 286
- LocateAttachVar on page 306
- ParseAttachment on page 312

**Example**

```
Dim sAttach(0 To 4, 0 To 1) As String
sAttach(0, 0) = "Name0"
sAttach(0, 1) = "Value0"
sAttach(1, 0) = "Name1"
sAttach(1, 1) = "Value1"
sAttach(2, 0) = "Name2"
sAttach(2, 1) = "Value2"
sAttach(3, 0) = "Name3"
sAttach(3, 1) = "Value3"
sAttach(4, 0) = "Name4"
sAttach(4, 1) = "Value4"

' put all of these name/value pairs in the attachment
oDSI.AttachList hInstance, dsiOUTPUTQUEUE, sAttach

' set up the echo request
oDSI.SetReqType hInstance, dsiOUTPUTQUEUE, "ECH"

oDSI.SetUniqueID hInstance, dsiOUTPUTQUEUE, sUnique

' move the attachment from local storage to the queue record
oDSI.StoreAttachment hInstance, dsiOUTPUTQUEUE

' release the record to the queue
oDSI.AddToQueue hInstance, dsiOUTPUTQUEUE

' recover the attachment echoed back to us
```
GetAttachment

oDSI.GetQueueRec hInstance, dsiINPUTQUEUE, sUnique

' get the unparsed attachment
oDSI.GetAttachment hInstance, dsiINPUTQUEUE, sAttach

text.Caption = sAttach
**GetAttachmentAll**

Use this method to return the entire parsed attachment as a two-dimensioned array of name/value pairs.

**Syntax**

```vbnet
GetAttachmentAll(hInstance as Long, QueueID as DSIQUEUE, Attach() as String)
```

**NOTE:** Do not call the ParseAttachment method before you call this method.

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
<tr>
<td>Attach()</td>
<td>A two-dimensional array with the attachment name/value pairs.</td>
</tr>
</tbody>
</table>

### See also

- DeleteAttachVar on page 274
- GetAttachment on page 282
- GetAttachRecSet on page 286
- GetAttachVarSet on page 288
- LocateAttachVar on page 306
- ParseAttachment on page 312

### Example

From the CSamAPI.cls file in the DSICoSAM example:

```vbnet
sAttach1(0, 0) = "Name0"
sAttach1(0, 1) = "Value0"
sAttach1(1, 0) = "Name1"
sAttach1(1, 1) = "Value1"
sAttach1(2, 0) = "Name2"
sAttach1(2, 1) = "Value2"
sAttach1(3, 0) = "Name3"
sAttach1(3, 1) = "Value3"
sAttach1(4, 0) = "Name4"
sAttach1(4, 1) = "Value4"
s
sAttach2(0, 0) = "Name20"
sAttach2(0, 1) = "Value20"
sAttach2(1, 0) = "Name21"
sAttach2(1, 1) = "Value21"
sAttach2(2, 0) = "Name22"
sAttach2(2, 1) = "Value22"
sAttach2(3, 0) = "Name23"
sAttach2(3, 1) = "Value23"
sAttach2(4, 0) = "Name24"
sAttach2(4, 1) = "Value24"
```
' send the attachment to the server with the request it be echoed back
sUnique = "" ' to get us a new UniqueID
oDSI.Submit hInstance, "ECH", sUnique, sAttach1, sAttach2

' wait for the server to return the attachment
oDSI.GetQueueRec hInstance, dsiINPUTQUEUE, sUnique, 1000, nTIMEOUT

' get the attachment into an array
oDSI.GetAttachmentAll hInstance, dsiINPUTQUEUE, sAttachIn

For i = LBound(sAttachIn, 1) To UBound(sAttachIn, 1)
    MsgBox (sAttachIn(i, 0) &": " & sAttachIn(i, 1))
Next i
GetAttachRecSet

Use this method for attachments which consist of a series of variables (RECORD1, RECORD2, and so on) with stem variables. The paradigm is that of a series of structures or records so this method recovers the record set as a matrix. The top row in the matrix contains the variable names, like in a spreadsheet.

Syntax

GetAttachRecSet (hInstance as Long, QueueID as DSIQUEUE, RecBase as String, Vars() as String, _
Optional Headings as Boolean, _
Optional FirstRec as Long, Optional LastRec as Long)

IDS supports records within an attachment. For instance, the following might be returned from a rule:

FISH1.TYPE      BASS
FISH1.SIZE      LARGE
FISH1.STATUS    CAUGHT
FISH1.LOCATION  BOAT
FISH2.TYPE      GUPPY
FISH2.SIZE      TINY
FISH2.STATUS    RETURNED
FISH2.LOCATION  LAKE
FISH3.TYPE      SHARK
FISH3.SIZE      LARGE
FISH3.STATUS    APPROACHING
FISH3.LOCATION  CLOSE!

Calling this method will return:

TYPE    SIZE      STATUS        LOCATION
BASS    LARGE     CAUGHT        BOAT
GUPPY   TINY      RETURNED      LAKE
SHARK   LARGE     APPROACHING   CLOSE!

**NOTE:** You must use the ParseAttachment method before you call this method. You can optionally specify the range of records to be extracted from the attachment.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle (from the server if invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
<tr>
<td>RecBase</td>
<td>The record identification string (such as FISH).</td>
</tr>
<tr>
<td>Vars</td>
<td>The output array.</td>
</tr>
<tr>
<td>Titles</td>
<td>(Optional) Include record names as column headings. The default is TRUE.</td>
</tr>
<tr>
<td>FirstRec</td>
<td>(Optional) The first record to recover. The default is one (1).</td>
</tr>
</tbody>
</table>
GetAttachRecSet

Returns:

- **DSIERR_SUCCESS**
- **DSIERR_NOTFOUND**

See also:
- AddToAttachRec on page 254
- AttachCursorFirst on page 257
- AttachCursorLast on page 258
- AttachCursorNext on page 261
- AttachCursorPrev on page 262
- GetAttachmentAll on page 284
- ParseAttachment on page 312

Example:

From the CSamAPI.cls file in the DSICoSAM example:

```vbscript
' wait for the server to return the attachment
oDSI.GetQueueRec hInstance, dsiINPUTQUEUE, sUnique

' parse the attachment
oDSI.ParseAttachment hInstance, dsiINPUTQUEUE

' get the attachment records
oDSI.GetAttachRecSet hInstance, dsiINPUTQUEUE, sRecID, sRecSet

' show results
For i = 0 To UBound(sRecSet, 1)
    MsgBox (sRecSet(i, 0) + " "+ sRecSet(i, 1))
Next i
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LastRec</td>
<td>(Optional) The last record to recover. The default is zero (0), which is translated to LONG_MAX.</td>
</tr>
</tbody>
</table>
GetAttachVarSet

Use this method to help locate a set of variables in an attachment. This method lets you pass in an array of the names you are looking for and get back the values associated with those names.

**Syntax**

```vbnet
GetAttachVarSet(hInstance as Long, QueueID as DSIQUEUE, Names() as String, Values() as String) as Long
```

You will get back a dsiERR_NOTFOUND if and only if none of the names are found.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
<tr>
<td>Names</td>
<td>The vector that contains the names you want to look for.</td>
</tr>
<tr>
<td>Values</td>
<td>The array that contains the matching values, if any. The value can be a pointer to an empty array, in which case the system dimensions it as a vector with the same length as the name array. If the array is defined before you call this method, it must be a two-dimensional array and the method will append a column to it.</td>
</tr>
</tbody>
</table>

**Returns**

- DSIERR_SUCCESS
- DSIERR_NOTFOUND

**See also**

- AddToAttachRec on page 254
- AttachCursorFirst on page 257
- AttachCursorLast on page 258
- AttachCursorNext on page 261
- AttachCursorPrev on page 262
- GetAttachmentAll on page 284
- LocateAttachVar on page 306
- ParseAttachment on page 312

**Example**

From the CSamAPI.cls file in the DSICoSAM example:

```vbnet
Dim sDummy1() as String
Dim sDummy2() as String
Dim sUnique as String
'
' there is no attachment for SSS, so we use empty arrays.
' sUnique is empty so we will get back the unique ID we can use to recover the server response
oDSI.Submit hInstance, "SSS", sUnique, sDummy1(), sDummy2()
'
' get the server status record
oDSI.GetQueueRec hInstance, dsiINPUTQUEUE, sUnique
```
DoEvents

' parse the attachment
oDSI.ParseAttachment hInstance, dsiINPUTQUEUE

sNames(0) = "UPTIME"
sNames(1) = "LASTRESTART"
sNames(2) = "RESTARTCOUNT"
sNames(3) = "SUCCESSCOUNT"
sNames(4) = "ERRORCOUNT"
sNames(5) = "ALLOCACCOUNT"
sNames(6) = "FREECOUNT"

' Get the current statistics from IDS
lRet = oDSI.GetAttachVarSet(hInstance, dsiINPUTQUEUE, sNames, asStats)
If lRet = dsiERR_EOF Then
    MsgBox ("FAILED. Code = ", Val(lRet))
Else
    Dim i
    Dim L, U
    L = LBound(sNames)
    U = UBound(sNames)
    For i = L To U
        MsgBox (sNames(i) + " : " + asStats(i))
    Next i
End If
GetPriority

Use this method to get the priority of the current queue record.

Syntax

GetPriority(hInstance as Long, QueueID as DSIQUEUE) as String

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
<tr>
<td>Priority</td>
<td>The priority as a string.</td>
</tr>
</tbody>
</table>

See also

GetQFieldLength on page 292
GetUniqueID on page 297
GetReqType on page 295
GetStatus on page 296
SetPriority on page 316
SetQField on page 317

Example

```vbnet
Dim sPri as String
sPri = oDSI.GetPriority(hInstance, dsiINPUTQUEUE)
```
GetQField

Use this method to retrieve the value of a queue field.

Syntax

GetQField(hInstance as Long, QueueID as DSIQUEUE, FieldID as long, Field as String)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
<tr>
<td>FieldID</td>
<td>A field identifier, such as: dsiQSET_REQTYPE, dsiQSET_STATUS, dsiQSET_INTIME, dsiQSET_OUTTIME, dsiQSET_USERID, dsiQSET_PRIORITY, dsiQSET_UNIQUE_ID, or dsiQSET_ATTACHMENT.</td>
</tr>
<tr>
<td>Field</td>
<td>The returned field value as a string.</td>
</tr>
</tbody>
</table>

See also

GetQFieldLength on page 292
GetPriority on page 290
GetUniqueID on page 297
GetReqType on page 295
GetStatus on page 296
SetPriority on page 316
SetQField on page 317
SetUserID on page 321
SetReqType on page 318
SetStatus on page 319
SetUniqueID on page 320

Example

oDSI.GetQField (hInstance, dsiINPUTQUEUE, dsiQSET_REQTYPE, sReq)
MsgBox ("The request was " + sReq)
GetQFieldLength

Use this method to get the field length of a field in a queue.

Syntax

GetQFieldLength(hInstance as Long, QueueID as DSIQUEUE, FieldID as Long) as Long

**NOTE:** This length can change from one release to the next so it is a good practice to interrogate the length at least once at run time rather than rely on hard-coded values.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
<tr>
<td>FieldID</td>
<td>A field identifier, such as: dsiQSET_REQTYPE, dsiQSET_STATUS, dsiQSET_INTIME, dsiQSET_OUTTIME, dsiQSET_USERID, dsiQSET_PRIORITY, dsiQSET_UNIQUE_ID, or dsiQSET_ATTACHMENT.</td>
</tr>
</tbody>
</table>

Returns

FieldLen, which provides the length of the requested queue field.

See also

GetQField on page 291

Example

dim cbUniqueID
    cbUniqueID = GetQFieldLength
    (hInstance, dsiINPUTQUEUE, dsiQSET_UNIQUE_ID)
GetQueueRec

Use this method to look for a specific record in the queue.

Syntax

GetQueueRec(hInstance as Long, QueueID as DSIQUEUE, UniqueID as String, _ Optional Wait as Long, Optional TimeOut as Long)

Please note:

- Oracle Insurance supplies timing defaults of 1000 and 15000 in one millisecond ticks
- If the queue record fails to appear in the specified time, dsiERR_EOF is returned
- A time-out usually indicates the server is down or unreachable

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
<tr>
<td>UniqueID</td>
<td>The record name.</td>
</tr>
<tr>
<td>Wait</td>
<td>The retry wait period in milliseconds.</td>
</tr>
<tr>
<td>TimeOut</td>
<td>The timeout in milliseconds.</td>
</tr>
</tbody>
</table>

See also

FindInQueue on page 281

Example

From the CSamAPI.cls file in the DSICoSAM example:

```vba
    sAttach(0, 0) = "Name0"
sAttach(0, 1) = "Value0"
sAttach(1, 0) = "Name1"
sAttach(1, 1) = "Value1"
sAttach(2, 0) = "Name2"
sAttach(2, 1) = "Value2"
sAttach(3, 0) = "Name3"
sAttach(3, 1) = "Value3"
sAttach(4, 0) = "Name4"
sAttach(4, 1) = "Value4"

dim sDummy() as String
    ' send the attachment to the server with the request it be echoed back
    sUnique = ***    ' to get us a new UniqueID
    oDSI.Submit hInstance, "ECH", sUnique, sAttach1, sDummy
    ' Look for the result.
    ' The DSI Document server will process our request and put the
    ' result in our result queue. We look for it in our result queue
    ' providing wait and lock timeout.
    ' If OnError gets invoked here, one of the error returns could
    ' be time out.
    oDSI.GetQueueRec hInstance, dsiINPUTQUEUE, sUnique
```

293
' Parse and present our results.
odSI.ParseAttachment hInstance, dsiINPUTQUEUE

' Open a cursor for the attachment
hCursor = oDSI.OpenAttachCursor(hInstance, dsiINPUTQUEUE)

' Position to the first element of the attachment'
lRet = oDSI.AttachCursorFirst(hCursor, sName, sValue)

' Loop through all elements of the parsed attachment printing
' the name and value pairs and put them in the right hand list box
While lRet = dsiERR_SUCCESS
    MsgBox (sName + " : " + sValue)
    lRet = oDSI.AttachCursorNext(hCursor, sName, sValue)
Wend

' Close the attachment cursor'
odSI.CloseAttachCursor hCursor
GetReqType

Use this method to get the DSI request type, such as SSS or IMP, from the current queue record.

Syntax

GetReqType(hInstance as Long, QueueID as DSIQUEUE) as String

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
</tbody>
</table>

Returns

ReqType, which provides the request type.

See also

- GetQFieldLength on page 292
- GetPriority on page 290
- GetUniqueID on page 297
- GetStatus on page 296

Example

MsgBox ("Request type was " + oDSI.GetReqType(hInstance, dsiINPUTQUEUE))
GetStatus

Use this method to get the status byte from the current queue record.

Syntax

GetStatus(hInstance as Long, QueueID as DSIQUEUE) as String

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
</tbody>
</table>

Returns
Status, which provides the status byte from the queue record.

See also
GetQFieldLength on page 292
GetPriority on page 290
GetUniqueID on page 297
GetReqType on page 295

Example

Dim sStatus as String
sStatus = oDSI.GetStatus (hInstance, dsiINPUTQUEUE)
GetUniqueID

Use this method to get the unique ID from a queue record.

Syntax

GetUniqueID(hInstance as Long, QueueID as DSIQUEUE) as String

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
</tbody>
</table>

Returns

UniqueId, which provides the unique ID for this record.

See also

GetQFieldLength on page 292
GetPriority on page 290
GetReqType on page 295

Example

From the CSamAPI.cls file in the DSICoSAM example:

MsgBox ("UniqueId is " + oDSI.GetUniqueID(hInstance, dsiINPUTQUEUE))
GetUniqueIDLength

Use this method to get the length of the unique ID field the queue is expecting.

Syntax

\[
\text{GetUniqueIDLength}(\text{hInstance as Long, QueueID as DSIQUEUE}) \rightarrow \text{Long}
\]

NOTE: This length can change from release to release.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
</tbody>
</table>

Returns

UniqueLen, which provides the returned length of the dsiQSET_UNIQUE_ID field.

See also

GetQField on page 291
GetQFieldLength on page 292
GetPriority on page 290
GetReqType on page 295
GetUniqueID on page 297

Example

From the CSamAPI.cls file in the DSICoSAM example:

```vbnet
Dim cbField as Long
cbField = oDSI.GetUniqueIDLength(hInstance, dsiINPUTQUEUE)
MsgBox("Unique ID field length is "+cbField)
```
GetUniqueString

Use this method to fill Unique with a unique string. You can, for instance, use this method to generate unique file names.

Syntax

GetUniqueString(hInstance as Long, Unique as String, Optional Long LengthRequested)

If LengthRequested is zero, the length of the UniqueID field in the queue record will be used. The GetUniqueID method is better suited for this purpose.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>Unique</td>
<td>returned unique ID. Unique will be space filled beyond 32 bytes.</td>
</tr>
<tr>
<td>LengthRequested</td>
<td>length of string requested. If the result is zero, the default, then the dsiQSET_UNIQUE_ID length is used.</td>
</tr>
</tbody>
</table>

See also

GetPriority on page 290
GetQField on page 291
GetQFieldLength on page 292
GetReqType on page 295
GetUniqueID on page 297

Example

Dim sUnique as String
GetUniqueString hInstance, sUnique, 8
MsgBox ("Here's your unique filename: " + sUnique + ".dat")
**GetUserID**

Use this method to get the user ID from the current queue record.

**Syntax**

```vbnet
getUserID(hInstance as Long, QueueID as DSIQUEUE) as String
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
</tbody>
</table>

**Returns**

UserID, which provides the user ID returned as a string.

**See also**

- GetPriority on page 290
- GetQField on page 291
- GetQFieldLength on page 292
- GetReqType on page 295
- GetUniqueID on page 297

**Example**

From the CSamAPI.cls file in the DSICoSAM example:

```vbnet
dim sField as String
sField = oDSI.GetUserID(hInstance, dsiINPUTQUEUE)
```
Init

Use this method to make an API call to initialize an IDS session. It is also called by InitSession, which is the preferred way to link up with IDS. Unless you want to administer the session directly, there is no need to call this method.

Syntax

Init() as Long

**NOTE:** This method should be called only once per process—without an intervening call to the Term method. You cannot use this method in a rule.

Arguments

None

Returns

phApp, which provides the DSI session handle (not instance).

See also

Term on page 325
InitSession on page 304
TermSession on page 328

Example

From the CSamAPI.cls file in the DSICoSAM example:

```vba
hApp = oDSI.Init()

hInstance = oDSI.InitInstance(hApp)

' init the queues but use DSI.INI by passing in "" as the path
oDSI.InitQueue hInstance, dsiINPUTQUEUE, ""
oDSI.InitQueue hInstance, dsiOUTPUTQUEUE, ""

' do something useful

' shut down
oDSI.TermQueue hInstance, dsiINPUTQUEUE
oDSI.TermQueue hInstance, dsiOUTPUTQUEUE
oDSI.TermInstance hInstance
oDSI.Term
```
InitInstance

Use this method to make an API call to initialize a thread instance. This method is also called by InitSession, which is the preferred way to link to IDS. Unless you want to administer the session directly, there is no need to call this routine.

Syntax

InitInstance(LONG hApp) as Long

**NOTE:** You cannot use this method in a rule.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hApp</td>
<td>IDS Server session</td>
</tr>
</tbody>
</table>

Returns

Instance, which provides the instance handle.

See also

TermInstance on page 326
InitSession on page 304

Example

From the CSamAPI.cls file in the DSICoSAM example:

```vb
' initialize DSI for this process
hApp = oDSI.Init()

' initialize DSI for this thread
hInstance = oDSI.InitInstance(hApp)

' init the queues but use DSI.INI by passing in "" as the path
oDSI.InitQueue hInstance, dsiINPUTQUEUE, ""
oDSI.InitQueue hInstance, dsiOUTPUTQUEUE, ""

' do something useful

' shut down
oDSI.TermQueue hInstance, dsiINPUTQUEUE
oDSI.TermQueue hInstance, dsiOUTPUTQUEUE
oDSI.TermInstance hInstance
oDSI.Term
```
InitQueue

Use this method to initialize a DSI Queue for this instance. This method is also called by InitSession, which is the preferred way to link to IDS and the queues.

NOTE: You cannot use this method in a rule.

Syntax

InitQueue(hInstance as Long, QueueID as DSIQUEUE, FileName as String)

If the file name is empty, DSI will look for the DSI.INI file in either the current working directory or the directory which contains the DSIW32.DLL file. For greater flexibility in your applications, do not specify the file name.

NOTE: Unless you want to administer the queues directly for a special purpose, this method should not be used. InitSession will make the necessary calls.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>thread instance handle</td>
</tr>
<tr>
<td>QueueID</td>
<td>queue index</td>
</tr>
<tr>
<td>FileName</td>
<td>queue path. Most applications will set this to &quot;&quot;.</td>
</tr>
</tbody>
</table>

See also

InitSession on page 304
TermQueue on page 327
TermSession on page 328

Example

From the CSamAPI.cls file in the DSICoSAM example:

```
' initialize DSI for this process
hApp = oDSI.Init()

' initialize DSI for this thread
hInstance = oDSI.InitInstance(hApp)

' init the queues but use DSI.INI by passing in "" as the path
oDSI.InitQueue hInstance, dsiINPUTQUEUE, ""
oDSI.InitQueue hInstance, dsiOUTPUTQUEUE, ""

' do something useful

' shut down
oDSI.TermQueue hInstance, dsiINPUTQUEUE
oDSI.TermQueue hInstance, dsiOUTPUTQUEUE
oDSI.TermInstance hInstance
oDSI.Term
```
**InitSession**

Use this method to initialize your IDS session through the Visual Basic API for the current thread. Most applications begin their processing with a call to InitSession.

**Syntax**

```vbnet
InitSession(long hApp) as Long
```

**NOTE:** You cannot use this method in a rule.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hApp</td>
<td>The app handle returned by the Init method. This is available for diagnostic purposes only.</td>
</tr>
</tbody>
</table>

**Returns**

The thread instance handle.

**See also**

- TermSession on page 328
- Init on page 301
- InitInstance on page 302
- InitQueue on page 303

**Example**

```vbnet
Dim sUnique as String
Dim sDummy() as String
Dim sReturn() as String
Dim sAttach(0 To 4, 0 To 1) As String
sAttach(0, 0) = "Name0"
sAttach(0, 1) = "Value0"
sAttach(1, 0) = "Name1"
sAttach(1, 1) = "Value1"
sAttach(2, 0) = "Name2"
sAttach(2, 1) = "Value2"
sAttach(3, 0) = "Name3"
sAttach(3, 1) = "Value3"
sAttach(4, 0) = "Name4"
sAttach(4, 1) = "Value4"

' set up our server session
hInstance = oDSI.InitSession()

' send the attachment to the server with the request t be echoed back
sUnique = ""' to get us a new UniqueID
oDSI.Submit hInstance, "ECH", sUnique, sAttach1, sDummy

' Look for the result.
oDSI.GetQueueRec hInstance, dsiINPUTQUEUE, sUnique

' get the attachment into an array
oDSI.GetAttachmentAll hInstance, dsiINPUTQUEUE, sAttachIn
```
' shut down
  oDSI.TermSession hInstance
LocateAttachVar

Use this method to locate an attachment variable in the current queue record.

Syntax

LocateAttachVar(hInstance as Long, QueueID as DSIQUEUE, Name as String, Value as String) as Long

You must call the ParseAttachment method before you use this method.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
<tr>
<td>Name</td>
<td>The search target.</td>
</tr>
<tr>
<td>Value</td>
<td>The value found associated with that name returned as a string.</td>
</tr>
</tbody>
</table>

Returns
dsiERR_SUCCESS

dsiERR_NOTFOUND

See also
AddAttachVar on page 253
DeleteAttachVar on page 274
ParseAttachment on page 312
GetAttachVarSet on page 288
GetAttachRecSet on page 286

Example

From the CSamAPI.cls file in the DSICoSAM example:

```vbnet
oDSI.ParseAttachment hInstance, dsiINPUTQUEUE

lRet = oDSI.LocateAttachVar(hInstance, dsiINPUTQUEUE, "RESULTS", sValue)

If lRet = dsiERR_SUCCESS Then
    MsgBox ("Success: found RESULTS = " + sValue)
Else
    MsgBox ("Failure: " + Hex(lRet) +" No data found: ")
End If
```
LocateValue

Use this method to locate a persistent value by name. These variables are persistent and must be destroyed by a call to DestroyValue method. They are not associated with the queues or attachments and exist to aid communication or provide state information between rules and calls to rules.

Syntax

LocateValue(hInstance as Long, Name as String, Value as VARIANT) as Long

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>Name</td>
<td>name of the persistent value</td>
</tr>
<tr>
<td>Value</td>
<td>the value that is found</td>
</tr>
</tbody>
</table>

Returns
dsiERR_SUCCESS
dsiERR_NOTFOUND

See also
CreateValue on page 270
DestroyValue on page 275
LocateValueObj on page 309
QueryValueSize on page 313

Example

From the CSamAPI.cls file in the DSICoSAM example:

```vbscript
' save our string
sTestValue = "Hello World"
oDSI.CreateValue hInstance, sSTRING_TAG, sTestValue

' now get it back
Dim lRet
lRet = oDSI.LocateValue(hInstance, sSTRING_TAG, sReturnedValue)
If lRet <> dsiERR_SUCCESS Then
    MsgBox ("Failed")
Else
    MsgBox ("Success")
End If

' we’re through with it so we destroy it
oDSI.DestroyValue hInstance, sSTRING_TAG

' now lets see how integers fare
iTestValue = 234
oDSI.CreateValue hInstance, sINT_TAG, iTestValue

lRet = oDSI.LocateValue(hInstance, sINT_TAG, iReturnedValue)
If lRet <> dsiERR_SUCCESS Then
    MsgBox ("Failed")
Else
    MsgBox ("Success")
End If
```
Or iTestValue <> iReturnValue Then
    MsgBox ("Failed")
Else
    MsgBox ("Success")
End If

' we're through with it so we destroy it
oDSI.DestroyValue hInstance, sINT_TAG
**LocateValueObj**

Use this method to locate a persistent value containing the name of an object. These variables are persistent and must be destroyed by a call to DestroyValueObj. These variables are not associated with the queues or attachments and exist to aid communication or provide state information between rules and calls to rules.

**Syntax**

```
LocateValueObj(hInstance as Long, Name as String, oRef as Object) as Long
```

**NOTE:** ActiveX components are referenced counted and VB is very good about its record keeping so few are even aware that it is going on. If you use this method to save a reference to an object VB will take over that responsibility as much as it can. If, however, you fail to call DestroyValueObj, including in On Error handlers, you can leave a dangling reference which can tie up resources unnecessarily, perhaps even crash the server or your application.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>Name</td>
<td>name of the persistent value</td>
</tr>
<tr>
<td>oRef</td>
<td>a reference to an object</td>
</tr>
</tbody>
</table>

**Returns**

- dsiERR_SUCCESS
- dsiERR_NOTFOUND

**See also**

- CreateValueObj on page 272
- DestroyValueObj on page 277
- LocateValue on page 307

**Example**

From the CSamAPI.cls file in the DSICoSAM example:

```vbnet
Dim oTestValue As New CSamTObj
Dim oOtherTestValue As CSamTObj

' Test with early bound object
oDSI.CreateValueObj hInstance, "MY_OBJECT", oTestValue

' clear our reference
Set oTestValue = Nothing

' get it back
lRet = oDSI.LocateValueObj(hInstance, "MY_OBJECT", oOtherTestValue)

' use the object to make sure we got back what we sent out
MsgBox (oOtherTestValue.TestReturn("Hello World"))
```
' clear our reference
Set oOtherTestValue = Nothing

' we don’t want a dangling reference
oDSI.DestroyValueObj hInstance, "MY_OBJECT"

' -----------------------------------------------------------
'  Test with late bound object

Dim oObject As Object
Dim oOtherObject As Object

Set oObject = CreateObject("Docucorp_IDS_SamTObj.CSamTObj")
oDSI.CreateValueObj hInstance, "MY_OBJECT", oObject

' clear our reference
Set oObject = Nothing

' get it back
lRet = oDSI.LocateValueObj(hInstance, "MY_OBJECT", oOtherObject)

' use the object to make sure we got back what we sent out
MsgBox ("Object #2 Recovered: " + oOtherObject.TestReturn ("Hello New World"))

' clear our reference
Set oOtherObject = Nothing

' we don’t want a dangling reference
oDSI.DestroyValueObj hInstance, "MY_OBJECT"
OpenAttachCursor

Use this method to open a cursor into the attachment list for the specified queue. Be sure to call the CloseAttachCursor method when you are through to free resources.

Syntax

```
OpenAttachCursor(hInstance as Long, QueueID as DSIQUEUE) as Long
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
</tbody>
</table>

Returns

Cursor, which provides the newly-created cursor.

See also

- AttachCursorLast on page 258
- AttachCursorNext on page 261
- AttachCursorPrev on page 262
- CloseAttachCursor on page 267
- ParseAttachment on page 312

Example

From the CSamAPI.cls file in the DSISAM example:

```vbs
oDSI.ParseAttachment hInstance, dsiINPUTQUEUE

' Open a cursor for the attachment
' This cursor will allow us to walk through the attachment serially
hCursor = oDSI.OpenAttachCursor(hInstance, dsiINPUTQUEUE)

' Position to the first element of the attachment'
lRet = oDSI.AttachCursorFirst(hCursor, sName, sValue)

' Loop through all elements of the parsed attachment printing
' the name and value pairs and put them in the right hand list box
While lRet = dsiERR_SUCCESS
    ... do something useful
    lRet = oDSI.AttachCursorNext(hCursor, sName, sValue)
Wend

' close out the cursor to free the resources
oDSI.CloseAttachCursor hCursor
```
ParseAttachment

Use this method to parse the attachment field in the queue record into an internal list of name/value pairs which can be accessed by other methods.

Syntax

```
ParseAttachment(hInstance as Long,QueueID as DSIQUEUE)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
</tbody>
</table>

See also

- GetAttachment on page 282
- LocateAttachVar on page 306
- DeleteAttachVar on page 274
- GetAttachmentAll on page 284
- GetAttachVarSet on page 288

Example

From the CSamAPI.cls file in the DSICoSAM example:

```
oDSI.ParseAttachment hInstance, dsiINPUTQUEUE

' Open a cursor for the attachment
' This cursor will allow us to walk through the attachment serially
hCursor = oDSI.OpenAttachCursor(hInstance, dsiINPUTQUEUE)

' Position to the first element of the attachment'
lRet = oDSI.AttachCursorFirst(hCursor, sName, sValue)

' Loop through all elements of the parsed attachment printing
' the name and value pairs and put them in the right hand list box
While lRet = dsiERR_SUCCESS
    ... do something useful
    lRet = oDSI.AttachCursorNext(hCursor, sName, sValue)
Wend

' close out the cursor to free the resources
oDSI.CloseAttachCursor hCursor
```
QueryValueSize

Use this method to get the length of a DSI persistent variable. These variables are persistent and must be destroyed by a call to DestroyValue method. They are not associated with the queues or attachments and exist to aid communication or provide state information between rules and calls to rules.

Syntax

```
QueryValueSize(hInstance as Long, sName as String) as Long
```

**NOTE:** Use of this method with a DSI persistent variable that is an object will return a value that is unreliable.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>Name</td>
<td>the name of the persistent variable</td>
</tr>
</tbody>
</table>

Returns

ValueLength, which provides the length in bytes.

See also

CreateValue on page 270
DestroyValue on page 275
LocateValue on page 307
CreateValueObj on page 272
LocateValueObj on page 309
DestroyValueObj on page 277

Example

From the CSamAPI.cls file in the DSICoSAM example:

```vbnet
sTestValue = "Hello World"
oDSI.CreateValue hInstance, "START_STMT", sTestValue

Dim cbValue
cbValue = oDSI.QueryValueSize(hInstance, "START_STMT")
MsgBox ("returned size=", Str(cbValue))
```
SetAttachment

Use this method to insert an attachment as a single, continuous string (almost a BLOB) into the queue record. Use for situations in which the name/value pair paradigm does not support the needs of the application.

Syntax

```
SetAttachment(hInstance as Long, QueueID as DSIQUEUE, Attachment as String)
```

Most applications which interact with IDS will not need to use this method.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
<tr>
<td>Attachment</td>
<td>The attachment as a string.</td>
</tr>
</tbody>
</table>

See also

GetAttachment on page 282

Example

Here is an excerpt from the CSamAPI.cls file in the DSICoSAM example:

```
Dim sBLOB As String
sBLOB = "Of all the dispositions and habits, which lead to political prosperity," + _
"Religion and Morality are indispensable supports. In vain would that man " + _
"claim the tribute of Patriotism, who should labor to subvert these great " + _
"pillars of human happiness, these firmest props of the duties of Men and " + _
"Citizens. The mere Politician, equally with the pious man, ought to respect " + _
"and to cherish them. A volume could not trace all their connexions with " + _
"private and public felicity. Let it simply be asked, Where is the security " + _
"for property, for reputation, for life, if the sense of religious obligation " + _
"desert the oaths, which are the instruments of investigation in Courts " + _
"of Justice? And let us with caution indulge the supposition, that morality " + _
"can be maintained without religion. Whatever may be conceded to the influence " + _
"of refined education on minds of peculiar structure, reason and experience " + _
"both forbid us to expect, that national morality can prevail in exclusion " + _
"of religious principle. -- George Washington"

oDSI.SetAttachment hInstance, dsiOUTPUTQUEUE, sBLOB

'set the Echo request type
oDSI.SetReqType hInstance, dsiOUTPUTQUEUE, "ECH"
```
' set up a unique id for our record
sUnique = ""      ' make sure we get a new one this time
oDSI.SetUniqueID hInstance, dsiOUTPUTQUEUE, sUnique

' insert record into queue for processing by the server
goDSI.AddToQueue hInstance, dsiOUTPUTQUEUE

' get our record back after processing by the server
oDSI.GetQueueRec hInstance, dsiINPUTQUEUE, sUnique

Dim sBLOBOut

oDSI.GetAttachment hInstance, dsiINPUTQUEUE, sBLOBOut

MsgBox (sBLOBOut)
SetPriority

Use this method to set the priority of the current queue record.

Syntax

```
SetPriority(hInstance as Long, QueueID as DSIQUEUE, Priority as String)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
<tr>
<td>Priority</td>
<td>The priority as a string.</td>
</tr>
</tbody>
</table>

See also

SetQField on page 317

SetUserID on page 321

SetReqType on page 318

SetStatus on page 319

SetUniqueID on page 320

GetPriority on page 290

Example

```
oDSI.SetPriority hInstance, dsiOUTPUTQUEUE, "1"
oDSI.AddToQueue hInstance, dsiOUTPUTQUEUE

oDSI.SetPriority hInstance, dsiOUTPUTQUEUE, "0"
oDSI.AddToQueue hInstance, dsiOUTPUTQUEUE
```
SetQField

Use this method to set a specific queue field in the current queue record.

Syntax

SetQField(hInstance as Long, QueueID as DSIQUEUE, FieldID as Long, Value as String)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
<tr>
<td>FieldID</td>
<td>A field identifier, such as: dsiQSET_REQTYPE, dsiQSET_STATUS, dsiQSET_USERID, dsiQSET_PRIORITY, dsiQSET_UNIQUE_ID, or dsiQSET_ATTACHMENT.</td>
</tr>
<tr>
<td>Value</td>
<td>The value to be updated in current queue record.</td>
</tr>
</tbody>
</table>

See also

GetQField on page 291
SetPriority on page 316
SetUserID on page 321
SetReqType on page 318
SetStatus on page 319
SetUniqueID on page 320

Example

From the CSamAPI.cls file in the DSICoSAM example:

' put our message in the attachment
doSI.AddAttachVar hInstance, dsiOUTPUTQUEUE, "What", "Me Worry?"

' put the attachment into the queue record
doSI.StoreAttachment hInstance, dsiOUTPUTQUEUE

' set up the request type (all queue records must have a request type)
doSI.SetQField hInstance, dsiOUTPUTQUEUE, dsiQSET_REQTYPE, "ECH"

' put a unique id in the queue record so we can get it from the server
sUnique = "" ' make sure we get a new one this time
doSI.SetUniqueID hInstance, dsiOUTPUTQUEUE, sUnique

' submit the queue record to the queue for processing by the server
doSI.AddToQueue hInstance, dsiOUTPUTQUEUE
SetReqType

Use this method to set the DSI request type in the current queue record.

Syntax

SetReqType(hInstance as Long, QueueID as DSIQUEUE, Type as String)

Every queue record submitted to the server must have a request type. This request type should also be found in the DOCSERV configuration file. For instance, the ECH request has the following entry in the DOCSERV configuration file:

```
< ReqType:ECH >
  function = atcw32->ATCLoadAttachment
  function = DSICOru1->Invoke, DocuCorp_IDS_SAMSupp.CSAMSupp->Echo
  function = atcw32->ATCUnloadAttachment
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
<tr>
<td>Type</td>
<td>The request type as a string.</td>
</tr>
</tbody>
</table>

See also

GetReqType on page 295
SetQField on page 317
SetPriority on page 316
SetUserID on page 321
SetStatus on page 319
SetUniqueID on page 320

Example

From the CSamAPI.cls file in the DSICoSAM example:

```
' put our message in the attachment
  oDSI.AddAttachVar hInstance, dsiOUTPUTQUEUE, "What", "Me Worry?"

' put the attachment into the queue record
  oDSI.StoreAttachment hInstance, dsiOUTPUTQUEUE

' set up the request type (all queue records must have a request type)
  oDSI.SetReqType hInstance, dsiOUTPUTQUEUE, "ECH"

' put a unique ID in the queue record
  sUnique = ""  ' make sure we get a new one this time
  oDSI.SetUniqueID hInstance, dsiOUTPUTQUEUE, sUnique

' submit the queue record to the queue for processing by the server
  oDSI.AddToQueue hInstance, dsiOUTPUTQUEUE
```
SetStatus

Use this method to set the status flag by OR’ing the bits, which will prevent the ERROR bit from being reset. This field has a length of one byte.

Syntax

SetStatus(hInstance as Long, QueueID as DSIQUEUE, Status as String)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
<tr>
<td>Status</td>
<td>The status as a string.</td>
</tr>
</tbody>
</table>

See also

- GetStatus on page 296
- SetQField on page 317
- SetPriority on page 316
- SetUserID on page 321
- SetReqType on page 318
- SetUniqueID on page 320

Example

ODSI.SetStatus hInstance, dsiINPUTQUEUE, "E"
SetUniqueID

Use this method to set the UniqueID for a queue record. In a multi-user environment, this is the way to keep your stuff separated from that of the other users. This value is supplied to the GetQueueRec method to recover your queue record after it’s processed by the server.

Syntax

```
SetUniqueID(hInstance as Long, QueueID as DSIQUEUE, UniqueID as String)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
<tr>
<td>UniqueID</td>
<td>UniqueID as a string. If UniqueID is empty or &quot;&quot;, a new unique ID is returned.</td>
</tr>
</tbody>
</table>

See also

- GetUniqueID on page 297
- SetQField on page 317
- SetPriority on page 316
- SetUserID on page 321
- SetReqType on page 318
- SetStatus on page 319

Example

From the CSamAPI.cls file in the DSICoSAM example:

```vbscript
sUnique = ""  ' make sure we get a new one this time
oDSI.SetUniqueID hInstance, dsiOUTPUTQUEUE, sUnique

' insert our record into the queue for processing by the server
oDSI.AddToQueue hInstance, dsiOUTPUTQUEUE

' recover our record from the server after processing
oDSI.GetQueueRec hInstance, dsiINPUTQUEUE, sUnique
```
SetUserID

Use this method to set up a user ID for the current queue record. The server does not use this, but a client can use it to keep separate various requests.

**Syntax**

```plaintext
SetUserID(hInstance as Long, QueueID as DSIQUEUE, UserID as String)
```

If the user ID is not going to change, you only need to make this call once. You can also use the UserID property to set this field.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
<tr>
<td>UserID</td>
<td>Any string.</td>
</tr>
</tbody>
</table>

**See also**

- [GetUserID on page 300](#)
- [SetQField on page 317](#)
- [SetPriority on page 316](#)
- [SetReqType on page 318](#)
- [SetStatus on page 319](#)
- [SetUniqueID on page 320](#)

**Example**

From the CSamAPI.cls file in the DSICoSAM example:

```plaintext
oDSI.SetUserID hInstance, dsiOUTPUTQUEUE, "Walleye"
```
StoreAttachment

Use this method to update the attachment field in the queue record from the internal attachment list buffer.

Syntax

```vbscript
StoreAttachment(hInstance as Long, DSIQUEUE QueueID)
```

If you call the AddAttachVar or AttachList methods, you must call this method afterwards. This method is not required after calls to the Submit method.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
</tbody>
</table>

See also

- AddAttachVar on page 253
- AttachList on page 265
- Submit on page 323

Example

From the CSamAPI.cls file in the DSICoSAM example:

```vbscript
ODSI.AddAttachRec hInstance, dsiOUTPUTQUEUE, "FISH", sBuf

' Next we want to supply the values. To do this we use the ' add to attach record functionality. We supply the buffer ' returned from or earlier add attach record call.

' Add name of my DLL
ODSI.AddToAttachRec hInstance, dsiOUTPUTQUEUE, sBuf, "TYPE", "BASS"

' Add date DLL was built
ODSI.AddToAttachRec hInstance, dsiOUTPUTQUEUE, sBuf, "SIZE", "LARGE"

' Add time DLL was built
ODSI.AddToAttachRec hInstance, dsiOUTPUTQUEUE, sBuf, "STATUS", "CAUGHT"

' Add my DLL version number
ODSI.AddToAttachRec hInstance, dsiOUTPUTQUEUE, sBuf, "LOCATION", "BOAT"

' Put the attachment into the queue record
ODSI.StoreAttachment hInstance, dsiOUTPUTQUEUE
```
Submit

Use this method for most client submissions to the server.

Syntax

```
Submit(hInstance as Long, Request as String, UniqueID as String, parms1() as String, parms2() as String)
```

The lists parms1() and parms2() can be empty.

**NOTE:** Each call to submit generates another OUTPUT queue record.

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>Request</td>
<td>A server request, such as SSS.</td>
</tr>
<tr>
<td>UniqueID</td>
<td>The unique ID for this submission. Any empty string will be returned with the unique ID assigned to this queue record.</td>
</tr>
<tr>
<td>Parms1()</td>
<td>A two-dimensional array with the parameter list to attach to the queue record.</td>
</tr>
<tr>
<td>Parms2()</td>
<td>A two-dimensional array with the second parameter list to be also attached to the queue record.</td>
</tr>
</tbody>
</table>

**See also**

AddAttachVar on page 253

AttachList on page 265

**Example 1** From the CSamAPI.cls file in the DSICoSAM example:

```
Dim sDummy1() as String
Dim sDummy2() as String
Dim sUnique as String
' there is no attachment for SSS, so we use empty arrays.
' sUnique is empty so we will get back the unique ID we can use to recover the server response
oDSI.Submit hInstance, "SSS", sUnique, sDummy1(), sDummy2()

' get the server status record
oDSI.GetQueueRec hInstance, dsiINPUTQUEUE, sUnique, DoEvents

' parse the attachment
oDSI.ParseAttachment hInstance, dsiINPUTQUEUE

sNames(0) = "UPTIME"
sNames(1) = "LASTRESTART"
sNames(2) = "RESTARTCOUNT"
sNames(3) = "SUCCESSCOUNT"
sNames(4) = "ERRORCOUNT"
sNames(5) = "ALLOCCOUNT"
```
sNames(6) = "FREECOUNT"

' Get the current statistics from IDS
lRet = oDSI.GetAttachVarSet(hInstance, dsiINPUTQUEUE, sNames, asStats)
If lRet = dsiERR_EOF Then
    MsgBox ("FAILED. Code = ", Val(lRet))
Else
    Dim i
    Dim L, U
    L = LBound(sNames)
    U = UBound(sNames)
    For i = L To U
        MsgBox (sNames(i) + ": " + asStats(i))
    Next i
End If

Example 2 From the CSamAPI.cls file in the DSICoSAM example:

sAttach1(0, 0) = "Name0"
sAttach1(0, 1) = "Value0"
sAttach1(1, 0) = "Name1"
sAttach1(1, 1) = "Value1"
sAttach1(2, 0) = "Name2"
sAttach1(2, 1) = "Value2"
sAttach1(3, 0) = "Name3"
sAttach1(3, 1) = "Value3"
sAttach1(4, 0) = "Name4"
sAttach1(4, 1) = "Value4"

sAttach2(0, 0) = "Name20"
sAttach2(0, 1) = "Value20"
sAttach2(1, 0) = "Name21"
sAttach2(1, 1) = "Value21"
sAttach2(2, 0) = "Name22"
sAttach2(2, 1) = "Value22"
sAttach2(3, 0) = "Name23"
sAttach2(3, 1) = "Value23"
sAttach2(4, 0) = "Name24"
sAttach2(4, 1) = "Value24"

' send the attachment to the server with the request it be echoed back
sUnique = "" ' to get us a new UniqueID
oDSI.Submit hInstance, "ECH", sUnique, sAttach1, sAttach2

' wait for the server to return the attachment
oDSI.GetQueueRec hInstance, dsiINPUTQUEUE, sUnique, 1000, nTIMEOUT

' get the attachment into an array
oDSI.GetAttachmentAll hInstance, dsiINPUTQUEUE, sAttachIn

For i = LBound(sAttachIn, 1) To UBound(sAttachIn, 1)
    MsgBox (sAttachIn(i, 0) +": " + sAttachIn(i, 1))
Next i
Use this method to terminate the server session.

**Syntax**

```
Term()
```

The `InitSession` and `TermSession` methods are the preferred means of managing your connection to IDS. Unless you want to manage the server session directly, you should not call this routine.

**NOTE:** This method will be automatically called when you exit. Most applications will not use it. This method cannot be called from a rule.

**Arguments**

None

**See also**

*Init on page 301*

*InitSession on page 304*

**Example**

From the `CSamAPI.cls` file in the DSICoSAM example:

```
' initialize DSI for this process
hApp = oDSI.Init()

' initialize DSI for this thread
hInstance = oDSI.InitInstance(hApp)

' init the queues but use DSI.INI by passing in "" as the path
oDSI.InitQueue hInstance, dsiINPUTQUEUE, ""
oDSI.InitQueue hInstance, dsiOUTPUTQUEUE, ""

' do something useful

' shut down
oDSI.TermQueue hInstance, dsiINPUTQUEUE
oDSI.TermQueue hInstance, dsiOUTPUTQUEUE
oDSI.TermInstance hInstance
oDSI.Term
```
TermInstance

Use this method to terminate the thread instance. It is also called by TermSession, which is the preferred way to unlink from IDS.

Syntax

```vbnet
TermInstance(hInstance as Long)
```

**NOTE:** This method cannot be called from rules.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
</tbody>
</table>

**See also**

- [InitInstance on page 302](#)
- [InitSession on page 304](#)
- [TermSession on page 328](#)

**Example**

From the CSamAPI.cls file in the DSICoSAM example:

```vbnet
' initialize DSI for this process
hApp = oDSI.Init()

' initialize DSI for this thread
hInstance = oDSI.InitInstance(hApp)

' init the queues but use DSI.INI by passing in "" as the path
oDSI.InitQueue hInstance, dsiINPUTQUEUE, ""
oDSI.InitQueue hInstance, dsiOUTPUTQUEUE, ""

' do something useful

' shut down
oDSI.TermQueue hInstance, dsiINPUTQUEUE
oDSI.TermQueue hInstance, dsiOUTPUTQUEUE
oDSI.TermInstance hInstance
oDSI.Term
```
TermQueue

Use this method to terminate the linkage to one of the two queues. Called by InitSession, which is the preferred way to link to IDS.

Syntax

TermQueue(hInstance as Long, QueueID as DSIQUEUE)

**NOTE:** This method cannot be called from rules.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
</tbody>
</table>

See also

- InitQueue on page 303
- InitSession on page 304
- TermSession on page 328

Example

From the CSamAPI.cls file in the DSICoSAM example:

```vbs
' initialize DSI for this process
hApp = oDSI.Init()

' initialize DSI for this thread
hInstance = oDSI.InitInstance(hApp)

' init the queues but use DSI.INI by passing in "" as the path
oDSI.InitQueue hInstance, dsiINPUTQUEUE, ""
oDSI.InitQueue hInstance, dsiOUTPUTQUEUE, ""

' do something useful

' shut down
oDSI.TermQueue hInstance, dsiINPUTQUEUE
oDSI.TermQueue hInstance, dsiOUTPUTQUEUE
oDSI.TermInstance hInstance
oDSI.Term
```
**TermSession**

Use this method to end the relationship with IDS. You must pair this method with the InitSession method.

**Syntax**

```
TermSession(hInstance as Long)
```

**NOTE:** This method cannot be called from rules.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle.</td>
</tr>
</tbody>
</table>

**See also**

- InitSession on page 304
- Init on page 301

**Example**

From the CSamAPI.cls file in the DSICoSAM example:

```vbnet
Dim sUnique as String
Dim sDummy() as String
Dim sReturn() as String
Dim sAttach(0 To 4, 0 To 1) As String
sAttach(0, 0) = "Name0"
sAttach(0, 1) = "Value0"
sAttach(1, 0) = "Name1"
sAttach(1, 1) = "Value1"
sAttach(2, 0) = "Name2"
sAttach(2, 1) = "Value2"
sAttach(3, 0) = "Name3"
sAttach(3, 1) = "Value3"
sAttach(4, 0) = "Name4"
sAttach(4, 1) = "Value4"

hInstance = oDSI.InitSession()

' send the attachment to the server with the request it be echoed back
sUnique = "" ' to get us a new UniqueID
oDSI.Submit hInstance, "ECH", sUnique, sAttach, sDummy

' Look for the result.
oDSI.GetQueueRec hInstance, dsiINPUTQUEUE, sUnique

' get the attachment into an array
oDSI.GetAttachmentAll hInstance, dsiINPUTQUEUE, sAttachIn

' shut down
oDSI.TermSession hInstance
```
Trace

Use this method to put a couple of strings in the VB trace file. If tracing is not enabled, no action is taken.

Syntax

```
Trace(hInstance as Long, Caller as String, Msg as String)
```

The trace file is named DSICO.TRC. This file is stored in the current working directory of the application, IDS Server, or IIS Server, unless you specify otherwise using the TracePath property.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle.</td>
</tr>
<tr>
<td>Caller</td>
<td>The routine making the call.</td>
</tr>
<tr>
<td>Msg</td>
<td>A message string.</td>
</tr>
</tbody>
</table>

See also

- TraceSnapshot on page 333
- TraceEnableRule on page 331
- Property TracePath on page 337
- TraceEnableRule on page 331

Example

```
oDSI.Trace hInstance, "Fish Rule::GoFish", "Bass bait ignored"
```
TraceAttach

Use this method to write the entire attachment to the trace file.

Syntax

```
TraceAttach(hInstance as Long, QueueID as DSIQUEUE)
```

The trace file is always named DSICO.TRC. It will go in the current working directory of the application, IDS Server, or IIS Server, unless you specify otherwise using the TracePath property.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>QueueID</td>
<td>Either dsiINPUTQUEUE or dsiOUTPUTQUEUE.</td>
</tr>
</tbody>
</table>

See also

- Trace on page 329
- TraceEnableRule on page 331
- TraceSnapshot on page 333
- TraceEnableRule on page 331
- TraceEnableRule on page 331
- Property TracePath on page 337

Example

From the CSamAPI.cls file in the DSICoSAM example:

```
oDSI.TraceAttach hInstance, dsiINPUTQUEUE
```
TraceEnableRule

Use this method to turn the tracing on and off in a rule. The TraceEnable property cannot be used in rules.

Syntax

```
TraceEnableRule(hInstance as Long, Enable as Boolean)
```

The trace file is always named DSICO.TRC. It will go in the current working directory of the application, IDS Server, or IIS Server, unless you specify otherwise using the TracePath property.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
<tr>
<td>bEnable</td>
<td>Enter True to enable tracing. Enter False to disable tracing.</td>
</tr>
</tbody>
</table>

See also

- Trace on page 329
- TraceAttach on page 330
- TraceEnableRule on page 331
- TraceSnapshot on page 333
- Property TracePath on page 337

Example

```
oDSI.TraceEnableRule hInstance, TRUE
```
TraceList

Use this method to trace an attachment list of name/value pairs.

Syntax

```vbnet
TraceList(ID as String, List() as String)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>A list identifier.</td>
</tr>
<tr>
<td>List ()</td>
<td>A two-dimensional array of name/value pairs.</td>
</tr>
</tbody>
</table>

See also

- Trace on page 329
- TraceAttach on page 330
- TraceEnableRule on page 331
- TraceEnableRule on page 331
- TraceSnapshot on page 333

Example

From the CSamAPI.cls file in the DSICoSAM example:

```vbnet
dim sAttach(0 to 4, 0 to 1) as string
sAttach(0, 0) = "Name0"
sAttach(0, 1) = "Value0"
sAttach(1, 0) = "Name1"
sAttach(1, 1) = "Value1"
sAttach(2, 0) = "Name2"
sAttach(2, 1) = "Value2"
sAttach(3, 0) = "Name3"
sAttach(3, 1) = "Value3"
sAttach(4, 0) = "Name4"
sAttach(4, 1) = "Value4"

oDSI.TraceList "Initial list state", sAttach
```
TraceSnapshot

Use this method to dump the current state of the queues, including attachments in the current queue record, to the trace file. This method then closes and reopens the trace file to flush the buffers.

Syntax

```
TraceSnapshot (hInstance as Long)
```

Arguments

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hInstance</td>
<td>The thread instance handle. This comes from the server if it was invoked by a rule.</td>
</tr>
</tbody>
</table>

See also

- Trace on page 329
- TraceAttach on page 330
- TraceEnableRule on page 331

Example

From the CSamAPI.cls file in the DSICoSAM example:

```
' recover the attachment echoed back to us
ODSI.GetQueueRec hInstance, dsiINPUTQUEUE, sUnique

ODSI.Trace "Fish::GoFish","where are the worms?"
ODSI.TraceSnapshot hInstance
```
Property Instance

Use this property to return the DSI instance handle.

Syntax

```
Property Instance as Long (read only)
```

This method is for diagnostic purposes only.

**NOTE:** In a multi-threaded context, such as an ASP Active X component running under Microsoft IIS, you cannot rely on this value.

See also

- InitInstance on page 302
- InitSession on page 304
- TermInstance on page 326
- TermSession on page 328

Example

```
MsgBox("Instance handle is " + Str(oDSI.Instance))
```
**Property Signature**

Use this property to return the DLL “signature” for diagnostic purposes.

**Syntax**

```vbscript
Property Signature as String
```

**NOTE:** This information is subject to change in content and format without notice.

**Returns**

A string with data identifying the VB ActiveX DLL.

**Example**

```vbscript
MsgBox("DSICoLib signature: " + Str(oDSI.Signature))
```
Property TraceEnable

Use this property to start and stop tracing.

**Syntax**

```
Property TraceEnable as BOOL (read only)
```

The trace file is always named DSICO.TRC. It will go in the current working directory of the application, IDS Server, or IIS Server, unless you specify otherwise using the TracePath property.

The trace file will be automatically closed when the application exits.

**See also**

- TraceSnapshot on page 333
- TraceAttach on page 330
- Trace on page 329
- TraceEnableRule on page 331
- Property TracePath on page 337

**Example**

From the CSamAPI.cls file in the DSICoSAM example:

```
oDSI.TraceEnable = true
oDSI.InitSession
```
Property TracePath

Use this property to get the path and file name of the trace file, if the trace file has been opened, the system will set the trace file name. This name will take effect only after the trace file is opened.

Syntax

Property TracePath as String

The trace file is always named DSICO.TRC. It will go in the current working directory of the application, IDS Server, or IIS Server, unless you specify otherwise using this property.

See also TraceEnableRule on page 331

Example

oDSI.TracePath = "D:\TEMP"
oDSI.TraceEnable = true
oDSI.InitSession
Index

A

accessing variables 15
Active Server Page
   ADMAsp sample 76
   and COM objects 76
   Debug.ASP 72
ActiveX
   registering DLLs 62
   writing Visual Basic rules 61
AddAllRequest
   JavaBeans 172
AddAttachRec
   in C 15, 83
   in Visual Basic 251
AddAttachVar
   in C 8, 15, 84, 85
   in Visual Basic 253
ADDINST program 62
AddRequest
   JavaBeans 172
AddToAttachRec
   in C 15, 86, 87
   in Visual Basic 254
AddToQueue
   in C 8, 89
   in Visual Basic 256
ADMAsp
   sample 76
ADMIN.ASP
   sample 76
Index

APIs
   C 79
   Java 171
   Visual Basic 245
ATCApend2Attachment 226
ATCLoadAttachment 13, 227
ATCLogTransaction 228
ATCReceiveFile 230
   referencing attachment variables 78
ATCSendFile 233
   referencing attachment variables 78
ATCSendMultipleFiles rule 235
ATCUnloadAttachment 13, 236
AttachCursorFirst
   in C 9, 15, 90, 92
   in Visual Basic 257
AttachCursorLast
   in C 9, 15, 94
   in Visual Basic 258
AttachCursorName
   in C 98
   in Visual Basic 260
AttachCursorNext
   in C 9, 15, 100, 102
   in Visual Basic 261
AttachCursorPrev
   in C 9, 15, 104, 106
   in Visual Basic 262
AttachCursorValue
   in C 108, 110
   in Visual Basic 264
AttachList
   in Visual Basic 265
Attachment control group 199
attachment variables
   referencing 78
AttachmentPath option 227, 236

attachments
   accessing using C 15
   defined 4
   DSIReceiveFile 147
   DSISendFile 155
   list 7
   using a proprietary format with C 8
   variables 9, 13
Attachments option 231, 233
autorun rules
   CacheFile 115
avoiding file naming conflicts 16

B

Bounds Checker 12
buffer 8

C

C
   APIs 79
   creating, accessing, and destroying variables 15
   DSI API 3, 7
   list of API functions 80
   sample program 10
   writing processing rules 12
cache
   purging 204
CacheFile
   in C 16, 115
   in Visual Basic 266
CacheGetFile option 198
CAD request type 198
case statements
   in VB 62
 CGI client
   DSI API 3
   writing processing rules 12
class modules 61
classes
  list of Java classes and methods 182
CLF request type 17
client rules
  list of 225
CloseAttachCursor
  in C 15, 116
  in Visual Basic 267
CoClasses folder 63
COM
  and Visual Basic rules 66
  GUIDS 66
  interfaces and VB 61
  objects under ASP 76
  ProgIDs 61, 63
  registering objects 63
CONFIG.INI file
  referencing attachment variables 78
CopyAttachVars
  in C 15, 117
  in Visual Basic 268
CopyQRecord
  in C 118
  in Visual Basic 269
CreateValue
  in C 15, 119
  in Visual Basic 66, 270
CreateValueObj
  in Visual Basic 272
creating
  rules for reserved request types 16
  rules in C 13
  variables in C 15
  Visual Basic rules 61
Debug control group 231, 233
debug log file 201
Debug option 198, 201
Debug.ASP 72
DEFAULT request type 17
  UnloadAttachment 236
DeleteAttachVar
  in C 15, 120
  in Visual Basic 274
destroying variables
  C 15
DestroyValue
  in C 15, 121, 122, 123
  in Visual Basic 275
DestroyValueObj
  in Visual Basic 277
diagnostics
  Signature property 335
DOCC.BMP sample file 76
DOCSERV.INI file
  referencing attachment variables 78
  specifying Visual Basic rules 63
Document Server Interface (DSI) 171
DSI
  C APIs 79
  COM objects (ASP) 76
  Java APIs 171
  processing rules 183
  Visual Basic API 61
DSI Java APIs 76
DSI Visual Basic APIs 245
DSI_MSGINIT message 12
DSI_MSGRUNF message 12
DSI_MSGRUNR message 12, 13
DSI_MSGTERM message 12
DSIAddAttachVar
  in C 8
DSIAddAttachVarEx 85
DSICO.TRC 336
DSICO.ADM
  sample 76
DSICoAPI 65
daemon 17
DAP.INI file
  referencing attachment variables 78
Index

DSICoEx
   Visual Basic version 75
DSICoExV
   sample 74
DSICoRul
   and VB rule processing 61
   registering ActiveX DLLs 65
DSICoSAM
   sample 73
DSICoTB
   sample 68
   testing rules 63
DSIDiag
   sample 71
DSIENV_* flags 145
DSIEX.C 3, 11
DSIEXW32 program 10
DSIGetSOAPMessage 128
DSIGetSOAPMessageSize 129
DSIGetUniqueString
   in C 8
DSIInit 136
DSIInitInstance 137
DSIInitQueue 138
DSILIB
   passing JVM options 180
DSILIB.H 7
DSILocateAttachVar 112, 113, 139, 140
DSILocateValue 141
DSIOpenAttachCursor 142
DSIParseAttachment 144
DSIQSET_ATTACHMENT
   queue field 7
DSIQSET_REQTYPE
   queue field 7
DSIQSET_UNIQUE_ID
   queue field 7, 8
DSIQSET_USERID
   queue field 7
DSIQueryEnvOptions 145
DSIQueryValueSize 146
DSISendFile 147, 154
DSISendFileAsBuffer 148
DSISendFileAsBufferSize 150
DSIRowset2XML 152
DSIRowset2XMLSize 153
DSISendFile 155
DSISendAttachment
   in C 8
DSIW32.DLL 7
DSIW32.LIB 7
DSRVW32 program 66
DumpDebugInfo
   in Visual Basic 279
DUTTRACE.LOG 66

E

ERR request type 17
error handling
   Visual Basic rules 65
error log
   returning records from 203
error messages
   using ErrorMessage 15
ErrorMessage
   in C 15, 124
   in Visual Basic 280
ErrorMsg
   in C 125
ERS request type 17
ESS request type 17

F

fat clients
   in Visual Basic 245
files
   generating unique names 16
   removing temporary files 16
FindInQueue
  in C 8, 126
  in Visual Basic 281
firewalls
  debug log file 201
  IRLFileFTP 195
  IRLInitFTP 202
flags
  DSIENV_9 145
FTP control group 201
FTP servers
  IRLFileFTP 195
FTPCONNECTIONS variable 202
FTRule
  defined 186
functions
  list of C API 80

G

  generating unique file names 16
GetAttach 78
  GetAttachment
    in Visual Basic 282
GetAttachmentAll
  in Visual Basic 284
GetAttachRecSet
  in Visual Basic 286
GetAttachVarSet
  in Visual Basic 288
GetFirstFromQueue
  in C 127
GetPriority
  in Visual Basic 290
GetQError
  in C 16, 130
GetQField
  in C 9, 16, 131
  in Visual Basic 291
GetQFieldLength
  in C 16, 133
  in Visual Basic 292
GetQueueRec
  in C 8, 134
  in Visual Basic 293
GetReqType
  in Visual Basic 295
GetStatus
  in Visual Basic 296
GetUniqueID
  in Visual Basic 297
GetUniqueIDLength
  in Visual Basic 298
GetUniqueString
  in C 8, 16, 135
  in Visual Basic 299
GetUserID
  in Visual Basic 300
global methods 64
  GlobalDataClean method 65
  GlobalDataCreate method 65
  GlobalDataDestroy method 65
  GlobalDataRead method 65
  GlobalDataSize method 65
  GUIDs 66

H

  handling errors
    in Visual Basic 246
Heap Agent 12
hInstance parameter 13
how the system processes rules 12
HRESULT 246
HTML files
  IRCUnloadPage formatting rule 15
Index

IDS Servers
  storing data for multiple servers 64
INI request type 17
Init
  in C 7, 11, 14, 136
  in Visual Basic 301
initializing lists 12
InitInstance
  in C 7, 11, 14, 137
  in Visual Basic 302
InitQueue
  in C 7, 138
  in Visual Basic 303
InitSession
  in Visual Basic 304
Instance
  in Visual Basic 334
interfaces
  C 79, 171
  Java 76, 171
  Visual Basic 61, 245
Internet Document Server
  how rules are processed 13
  queues 4
  required queue fields 7
INTERNETSESSION variable 202
IRCInit 238
IRCPrint 239
IRCRequest 240
IRCResult 241
IRCSendVersion 242
IRCUUnloadPage 15, 241, 244
IRLClearLog 191
IRLCopyAttachment 192
IRLFileFTP 195
IRLInit 194
IRLInitFTP 202
IRLLog 203
IRLPurgeCache 16, 204
IRLSendVersion 206
IRLStatistics 208
Java
  API classes 182
  APIs 76, 171
  DSI API 3
  JNI 180
  JVM 180
  messaging library 179
JavaBean
  using 172
LocateAttachVar
  in C 15, 112, 113, 139, 140
  in Visual Basic 306
LocateValue
  in C 15, 141
  in Visual Basic 66, 307
LocateValueObj
  in Visual Basic 309
log files
  clearing 17
log4j 181
loops
  and Visual Basic rules 66
memory violations 12
messages
  getting the size 129
  retrieving from memory 128
Microsoft
   Visual Basic 5 61
MQSeries
   Java messaging library 179
multiple class modules 61

OLEVIEW program 63
OpenAttachCursor
   in C 15, 142
   in Visual Basic 311

ParseAttachment
   in C 9, 144
   in Visual Basic 312
Path option 199
   global data 64
performance
   and Visual Basic 66
processing rules
   template 14
   writing 12
ProgIDs
   and Visual Basic rules 66
Project Description field 66
properties
   Instance 334
   Signature 335
   TraceEnable 336
   TracePath 337
proprietary attachment format 16
   C 8
pszId parameter 8
pszParms parameter 14
public functions 64
PurgeCache 204

QueryEnvOptions
   in C 145
QueryValueSiz rule
   in Visual Basic 313
QueryValueSize
   in C 15, 146
queues
   accessing using C 16
   attachments 15, 235
   fields 7
   InitQueue 138
   overview 4
   processing rules 13
   time spent in 237

RAD request type 17
registering
   ActiveX DLLs 62
   relay daemon 17
RemoveOnGet option 198
RemoveOnPut option 198
removing
   temporary files 16
REQTYPE
   DOCserv INI file 64
request types
   creating rules for reserved 16
RRS request type 17
RSS request type 17
rule wizard
   example 66
Index

rules
  checking status 17
  client 225
  creating 13
  list 13
  memory violations 12
  processing 12, 183
  reserved request types 16
  server 184
  template 14
  VB processing 61
  writing 12

S

samples
  VB rule wizard 66
  Visual Basic 68
SAR request type 17
SCS request type 17
SDK
  Java APIs 171
  using the Internet Document Server SDK 1
servers
  log file layout 228
  rules 184
  statistics 208
SERVERTIMESPENT attachment variable 237
SetAttachment
  in Visual Basic 314
SetPriority
  in Visual Basic 316
SetQField
  in C 8, 16, 156
  in Visual Basic 317
SetReqType
  in Visual Basic 318
SetStatus
  in Visual Basic 319
SetUserID
  in Visual Basic 321
SetUniqueID
  in Visual Basic 320

T

temporary files
  creating 16
  removing 16
Term
  in C 7, 11, 159
  in Visual Basic 325
TermInstance
  in C 7, 11, 160
  in Visual Basic 326
TermQueue
  in C 161
  in Visual Basic 327
TermSession
  in Visual Basic 328
threads
  and Visual Basic rules 66
  InitInstance C function 137
TOTAlTIMESPENT attachment variable 237
Trace
  in Visual Basic 329
TraceAttach
  in Visual Basic 330
TraceEnable property 336
TraceEnableRule
  in Visual Basic 331
TraceList
  in Visual Basic 332
TracePath property 337
TraceSnapshot
  in Visual Basic 333
tracing
  in Visual Basic 336
transactions
  time spent in queues 237
troubleshooting
  Visual Basic rules 63

U
ulMsg parameter 14
ulOptions parameter 14
unique file names, generating 16
UNK request type 17
using
  the DSI APIs with C 7
  the Internet Document Server SDK 1
Visual Basic
  61
  Visual Basic prototypes and examples 246
utility functions
  for C 16

V
variables
  attachment 15
  creating, accessing, and destroying using C 15
  stem 15
version information
  IRCSendVersion 242
  IRLSendVersion 206

Visual Basic
  class files 62
  handling errors 246
  installing the rule wizard 62
  list of API functions 248
  loops 66
  performance 66
  projects 62
  rule wizard 61, 66
  samples 68
  sharing violations 66
  states 66
  test bed sample 68
  testing 65
  threads 66
  troubleshooting 63
  using prototypes and examples 246
  workgroups 62
  writing processing rules 61

Visual C++ 5.0 debugger 65

W
web applications
  firewalls 199
writing
  processing rules 12
  Visual Basic rules 61