

# XLNT<sup>®</sup> Reference Manual

Version 5

Advanced Systems Concepts, Inc.

---

**Eleventh Printing: July 2014 [V5.0]**  
**Tenth Printing: January 2006 [V4.0]**  
**Ninth Printing: April 2001 [V3.0]**  
**Eighth Printing: June 2000 [V2.0 SP4]**  
**Seventh Printing: September 1999 [V2.0 SP4]**  
**Sixth Printing: May 1999 [V2.0 SP3]**  
**Fifth Printing: February 1999 [V2.0 SP2]**  
**Fourth Printing: September 1998 [V2.0 SP1]**  
**Third Printing: May 1998 [V2.0]**  
**Second Printing: January 1998 (V1.1 (SP4))**  
**First Printing: March 1997 (V1.1)**

The information in this document is subject to change without notice and should not be construed as a commitment by Advanced Systems Concepts, Inc. (ASCI). ASCI assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under license, and may only be used or copied in accordance with the terms of such license. The terms and conditions of license are furnished with the product in both hard copy as well as electronic form.

ASCI logo and XLNT are registered trademarks of Advanced Systems Concepts, Inc.

ASCI, XLNT logo are trademarks of Advanced Systems Concepts, Inc.

Microsoft, Windows NT, ActiveX and Systems Management Server are trademarks of Microsoft Corporation.

---

**Copyright © 1997-2014, Advanced Systems Concepts, Inc., Morristown, New Jersey, 07960, All Rights Reserved.**

No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage and retrieval system, without the written permission of Advanced Systems Concepts, Inc.

# Table of Contents

<b>1</b>	<b>LICENSE AGREEMENT</b>	<b>1</b>
<b>2</b>	<b>INTRODUCTION</b>	<b>4</b>
2.1	System Requirements	4
2.2	Product Editions	5
2.3	Installation Procedure	7
2.3.1	Registry Entries	8
2.3.2	XLNT Console Window	8
2.3.3	SET HOST Installation	9
2.3.4	SET HOST Security and Registry	10
2.4	Support	12
2.4.1	XLNT Support Programs	12
<b>3</b>	<b>USING XLNT</b>	<b>13</b>
3.1	Command Interpreter Processing	13
3.2	Interacting with XLNT	13
3.2.1	XLNT Command Syntax	15
3.2.2	Entering an XLNT Command	15
3.2.3	Editing the XLNT Command Line	16
3.2.4	XLNT Mouse Handling	16
3.3	Symbols	18
3.3.1	Defining Foreign Commands	19
3.3.2	Display and Deleting Symbols	21
3.3.3	Declared Symbols	22
3.3.4	Implicit Symbols	24
3.3.5	Labels	24
3.3.6	Supported Datatypes	25
3.3.7	Object Reference	30
3.4	Operators	31
3.4.1	String Operators	31
3.5	Logical Values and Expressions	33
3.5.1	Numeric Operators	34
3.5.2	Logical Operators	34
3.6	Functions	34
3.7	Files and Directories	36
3.7.1	UNC Specifications	36
3.7.2	FTP URL Specifications	37
3.8	Date and Time Format	38
3.8.1	Absolute Time	38

3.8.2	Delta Time .....	39
3.8.3	Combination Time .....	40
<b>3.9</b>	<b>Writing Procedures.....</b>	<b>41</b>
3.9.1	Handling Error Conditions.....	42
3.9.2	Handling Error Conditions from DOS/CMD Programs.....	44
3.9.3	Handling Control/C Interruptions .....	44
<b>3.10</b>	<b>Command Procedure Input .....</b>	<b>45</b>
<b>3.11</b>	<b>In-Script Data Processing.....</b>	<b>46</b>
<b>3.12</b>	<b>Command Procedure Output.....</b>	<b>47</b>
<b>3.13</b>	<b>Reading and Writing Files .....</b>	<b>47</b>
<b>3.14</b>	<b>Controlling Execution Flow .....</b>	<b>48</b>
<b>4</b>	<b>INTERACTIVE DEVELOPMENT ENVIRONMENT AND DEBUGGING FACILITY</b>	<b>51</b>
4.1	Starting the IDE.....	51
4.2	Creating and Editing a Script.....	52
4.3	Debugging an XLNT Script .....	53
4.4	Tools and Customization of the IDE .....	57
<b>5</b>	<b>XLNT AND DOS/WIN CMD .....</b>	<b>60</b>
<b>6</b>	<b>USING XLNT AS A CGI .....</b>	<b>61</b>
6.1	Extracting QUERY_STRING.....	62
6.2	Installing the Sample CGI Script.....	62
<b>7</b>	<b>USING XLNT FOR ACTIVEX AND WINDOWS SCRIPTING HOST PROCESSING</b>	<b>66</b>
<b>8</b>	<b>USING XLNT FOR LOGON SCRIPT PROCESSING.....</b>	<b>70</b>
<b>9</b>	<b>FILE-PROCESSING COMMAND SUMMARY.....</b>	<b>71</b>
9.1	APPEND Command .....	71
9.2	CLOSE Command.....	73
9.3	COPY Command .....	74
9.4	CREATE Command.....	77
9.5	CREATE/DIRECTORY Command.....	78

9.6	DELETE Command .....	79
9.7	DELETE/SYMBOL Command .....	81
9.8	DIFFERENCES Command .....	82
9.9	DIRECTORY Command .....	85
9.10	DISABLE .....	90
9.11	DISCONNECT FILE Command .....	91
9.12	DISCONNECT SESSION Command .....	92
9.13	DUMP Command.....	93
9.14	EDIT Command.....	95
9.15	ENABLE .....	96
9.16	MAIL SEND Command .....	97
9.17	MAIL SEND/SMTP Command.....	98
9.18	OPEN Command .....	100
9.19	PRINT Command .....	102
9.20	READ Command .....	103
9.21	RENAME Command.....	105
9.22	SCHEDULE Command.....	107
9.23	SEARCH Command.....	109
9.24	TYPE Command.....	113
9.25	WRITE Command.....	114
<b>10</b>	<b>DISTRIBUTED FILE SYSTEM COMMANDS .....</b>	<b>115</b>
10.1	DFS ADD .....	115
10.2	DFS REMOVE .....	116
10.3	DFS SET .....	117
10.4	DFS SHOW .....	118
<b>11</b>	<b>PRINTER MANAGEMENT COMMANDS.....</b>	<b>120</b>
11.1	MANAGE ADD DRIVER.....	120

11.2	MANAGE DELETE DRIVER.....	122
11.3	MANAGE SHOW DRIVER .....	123
11.4	MANAGE SHOW MONITOR.....	124
11.5	MANAGE SHOW PORT .....	125
11.6	MANAGE ADD PRINTER.....	127
11.7	MANAGE CONNECT PRINTER.....	130
11.8	MANAGE DELETE PRINTER .....	131
11.9	MANAGE DISCONNECT PRINTER.....	132
11.10	MANAGE SET JOB .....	133
11.11	MANAGE SET PRINTER.....	135
11.12	MANAGE SHOW PRINTER.....	138
<b>12</b>	<b>SECURITY COMMANDS .....</b>	<b>140</b>
12.1	SECURITY CREATE .....	140
12.2	SECURITY DELETE.....	143
12.3	SECURITY MODIFY .....	145
12.4	SECURITY SHOW .....	148
<b>13</b>	<b>SERVICE COMMANDS.....</b>	<b>150</b>
13.1	CONFIGURE SERVICE Command.....	150
13.2	INSTALL SERVICE Command .....	153
13.3	PAUSE SERVICE Command.....	156
13.4	REMOVE SERVICE Command .....	157
13.5	RESET SERVICE Command.....	158
13.6	RESUME SERVICE Command.....	159
13.7	SHOW SERVICE Command .....	160
13.8	START SERVICE Command.....	161
13.9	STOP SERVICE Command .....	162
<b>14</b>	<b>SHARE COMMANDS .....</b>	<b>163</b>

14.1	SHARE ADD .....	163
14.2	SHARE DELETE.....	164
14.3	SHARE SET .....	165
14.4	SHARE SHOW .....	166
<b>15</b>	<b>ADS COMMANDS .....</b>	<b>167</b>
15.1	ADS CREATE.....	167
15.2	ADS DELETE .....	170
15.3	ADS MODIFY.....	171
15.4	ADS SHOW .....	174
<b>16</b>	<b>LANGUAGE STATEMENTS .....</b>	<b>175</b>
16.1	= (Assignment Statement) .....	175
16.2	:= (String Assignment).....	177
16.3	@ (Execute Procedure) Command .....	178
16.4	! (Comment).....	179
16.5	ABORT Command .....	180
16.6	CALL Command .....	181
16.7	CLS Command.....	183
16.8	DEBUGBREAK Command .....	183
16.9	DECK Command.....	184
16.10	DECLARE FUNCTION Command .....	185
16.11	DECLARE SYMBOL Command.....	187
16.12	DOS Command .....	189
16.13	ENDFOR Statement .....	190
16.14	ENDFOREACH Statement.....	191
16.15	ENDSTRUCTURE Statement .....	192
16.16	ENDSUBROUTINE Command.....	193
16.17	ENDUNTIL Statement .....	194

16.18	ENDWHILE Statement.....	195
16.19	EOD Statement.....	196
16.20	EXIT Command.....	197
16.21	FOR Command .....	198
16.22	FOREACH Command.....	199
16.23	GOSUB Command.....	201
16.24	GOTO Command.....	202
16.25	HELP Command.....	203
16.26	IF Command.....	204
16.27	INQUIRE Command .....	206
16.28	ITERATE Command.....	207
16.29	KILL Command.....	208
16.30	LEAVE Command.....	210
16.31	LOGOUT Command .....	210
16.32	ON Command.....	211
16.33	RECALL Command .....	212
16.34	RETURN Command.....	213
16.35	RUN Command.....	214
16.36	SPAWN Command .....	216
16.37	STRUCTURE Command .....	217
16.38	SUBROUTINE Command .....	219
16.39	UNTIL Command.....	220
16.40	WAIT Command .....	221
16.41	WHILE Command.....	222
16.42	XC Command.....	223
<b>17</b>	<b>SET / SHOW COMMANDS .....</b>	<b>224</b>
17.1	SET DEFAULT.....	224

17.2	SET DOSERROR Command .....	225
17.3	SET FILE Command.....	226
17.4	SET HOST Command.....	228
17.5	SET MESSAGE Command.....	230
17.6	SET ON Command.....	231
17.7	SET PERMISSIONS Command.....	232
17.8	SET PREFERENCE Command.....	236
17.9	SET PROCESS/PRIORITY.....	238
17.10	SET PROMPT.....	239
17.11	SET SYMBOL/SCOPE .....	240
17.12	SET TIME Command .....	240
17.13	SET VARIABLE Command .....	241
17.14	SET VERIFY.....	242
17.15	SHOW DEFAULT Command .....	243
17.16	SHOW DEVICE.....	244
17.17	SHOW DLLS.....	246
17.18	SHOW HOST .....	247
17.19	SHOW LOCALTIMEZONE .....	248
17.20	SHOW MACHINE .....	249
17.21	SHOW MEMORY .....	250
17.22	SHOW PERMISSIONS.....	251
17.23	SHOW PREFERENCES.....	253
17.24	SHOW PROCESS.....	254
17.25	SHOW SERVER/FILES .....	256
17.26	SHOW SESSION .....	257
17.27	SHOW SYMBOL.....	258
17.28	SHOW SYSTEM.....	259
17.29	SHOW TIME .....	260

17.30	SHOW USER .....	261
17.31	SHOW VARIABLE .....	262
<b>18</b>	<b>BUILT-IN (OR LEXICAL) FUNCTIONS.....</b>	<b>263</b>
18.1	F\$ADDREGISTRY Lexical .....	264
18.2	F\$CHANGeregistry Lexical .....	265
18.3	F\$CHECKLIBRARY Lexical.....	266
18.4	F\$CVSI Lexical.....	267
18.5	F\$CVTIME Lexical .....	268
18.6	F\$CVUI Lexical .....	270
18.7	F\$DELETERegistry Lexical.....	271
18.8	F\$DIRECTORY Lexical .....	272
18.9	F\$EDIT Lexical.....	273
18.10	F\$ELEMENT Lexical.....	274
18.11	F\$ENUMDOMAIN Lexical.....	275
18.12	F\$ENUMGROUP Lexical .....	276
18.13	F\$ENUMMACHINE Lexical.....	277
18.14	F\$ENUMSHAREPOINT Lexical .....	279
18.15	F\$ENUMUSER Lexical.....	280
18.16	F\$ENVIRONMENT Lexical.....	281
18.17	F\$EXTRACT Lexical .....	282
18.18	F\$FILE_ATTRIBUTES Lexical.....	283
18.19	F\$FORMAT Lexical.....	285
18.20	F\$FORMATDATE Lexical.....	286
18.21	F\$FORMATTIME Lexical .....	288
18.22	F\$FREELIBRARY Lexical.....	289
18.23	F\$GETBQI Lexical.....	290
18.24	F\$GETDVI Lexical.....	294

18.25	<b>F\$GETDVIEX</b> .....	296
18.26	<b>F\$GETHOSTBYADDR Lexical</b> .....	298
18.27	<b>F\$GETHOSTBYNAME Lexical</b> .....	299
18.28	<b>F\$GETJPI Lexical</b> .....	300
18.29	<b>F\$GETLASTERROR Lexical</b> .....	302
18.30	<b>F\$GETSYI Lexical</b> .....	303
18.31	<b>F\$GETVARIABLE Lexical</b> .....	305
18.32	<b>F\$GROUPINFO Lexical</b> .....	306
18.33	<b>F\$INTEGER Lexical</b> .....	308
18.34	<b>F\$LENGTH Lexical</b> .....	309
18.35	<b>F\$LOADLIBRARY Lexical</b> .....	310
18.36	<b>F\$LOCALTIMEZONE Lexical</b> .....	311
18.37	<b>F\$LOCATE Lexical</b> .....	312
18.38	<b>F\$LOOKUPREGISTRY Lexical</b> .....	312
18.39	<b>F\$MESSAGE Lexical</b> .....	314
18.40	<b>F\$MODE Lexical</b> .....	315
18.41	<b>F\$MSGBOX Lexical</b> .....	316
18.42	<b>F\$PARSE Lexical</b> .....	319
18.43	<b>F\$PERMISSIONS Lexical</b> .....	320
18.44	<b>F\$PID Lexical</b> .....	322
18.45	<b>F\$RANDOM Lexical</b> .....	324
18.46	<b>F\$READEVENT Lexical</b> .....	325
18.47	<b>F\$REPLACE Lexical</b> .....	327
18.48	<b>F\$REPORTEVENT Lexical</b> .....	328
18.49	<b>F\$SEARCH Lexical</b> .....	329
18.50	<b>F\$SEQARRAY Lexical</b> .....	330
18.51	<b>F\$SERVICE_STATUS Lexical</b> .....	331
18.52	<b>F\$STRING Lexical</b> .....	333

18.53	F\$TIME Lexical .....	334
18.54	F\$TYPE Lexical .....	335
18.55	F\$USERINFO Lexical .....	336
18.56	F\$UTCTIME Lexical.....	338
18.57	F\$VERIFY Lexical .....	339
<b>19</b>	<b>SAMPLE XLNT PROCEDURES .....</b>	<b>340</b>
19.1	Administrator’s Helper Procedure.....	340
19.2	View Dormant Accounts Procedure .....	343
19.3	Simple F\$SEARCH Example Procedure .....	344
19.4	Sample LOGIN Script .....	345
19.5	List Windows NT Accounts Machine/Domain Procedure.....	346
19.6	Process INI Files Procedure .....	348
19.7	Machine Reboot Procedure.....	349
19.8	Display System Information Procedure .....	350
19.9	Registry Procedure .....	350
19.10	Enumerate Domain/Machine Procedure .....	351
19.11	Enumerate Drives (WSH Script) .....	353
19.12	Display Environmental Variables (WSH).....	353
19.13	Optimized Display Environmental Variables (WSH).....	354
19.14	Shortcut Creation Script (WSH) .....	354
19.15	Sample Script Using Microsoft WORD Object.....	354
<b>20</b>	<b>SALES AND TECHNICAL SUPPORT .....</b>	<b>356</b>
<b>21</b>	<b>APPENDIX A – ACTIVE DIRECTORY COUNTRY CODES</b>	<b>357</b>
<b>22</b>	<b>INDEX.....</b>	<b>366</b>

## Table of Figures

Figure 1. XLNT Interactive Console Window.....	4
Figure 2. XLNTNET Service Characteristics .....	9
Figure 3. XLNT Cut & Paste .....	17
Figure 4. NET USE Foreign Command.....	20
Figure 5. DECLARE SYMBOL .....	23
Figure 6. DECLARE FUNCTION statement .....	35
Figure 7. IDE Initial Window .....	51
Figure 8. IDE Script View .....	52
Figure 9. IDE Edit Menu .....	53
Figure 10. IDE Debug Menu.....	54
Figure 11. Script Execution Settings Dialog Box .....	54
Figure 12. Breakpoints Dialog Box .....	55
Figure 13. Insert Breakpoint Dialog Box.....	55
Figure 14. Script View with Breakpoint .....	56
Figure 15. Symbols View .....	56
Figure 16. Symbols Modification Dialog Box.....	57
Figure 17. IDE Tools Menu .....	57
Figure 18. IDE Customize .....	58
Figure 19. IDE Options (Preferences).....	59
Figure 20. CGI Screen Powered by XLNT .....	64
Figure 21. XLNT Source for Sample CGI.....	65
Figure 22. WSH EXCEL Script.....	67
Figure 23. EXCEL Application Window .....	68
Figure 24. SHOW SYSTEM.....	259
Figure 25. F\$MSGBOX Example.....	318

# 1 LICENSE AGREEMENT

## XLNT<sup>®</sup> V5 for Windows Server 2012/2008/2003 and Windows 8/7/XP

The section that follows is the license agreement that governs this product. A separate license agreement also accompanies this document and product.

---

**IMPORTANT—PLEASE READ CAREFULLY.** THIS IS A LEGAL AGREEMENT BETWEEN YOU (EITHER AN INDIVIDUAL OR AN ORGANIZATION) AND ADVANCED SYSTEMS CONCEPTS, INC., FOR THE SOFTWARE PRODUCT IDENTIFIED ABOVE, WHICH INCLUDES COMPUTER SOFTWARE AND RELATED DOCUMENTATION. BY INSTALLING, OR OTHERWISE USING THE SOFTWARE, YOU ACCEPT ALL THE TERMS AND CONDITIONS OF THIS AGREEMENT.

Advanced Systems Concepts, Inc., a New Jersey Corporation, whose address is 1180 Headquarters Plaza, Morristown, NJ, 07960, USA ("ASCI") hereby grants to you a non-exclusive license (a "License") to use the software identified above (the "Software") and the accompanying printed materials and User Manual (the "Documentation") on the terms set forth below.

**1. GRANT OF LICENSE.** This license grants you the following rights.

- **Software.** Except as set forth below, you may use the Software on a single computer. This license is applicable to the following XLNT Product Editions: Professional Edition, Standard Edition, and Run-Time License Edition.
- **Run-Time License Edition.** This term is specific to the XLNT Run-Time Edition License ("RTLE"). The RTLE Software is provided as a specific and separate type of software installation known as the RTLE Software Kit. You are also provided with a special RTLE Serial Number that must be used when installing the RTLE Software. The RTLE Software Kit when coupled with a valid RTLE Serial Number allow encoded client-side XLNT scripting capability. No development capability is offered as part of this Software. You may install the RTLE Software on any computer system(s) that your Company owns, rents or leases where the computer system(s) are located in Company owned, rented or leased office space. In the event of a computer system rental or lease, you acknowledge and agree that the RTLE software must be de-installed and otherwise removed prior to sending the computer system back to the vendor or lessor. You are expressly forbidden to use any other XLNT Product Serial Numbers as a replacement for the RTLE Serial Number.
- **Storage/Network Use.** You may store or install a copy of the Software on a storage device, such as a network server, used only to install or run the Software on your computers over an internal network; however, you must acquire and dedicate a license for each separate computer on which the Software is installed or run from the storage device. A license for the Software may not be shared or used concurrently on different computers. You agree to inform all users of the Software of the terms and conditions for its use.
- **License Pack.** If you have acquired this Agreement as part of a License Pack, then you may make the number of additional copies of the Software as authorized on the printed copy of that License Pack, and you may use each copy in the manner described above.
- **Upgrades.** When a Software Upgrade is purchased it must be applied to a previously licensed and installed Software product on a single computer. Purchase of a Software Upgrade does not constitute the right to use the Software on another computer.
- **Transfer.** You may transfer the Software and Documentation to a single recipient on a permanent basis provided you retain no copies of the Software or Documentation (including backup or archival copies) and the recipient agrees to the terms and conditions of this Agreement. If the Software is an upgrade, any transfer must include all prior versions of the Software and Documentation.
- **Dual Media.** If this Software was delivered on more than one form of electronic media, the right to use the Software is still limited to a single computer.

**2. OTHER RIGHTS AND LIMITATIONS.**

- **Licensing Management.** This Software has the ability to regulate its use based on this Agreement and employs a License Management Facility. You understand and acknowledge that the ability to copy the Software and/or to breach the Agreement may be prevented through the use of this License Management Facility. If you purchase a Software Upgrade, you acknowledge that the Software Serial Number contains special logic to ensure that a previous already installed version of the Software is present for installation. Further you agree to keep the Software Serial Number confidential and not to disclose the serial number to anyone outside your Company. You understand and acknowledge that the various XLNT Product Editions provide varying access to the underlying XLNT technology. Any product limitations are noted within the Documentation.
- **Reverse Engineering.** You may not modify, translate, reverse engineer, decompile, disassemble the Software, or make any attempt to

discover the source code to the Software.

- **Copying.** You may not copy the Software or Documentation except as specifically provided by this Agreement.
- **Separation of Components.** The Software is licensed as a single product. You may not separate the Software's component parts for use on more than one computer.
- **Rental.** You may not rent or lease the Software or Documentation.
- **Proprietary Notices.** You may not remove or modify any proprietary notices, labels or marks on the Software or Documentation.

### 3. **TITLE AND COPYRIGHT.**

Title, ownership rights, and intellectual property rights in and to the Software and Documentation shall remain with ASCI. The Software and Documentation is protected by copyright laws of the United States and international copyright treaties. You may not copy any of the written materials accompanying the Software unless specifically authorized by ASCI in writing.

### 4. **TERMINATION.**

This License is in effect until terminated. The License will terminate automatically if you fail to comply with the limitations described herein. You may also voluntarily terminate the License. On termination, you must destroy all copies of the Software and Documentation.

### 5. **EXPORT CONTROLS.**

Since this Software is subject to the export control laws of the United States, you may not export or re-export the Software without the appropriate United States and foreign government licenses. You shall otherwise comply with all applicable export control laws and shall defend, indemnify and hold ASCI and all ASCI suppliers harmless from any claims arising out of your violation of such export control laws.

### 6. **MISCELLANEOUS.**

This Agreement represents the complete agreement concerning this license between the parties and supersedes all prior agreements and representations between them. This Agreement may be amended only in writing executed by both parties. **THE ACCEPTANCE OF ANY PURCHASE ORDER PLACED BY YOU IS EXPRESSLY MADE CONDITIONAL ON YOUR ASSENT TO THE TERMS SET FORTH HEREIN, AND NOT THOSE CONTAINED IN YOUR PURCHASE ORDER.** If any provisions of this Agreement is held to be unenforceable for any reason, such provision shall be reformed only to the extent necessary to make it enforceable and the remainder of this Agreement shall nonetheless remain in full force and effect. This Agreement shall be governed by and construed under New Jersey law as such law applies to agreements between New Jersey residents entered into and to be performed within New Jersey, except as governed by Federal law. The application of the United Nations Convention of Contracts for the International Sale of Goods is expressly excluded.

### 7. **U.S. GOVERNMENT END-USERS**

The Software and Documentation are provided with **RESTRICTED RIGHTS**. Use, duplication or disclosure by the U.S. Government is subject to restriction as set forth in subparagraph (c) (1)(ii) of The Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer-Restricted Rights at 48 CFR 52.227-19, as applicable. The contractor/manufacturer is Advanced Systems Concepts, Inc., 1180 Headquarters Plaza, West Tower – Third Floor, Morristown, NJ, 07960, USA.

### 8. **LIMITED WARRANTY**

**ASCI WARRANTS THAT THE MEDIA, IF PROVIDED BY ASCI, IS FREE FROM DEFECTS IN MATERIALS AND WORKMANSHIP FOR SIXTY (60) DAYS FROM THE DATE YOU ACQUIRE THE SOFTWARE. ASCI WARRANTS THAT THE SOFTWARE, AS PROVIDED BY ASCI, WILL PERFORM SUBSTANTIALLY ACCORDING TO THE ACCOMPANYING WRITTEN MATERIALS FOR SIXTY (60) DAYS FROM THE DATE YOU ACQUIRE THE SOFTWARE.**

ASCI's sole liability, and your sole remedy, for any breach of this warranty shall be, in ASCI's sole discretion: (i) to replace your defective media; or (ii) if the above remedy is impractical, to refund the License fee received by ASCI for the Software. Replaced Software shall be covered by this limited warranty for the period remaining under the warranty that covered the original Software, or if longer, for thirty (30) days after the date of shipment to you of the replaced Software. Only if you inform ASCI of your problem with the Software during the applicable warranty period and provide evidence of the date you acquired the Software will ASCI be obligated to honor this warranty. ASCI will use reasonable commercial efforts to replace or refund pursuant to the foregoing warranty within thirty (30) days of being so notified.

**THIS IS A LIMITED WARRANTY AND IT IS THE ONLY WARRANTY MADE BY ASCI. ASCI MAKES NO OTHER WARRANTY,**

REPRESENTATION OR CONDITION, EXPRESS OR IMPLIED, AND EXPRESSLY DISCLAIMS THE IMPLIED WARRANTIES OF MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. THE DURATION OF IMPLIED WARRANTIES OR CONDITIONS, INCLUDING WITHOUT LIMITATION, WARRANTIES OR CONDITIONS OF MERCHANTABILITY AND OF FITNESS FOR A PARTICULAR PURPOSE, IS LIMITED TO THE ABOVE LIMITED WARRANTY PERIOD; SOME JURISDICTIONS DO NOT ALLOW LIMITATIONS ON HOW LONG AN IMPLIED WARRANTY OR CONDITION LASTS, SO LIMITATIONS MAY NOT APPLY TO YOU. NO ASCI AGENT, SUPPLIER OR EMPLOYEE IS AUTHORIZED TO MAKE ANY MODIFICATIONS, EXTENSIONS OR ADDITIONS TO THIS WARRANTY.

This Limited Warranty is void if failure of the Software and/or media, has resulted from accident, abuse, or improper use; if any modifications are made to the Software other than by ASCI; or if you violate the terms of this Agreement.

THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS, AND YOU MAY HAVE OTHER LEGAL RIGHTS THAT VARY FROM STATE TO STATE OR BY JURISDICTION.

**9. LIMITATION OF LIABILITY.**

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, TORT, CONTRACT, OR OTHERWISE, SHALL ASCI OR ITS SUPPLIERS/RESELLERS/DISTRIBUTORS BE LIABLE TO YOU OR ANY OTHER PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES INCLUDING WITHOUT LIMITATION, DAMAGES FOR LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, OR FOR ANY DAMAGES IN EXCESS OF ASCI'S LIST PRICE FOR A LICENSE TO THE SOFTWARE AND DOCUMENTATION, EVEN IF ASCI SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY. FURTHERMORE, SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS LIMITATION AND EXCLUSION MAY NOT APPLY TO YOU.

**For more information about ASCI's licensing policies,** please call ASCI Product Licensing at (973) 539-2260 or via facsimile at (973) 539-3390 or via e-mail at: [admin@advsyscon.com](mailto:admin@advsyscon.com).

## 2 Introduction

XLNT® is a software product developed by Advanced Systems Concepts. XLNT provides a powerful, but easy to use, Enterprise Scripting language to facilitate command-line and batch processing interfaces for the users of the Microsoft Windows 2012/2008/2003 server operating systems and Windows 8/7/XP workstation operating systems. XLNT also provides a command shell that you can use as an interactive environment for running programs and performing various tasks.

If you are new to XLNT or command/scripting languages in general may we recommend that you initially start reading the "Getting Started with XLNT" manual. This manual is much more tutorial in nature and provides a good starting point for learning the language.

### 2.1 System Requirements

XLNT requires a valid Microsoft Windows operating system as noted in the Introduction section, running on a compatible Intel-based processor. The product has no special memory requirements and the installation procedure requires between 2 and 8 megabytes of hard disk depending on Product Edition and platform selected.

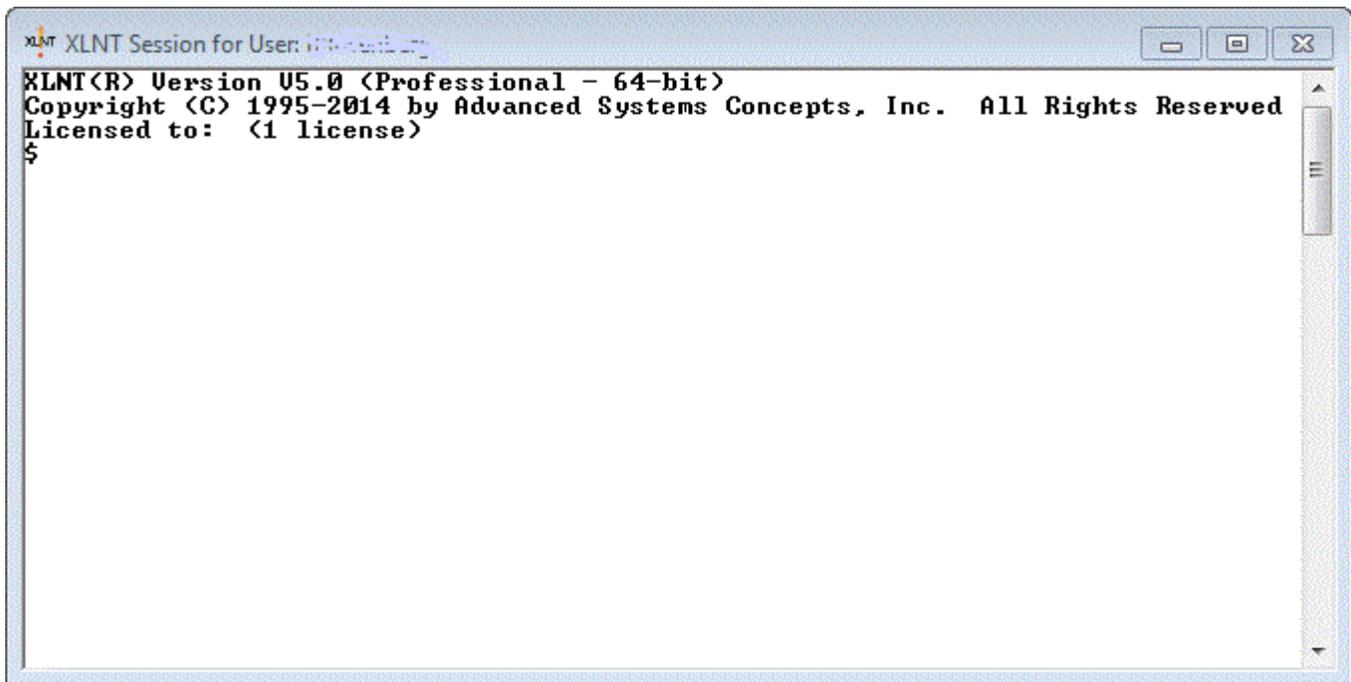


Figure 1. XLNT Interactive Console Window

## 2.2 Product Editions

XLNT is licensed in several product variations. Specific license terms may be found in the section on “Licensing” at the beginning of this manual (License terms are also provided through the action of product installation). The following types of product editions are provided:

1. Professional Edition.
2. Standard Edition.
3. Run-Time License Edition.

### Professional Edition

The Professional Edition is provided for the administrative and/or application developer scriptwriter who needs to create scripts or procedures for various system and/or application requirements. In particular, this edition provides an Interactive Development Environment (IDE), Script Debugging and the ability to compile scripts into an executable image (using the XC command). Once a script has been compiled into an encoded executable image, users of this edition may elect to purchase the Run-Time License Edition for deployment of created executable scripts on the other computer systems owned by the user. Users of the Professional Edition also enjoy all the features of the Standard Edition.

## **Standard Edition**

The Standard Edition provides all the features of XLNT described in this manual except those indicated under the Professional Edition heading.

## **Run-Time License Edition**

The Run-Time License Edition (RTLE) requires the purchase of at least one (1) Professional Edition product. The RTLE provides an environment for the execution of compiled XLNT scripts only. No interactive console level command execution is possible. Several other restrictions concerning this edition are noted below. The RTLE provides an inexpensive option for XLNT scriptwriters to deploy compiled scripts for execution on other machines owned by the user. Using the RTLE a scriptwriter can write and compile a Login script, for example, and then deploy that script to the other machines owned by the company.

1. No command level 0 or interactive command support.
2. No Services (no SET HOST Server or SECURITY services)
3. RTLE can only execute .EXE scripts. Source level XCP/XCS scripts cannot be invoked.
4. Several components omitted due to the above restrictions. (No Online Help, "About Box", Splash screens).

## 2.3 Installation Procedure

The installation media is supplied in two (2) forms for use with two (2) platforms.

XLNTV5x86.MSI or XLNTV5x86.EXE is to be used for x86 (32-bit) platforms. The MSI kit allows System Administrators more flexibility in terms of silent installation and administrative transforms. The EXE kit incorporates the MSI version within the executable and is the simplest form in which to install the XLNT product.

XLNTV5x64.MSI or XLNTV5x64.Exe is to be used for x64 (64-bit) platforms.

To install XLNT manually, run the appropriate installation file above. ASCI recommends that you perform this installation using the Administrator account or a similar account with administrative rights.

As part of the installation, you will be asked to supply your name, company name (if applicable), and the license serial number of your XLNT kit. This serial number is provided in the documentation you received with the kit. If you are installing a license upgrade you must have the previous XLNT product installed using a valid license serial number. After supplying this information, a license agreement will appear corresponding to the type of serial number entered. To continue the installation you must agree to the terms of this license by clicking on the YES button. If you do not agree, click NO to terminate the installation procedure. (Please note that a copy of the license agreement is included as part of the kit in an area that does not require you to break any seals or otherwise agree to terms that you will not first have had an opportunity to examine).

When the installation continues, you may enter a device and directory (or folder) into which XLNT will be installed. The default installation device and directory is "C:\Program Files\ASCI\XLNT". You may optionally also enter a folder name to indicate the placement of the product (the default folder name is XLNT).

After the installation is complete, XLNT will be installed in its own program group (or folder). Within this group, you will see icons that allow you to run the XLNT interactive command shell, access on-line help, read the release notes and, if you wish, de-install the product. To start up an XLNT session, choose the **XLNT** icon.

## 2.3.1 Registry Entries

Like most Windows products, XLNT uses the Registry to store important product information. The information is located in the **HKEY\_LOCAL\_MACHINE** hive under a key called

**SOFTWARE\ASCI\XLNT\CURRENT\_VERSION**

The following keys are defined by Installation along with their associated defaults (if any):

**InstallPath** : Product files location

**MaxLevel** : Maximum command levels (32)

**MaxRecallBuffers** : Maximum commands in Recall Buffer (20)

XLNT supports the concept of a common procedure that can be executed by everyone using XLNT as well as a “user-by-user” individual procedure that allows each user to perform unique actions. The system-wide login command procedure is kept in the **HKEY\_LOCAL\_MACHINE** registry under the key:

**SOFTWARE\ASCI\XLNT\CURRENT\_VERSION\SYLOGIN**

By default, the XLNT Installation process installs a template SYLOGIN.XCP file into the XLNT destination directory. Please examine this file (particularly if you are going to allow incoming SET HOST remote users). The SYLOGIN template procedure is very useful for providing common scripting actions for all XLNT users.

The user-specific command procedure is kept in the **CURRENT\_USER** hive under the key:

**SOFTWARE\ASCI\XLNT\CURRENT\_VERSION\USERLOGIN**

By default, there is no USERLOGIN procedure established. To establish the USERLOGIN.XCP procedure for a user, simply create a USERLOGIN.XCP and have the registry key point to it. XLNT will execute the SYLOGIN.XCP procedure first (if one exists and is enabled for use) and then execute the USERLOGIN.XCP procedure if it exists and is enabled. SYLOGIN and USERLOGIN allow the System Administrator to create both common and individual procedures that are guaranteed to execute when a user invokes XLNT.

Note: SYLOGIN and USERLOGIN procedures are not used by the Run-Time License Edition.

Note: ASCI recommends that you use the SET PREFERENCES command to change XLNT Registry Keys and/or Values. Also, please note that the defaults mentioned only apply to new installations of XLNT. When you upgrade XLNT with the installation of a Service Pack your existing preferential registry settings are not changed.

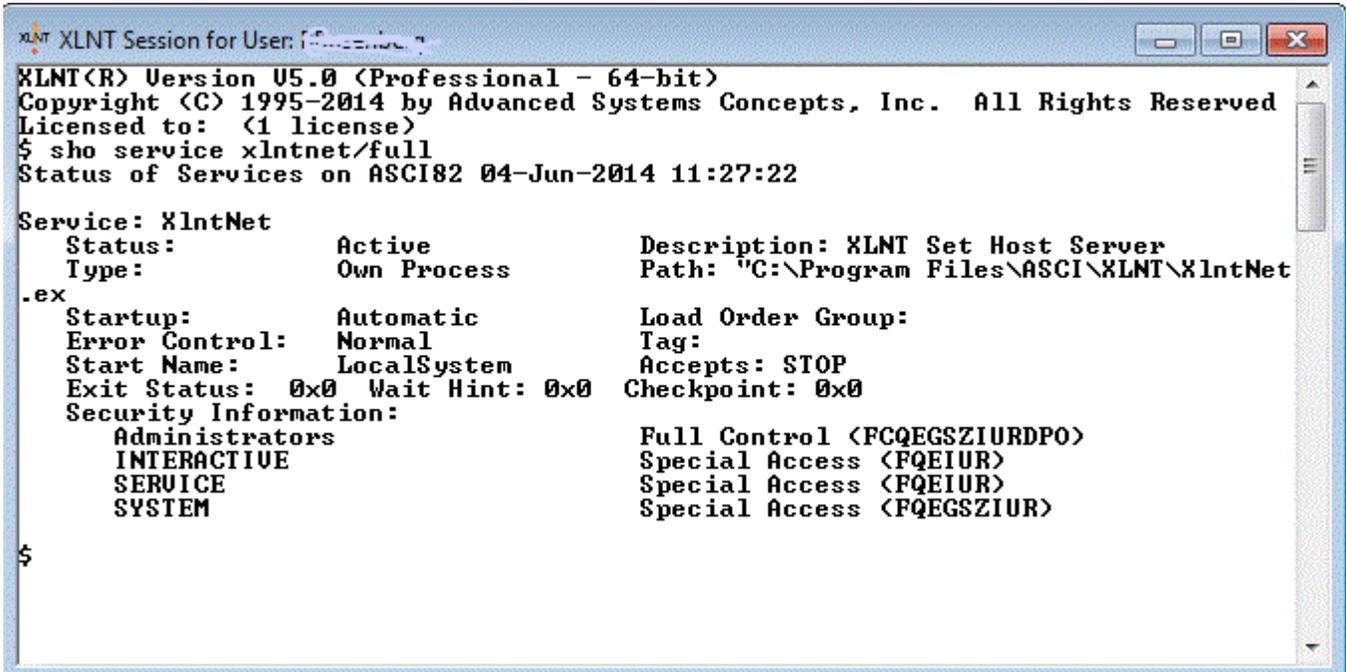
## 2.3.2 XLNT Console Window

When you execute XLNT by clicking on its icon, a console window will appear on your monitor. The console window is similar in appearance to that of the DOS or CMD command interpreter. By default, the background is white and the foreground text is black. This can be changed by clicking on the window’s handle (upper left corner) and changing its Properties. XLNT creates a console window with a screen buffer size of 100. This will allow you to scroll forward and/or backwards through previously executed commands and data. Please note that the PAGE qualifier uses the window buffer size (and not the screen buffer size) so /PAGE will stop every 25 lines. The size of the Screen Buffer Size can also be changed through the Properties menu item.

The SET PREFERENCES command can also be used to change the console’s appearance and other characteristics. In addition, SET PREFERENCE/CURRENT/**SAVE** can be used to save the current console properties for later restoration by XLNT.

### 2.3.3 SET HOST Installation

SET HOST refers to a facility within XLNT that allows users to connect to other Windows 2003/XP/2000 systems. Note that XLNT is required to be licensed on both the Client and Server systems you intend to execute the SET HOST on as well as the destination server system. After you have installed XLNT, the files necessary for enabling and starting the SET HOST facility are stored in the installation directory (the default is c:\program files\ascii\xlnt). XLNT installs the SET HOST Service (XLNTNET) as part of the installation. XLNTNET is set for Automatic startup whenever the system reboots. To confirm that the service is started issue the XLNT command, SHOW SERVICE XLNTNET. This will list the service and its current state. To stop the XLNTNET service, either use the Control Panel/Services icon or issue the XLNT command STOP SERVICE XLNTNET. If you want to remove the service entirely from your system (after you installed it as per the instructions above), issue the XLNT command REMOVE SERVICE XLNTNET.



```
XLNT Session for User: Administrator
XLNT<R> Version 05.0 (Professional - 64-bit)
Copyright (C) 1995-2014 by Advanced Systems Concepts, Inc. All Rights Reserved
Licensed to: (1 license)
$ sho service xlntnet/full
Status of Services on ASCII82 04-Jun-2014 11:27:22

Service: XlntNet
  Status:      Active
  Type:        Own Process
  Description: XLNT Set Host Server
  Path:        "C:\Program Files\ASCII\XLNT\XlntNet
.ex
  Startup:     Automatic
  Error Control: Normal
  Start Name:  LocalSystem
  Exit Status: 0x0 Wait Hint: 0x0
  Security Information:
    Administrators      Full Control (FCQEGSZIURDPO)
    INTERACTIVE         Special Access (FQEIUR)
    SERVICE             Special Access (FQEIUR)
    SYSTEM              Special Access (FQEGSZIUR)
$
```

Figure 2. XLNTNET Service Characteristics

## 2.3.4 SET HOST Security and Registry

Once the SET HOST service is running, you need to enable your machine for remote access. By default, the SYLOGIN.XCP file provided by XLNT during installation, contains the following XLNT commands:

```
$ IF $REMOTE
$ THEN
$     WRITE $STDOUT "Remote Logins not allowed."
$     LOGOUT
$ ENDIF
```

This code sequence indicates that should a remote user login to XLNT, the system-wide login procedure will log that user off immediately. We did this so that customers would be able to completely control remote access to their machine even after the XLNTNET service is started.

Therefore, the above code sequence should be commented out (using the ! comment symbol) if you want to allow remote access to your machine. The code sequence looks like this when commented out:

```
#! IF $REMOTE
#! THEN
#!     WRITE $STDOUT "Remote Logins not allowed."
#!     LOGOUT
#! ENDIF
```

By default, all login/logout's are logged through the Windows NT Event Logger (and thus viewable) under "Application".

There are several security considerations when allowing a remote user to access your machine. SET HOST will prevent any Windows applications from being started through the SET HOST facility. However, since the current version of Windows NT was not designed to support multiple interactive users, all users share all local and remote drive letters. Changes made to a network drive letter by an interactive user are viewable by the remote user and vice-versa.

User preferences are not applied for the remote user due to the use of HKEY\_CURRENT\_USER.

For additional SET HOST security on TCP/IP networks, you might want to change the registered port number for SET HOST from its default.

The registry key in LOCAL\_MACHINE is Software\ASCI\XLNET\Current\_Version\PortNumber.

(Remember to STOP and START the service to enable recognition of the new port number). If the value is changed from its default, then you will need to specify that port number as part of your connection using SET HOST. See the [SET HOST Command](#) for more information.

To disable TCP/IP protocol support, the PortNumber subkey may be set to zero (0).

This facility is useful in that you can prevent the use of TCP/IP, for example, while still allowing internal company connections over named pipes.

**Note:** If you are using a firewall or other proxy server, you will need to install XLNT on the firewall and then SET HOST to/from the firewall to your desired destination system.

## 2.4 Support

### 2.4.1 XLNT Support Programs

Advanced Systems Concepts offers many support programs to assist XLNT evaluators and customers. The following sections detail some of the features when registering your licensed usage with ASCI.

#### 2.4.1.1 Product Warranty

Registered users of XLNT are provided a sixty (60) day period in which an ASCI Support Engineer will be assigned to manage your technical question or problem. You may phone, FAX or E-mail your request to Advanced Systems Concepts. You will be asked for your product serial number (found on the Product Registration Card) when requesting assistance so please keep it handy.

#### 2.4.1.2 Problem Reporting

All users of XLNT may report product deficiencies, enhancement requests, new features, etc **via E-mail**. Problems will be corrected in either a future version of XLNT or in a Service Pack. Service Packs are cumulative and contain all corrections applied from previous Service Packs. E-mail of this nature should be directed to "support@advsyscon.com".

#### 2.4.1.3 Post-Warranty Support

Advanced Systems offers several plans for ensuring that you always have assistance after the product warranty period has expired. ASCI provides three plans for product support: Priority Technical Support, Priority Technical Support Plus and Incident Packs. The first two are annual maintenance plans designed to support and assistance your use of XLNT. The third plan provides support on a transactional or incident basis. ASCI also provides an annual plan where future upgrades and/or new versions are included as part of the plan.

Most of these plans need to be purchased at the time you licensed the software, so please contact ASCI or your local distributor for more information.

## 3 Using XLNT

Once you have established an XLNT session, you can tell the product to perform some action for you by issuing a **command**. The command will be carried out by the **command language interpreter** (or **shell**). This program will accept your commands from the keyboard, or, indirectly, from a command procedure, analyze them, and execute them.

### 3.1 Command Interpreter Processing

The command interpreter is installed in an XLNT applications group, along with the other facilities of the product. When its icon is selected, a *console window* will be made active. Within the console, startup notices will be displayed and you will be prompted to begin entering commands. The prompting symbol will be a dollar sign followed by a space: **\$** .

You may change this default prompt, at any time, to whatever string you desire by entering the *SET PROMPT* command. For example, if you wish to use the left angle bracket as a prompt, the following command will accomplish that:

```
SET PROMPT "> "
```

XLNT implements the concept of *command procedures*. A command procedure is a sequential text file that contains one or more XLNT commands. A command procedure is essentially a script of one or more valid XLNT commands. A command procedure is invoked at the interactive prompt level via the **@** command. When an **@** command is detected, XLNT will attempt to open the command procedure file and begin executing its commands. It will continue to do so until an *EXIT* command or the end of file is encountered.

A command procedure can invoke another command procedure. XLNT will allow up to 32 levels of command procedure nesting. As each command procedure is started, the nesting level will be incremented. Likewise, the number will be decremented when the command procedure completes. Level zero is the interactive prompt level.

The *LOGOUT* command, executed at any nesting level, will end the XLNT session and cause the command interpreter process to exit.

The default extension for a command procedure file is *.XCP* (for XLNT Command Procedure). If you omit the file extension when you invoke a procedure, XLNT will assume that the extension is *.XCP*. For example, the following commands invoke the same command procedure file:

```
$ @mycommands.xcp
```

```
$ @mycommands
```

A suitably privileged user may define a command procedure that is to be executed whenever a user logs into the system. Additionally, a user may define his/her own command procedure that will be executed whenever XLNT is invoked. The path names of each of these files are kept in the [Windows NT Registry](#), along with other product-related information.

### 3.2 Interacting with XLNT

XLNT (eXtended Language for Windows) is a command shell and scripting language that allows you to perform various tasks and operations.

XLNT supports several modes of operation:

In **interactive mode**, you enter commands through the XLNT Console Window and XLNT interacts with you in turn. Two types of interactive modes are possible through XLNT. **Local** refers to your interaction through the XLNT Console Window on your local display. **Remote** refers to your interaction through the SET HOST facility.

In **batch mode**, you create a script of XLNT statements called a command procedure that is executed separate and distinct from any interactive XLNT session you may have. This command procedure is submitted to the batch processing

system ActiveBatch and becomes a **batch job**.

In **ActiveX mode**, Microsoft's Internet Information Server (IIS) or the Windows Scripting Host (WSH) facilities execute an XLNT scriptlet as part of the ActiveX technologies. This method of product usage allows the scriptwriter to invoke various built-in COM objects provided by Microsoft, Advanced Systems Concepts and other third-party software providers. XLNT scripts can also be invoked as embedded scriptlets within HTML/DHTML scripts.

In **CGI mode**, an Internet HTML Server creates a separate process by which an XLNT script may be invoked for CGI processing. This mode is distinct from batch or non-interactive process since the CGI script must adhere to certain restrictions and limitations governing the use of CGI.

In **non-interactive or other mode**, a command procedure may be executed similar to that of a batch job although the procedure is not actually executing under the auspices of a batch system.

### 3.2.1 XLNT Command Syntax

XLNT, like any language, has its own vocabulary and usage rules. XLNT consists of commands, parameters and qualifiers, which are created in a way that XLNT can understand. An XLNT command is made up of the following information and in the format shown below:

[**\$**] [**label:**] **command** [/qualifier[=**value**]...] [**parameter**[/qualifier...]]

Items in brackets [ ] are optional and might not be required by a specific command. Each item above (from left to right) is explained more fully below.

- **XLNT prompt.** The dollar sign (\$) is the default XLNT prompt. When you work interactively with XLNT, XLNT displays the prompt when it is ready to accept a command. In a command procedure, each command should be preceded with a dollar sign at the beginning of each line.
- **Label.** A label identifies a line in a command procedure. Labels are never specified (and aren't allowed) in an interactive session.
- **XLNT command.** An XLNT command specifies the name of the command. A command is also referred to as a **verb**. You must enter as many characters of the command as is necessary to unambiguously identify the entered command.
- **Qualifier.** A qualifier modifies the action taken by a command. Qualifiers are always preceded by a slash (/). A qualifier may be placed in two possible positions within an XLNT command. In the **command position**, right after the command verb, a qualifier modifies the entire command. In the **parameter position**, a qualifier modifies the action or interpretation of an individual parameter. Some qualifiers can accept values. When a qualifier can accept more than one value, a set of parentheses is used to enclose the set of values. For example, /LIST=(1,2,3,4) indicates a qualifier named LIST which contains four separate values. The types of values a qualifier may accept depend on the individual qualifier. In addition, a qualifier may be negated by placing the word NO in front of the actual qualifier. For example, /NOLIST means that a list is not desired. Some qualifiers may prevent the use of negation.
- **Parameter.** A parameter specifies what the command acts upon. You must position parameters in a certain order based on the command itself. Some parameters have specific types of values (for example, file specifications, date/time, etc). Multiple parameters are always separated by at least a single space. Some commands support multiple values for a single parameter. A **parameter value list** is noted by each value separated by a single comma. For example,

```
$ COPY A.A,B.B,C.C C:\TEMP\*.*
```

In this COPY command, two parameters are required. The first parameter is the input specification and the second parameter is the output specification. The first parameter supports a parameter value list, so files A.A, B.B and C.C are input parameter values for this COPY command. Note that each parameter value is separated by a comma. The output parameter is separated by at least one space to denote the separate parameter. Some parameters, like qualifiers, allow the use of negation. Unlike qualifiers, some parameters may be required. In the COPY example above, both input and output parameters are required. If a command is entered with missing required parameters, XLNT will prompt you (interactive mode only).

### 3.2.2 Entering an XLNT Command

Use the following rules for entering an XLNT command:

- Use any combination of uppercase and lowercase characters. XLNT will convert all data entered on the command line to uppercase unless the data is enclosed in quotation marks (" ").
- Separate the command verb from the first parameter with at least one blank space or tab.
- Separate each additional parameter from the previous parameter or qualifier with at least one blank space or tab.

- Begin each qualifier with a forward slash (/). No blank spaces or tab characters can exist between the slash and the qualifier name.
- The maximum length of a command line is limited to 4096 bytes (or that value specified in the XLNT Registry).
- If a parameter or qualifier value includes a blank space or tab, enclose the parameter or qualifier value in quotation marks.
- A parameter can be omitted through the use of two consecutive quotation marks ("").

If you enter a command that is longer than one line, you can continue the command onto the next line, by inserting a hyphen (-) at the end of the line. In interactive mode, XLNT will display an underscore followed by the XLNT prompt (\_\$). You can now enter the rest of the line. Please note that XLNT does not insert any "free" blank space characters when lines are continued.

### 3.2.3 Editing the XLNT Command Line

The XLNT Console Window provides built-in facilities for command line recall and edit. This section describes those facilities that are present for interactive mode use only. Any text editor may be used to create and/or edit a command procedure.

At the XLNT command level, you can use many individual keys and key sequences to change what you type.

To delete the most recently entered data in a command line, use the BACKSPACE or  $\leftarrow$  key (**not** the left arrow key) to erase characters to the left of the current cursor position. To move the cursor to a new position within the command line, you may press the left or right arrow keys ( $\leftarrow$   $\rightarrow$ ), the HOME key (to move to the leftmost position) or the END key (to move to the rightmost position). To delete a character at the current cursor position, press the DELETE key). To insert data within a command line, press the INSERT key (the cursor should change in appearance). To overstrike existing data within a command line with new data, press the INSERT key to toggle between Insert and Overstrike mode. To erase an entire command line, press the ESC key (or enter Control/U).

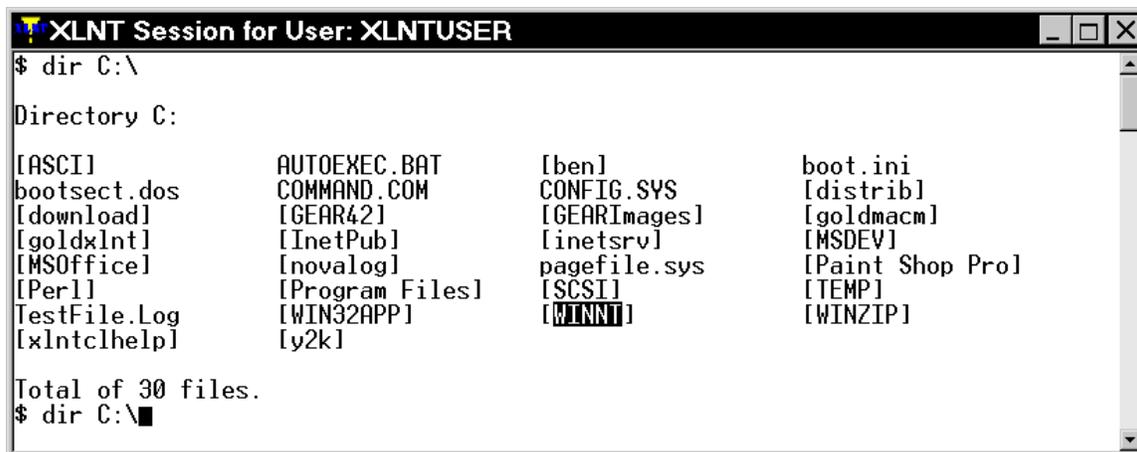
To recall previous commands, you have a choice of using the Up and Down Arrow keys ( $\uparrow$   $\downarrow$ ) to visually scroll through previous commands. Once you have selected the previous command line to execute, you may edit the command as described above or simply execute the command by pressing the ENTER key. To obtain a list of all previous commands still available for recall, use the RECALL command. Typing RECALL by itself will cause XLNT to display a numbered list of all previously entered commands from the most recent to the oldest. To select a specific command you may enter the number of the previous command (i.e. RECALL 5) or that portion of the previous command that uniquely identifies the command (i.e. RECALL DIR).

### 3.2.4 XLNT Mouse Handling

A local XLNT interactive session also supports use of your Mouse.

To position the cursor, click MB1 (Mouse Button 1). To select a word (or that portion between two spaces or between a margin and a space) double-click MB1. To select a group of characters, words or lines, press MB1 and *drag* the Mouse to highlight your selection. With your data selected, you may click MB2 once to paste your selection into the active console command line. If you double-click MB2, XLNT will execute your selection (analogous to RUNning a file). Information selected is placed in the Clipboard and may be used with other Windows applications.

Note: MB1 is either the leftmost or the rightmost mouse button depending on how you've configured your mouse with WinNT (either as a left handed or right handed mouse). MB2 is opposite the MB1 button.



The image shows a terminal window titled "XLNT Session for User: XLNTUSER". The prompt is "\$ dir C:\". The output is a directory listing for C:\, showing 30 files. The files are listed in four columns:

Column 1	Column 2	Column 3	Column 4
[ASCI]	AUTOEXEC.BAT	[ben]	boot.ini
bootsect.dos	COMMAND.COM	CONFIG.SYS	[distrib]
[download]	[GEAR42]	[GEARImages]	[goldmacm]
[goldxIntl]	[InetPub]	[inetsrv]	[MSDEV]
[MSOffice]	[noalog]	pagefile.sys	[Paint Shop Pro]
[Perl]	[Program Files]	[SCSI]	[TEMP]
TestFile.Log	[WIN32APP]	[WINNT]	[WINZIP]
[xIntclhelp]	[y2k]		

Total of 30 files.  
\$ dir C:\

Figure 3. XLNT Cut & Paste

## 3.3 Symbols

XLNT supports the use of symbols or variables. Symbols can be used for a variety of reasons, from passing information to procedures, performing numeric or string operations, passing data to functions or lexicals or to save the error status after a program has been run. Symbols derive real power in XLNT through their substitution facility. Symbols can be declared explicitly (**Declared Symbols**) or used implicitly. A symbol, like a conventional program's variable, has a data type. The values they represent can be used for data or commands.

The following list describes the uses of symbols:

- As a synonym for a command line. This usage is a string data type
- As a variable in a command procedure.
- As a value to be passed to a declared function or lexical.
- Used to represent both file and data records in the OPEN, CLOSE, READ, WRITE and INQUIRE commands
- As input arguments to a command procedure.

The following are built-in symbols:

<b>BATCH\$RESTART</b>	String symbol containing the last value set by the SET RESTART_VALUE command. (Only valid when used with ActiveBatch and \$RESTART is true).
<b>\$DOSSTATUS</b>	Integer symbol for current DOS/CMD program status (DOS/CMD commands only)
<b>\$ENTRY</b>	Integer symbol representing last valid batch job entry id submitted (BQMS only)
<b>\$FALSE</b>	Integer symbol that represents an XLNT and ActiveX Scripting "False" value
<b>\$JOBID</b>	Integer symbol for current ActiveBatch job number
<b>\$LOCAL_MACHINE</b>	String symbol containing local computer name
<b>\$LOGIN</b>	String symbol indicating the XLNT Installation directory
<b>\$NULL</b>	String symbol containing null byte
<b>\$PID</b>	Process Identification of current job
<b>\$QUEUE</b>	String symbol representing the actual BQMS queue the last valid batch job was submitted to.
<b>\$QUEUE_MANAGER</b>	String symbol representing the actual machine name on which the ActiveBatch Job Scheduler accepted the last submitted job.
<b>\$REMOTE</b>	Boolean symbol indicating remote XLNT login
<b>\$RESTART</b>	Boolean symbol indicating whether a Batch job has been restarted
<b>\$SEVERITY</b>	Integer symbol for Severity indication
<b>\$STATUS</b>	integer symbol holding last exit status of a program, command, statement or procedure. \$STATUS is always set when an error is encountered. \$STATUS is not set when the following statements are successfully executed: GOTO, GOSUB and the Assignment statement.
<b>\$STDERR</b>	Handle symbol for STD_ERROR
<b>\$STDIN</b>	Handle symbol for STD_INPUT
<b>\$STDOUT</b>	Handle symbol for STD_OUTPUT
<b>\$TRUE</b>	Integer symbol that represents an XLNT and ActiveX Scripting "True" value
<b>\$USERNAME</b>	String symbol for the authorized user
<b>\$VERSION</b>	String symbol representing the version of XLNT
<b>WSCRIPT</b>	Object symbol for WSH (when applicable)
<b>\$XLNT_LOGIN</b>	String symbol indicating the user's XLNT default Login directory (as set by SET PREFERENCE)

Symbols can also be abbreviated. An abbreviated symbol contains the wildcard character (\*) as part of the symbol name. For example:

```
$ DEL*ETE := DELETE/LOG
```

This assignment statement creates the abbreviated symbol DEL. The wildcard indicates the shortest length string that must be entered to correctly identify this symbol (DEL).

### 3.3.1 Defining Foreign Commands

Typically, symbols can be used as shorthand for the various XLNT commands. For example, if you always wanted COPY to prompt you prior to copying a file you could use:

```
$ COPY := = COPY/CONFIRM
```

This would create the global symbol named COPY and provide you with a way of reducing the number of qualifiers entered on a command line.

XLNT also offers a method of creating symbols that define non-XLNT commands. We refer to these types of symbols and commands as *foreign commands* since the command is not native to XLNT. Creation of a foreign command is quite simple and very powerful. The formats for defining a foreign command are as follows:

```
$ symbol-name := [=] $file-specification
```

```
$ symbol-name = [=] "$file-specification"
```

where *symbol-name* is any acceptable XLNT symbol and *file-specification* is any acceptable Windows file. Note that when the dollar sign (\$) precedes a file specification at the beginning of a symbol definition, without any space between the dollar sign and the file specification, the request to *run* the image is implied. For example, if you wanted to create a symbol named WORD that ran the Microsoft Word program that command would be:

```
$ WORD := = $C:\MSOFFICE\WINWORD\WINWORD.EXE
```

If you then typed:

```
$ WORD
```

the MS Word program would start. You can also enter input parameters to foreign commands. For example,

```
$ WORD C:\WORDOC\SAMPLE.DOC
```

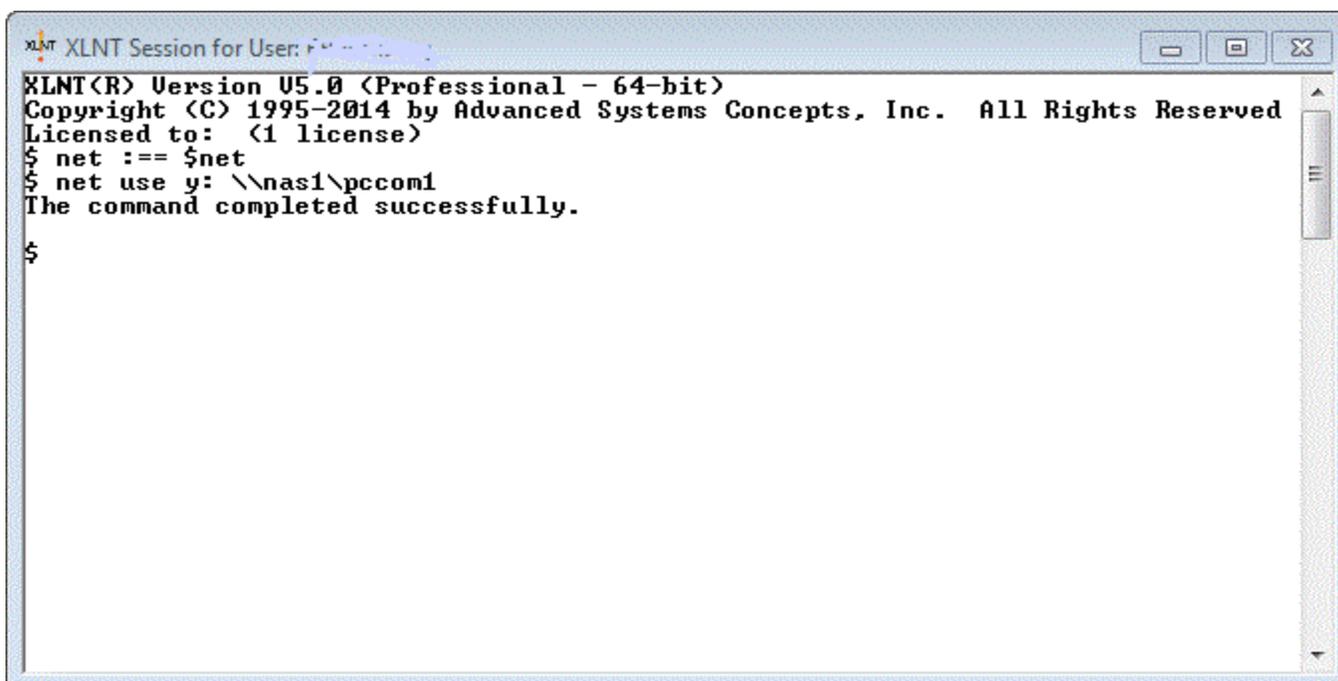
would startup MS Word and then open the file SAMPLE.DOC as an input file to that program. So if you ever tire of having to enter DOS NET or any other foreign command just:

```
$ NET := = $NET
```

```
$ NET USE Z: \\TEST\D$
```

XLNT does support implicit use of the PATH statement (which is how we find the NET.EXE command in case you were wondering).

XLNT also supports automatic foreign commands (if enabled, see the SET PREFERENCES command for more information). If a command is entered that is not a valid XLNT command, XLNT will search the current PATH looking for a file named "command.EXE". If that file is found, XLNT will invoke the program and pass it any parameters similar to that described above. If that file is not found, XLNT will search again looking for a file named "command.XCP". If that file is not found, XLNT will display the "invalid command" error.

A screenshot of an XLNT terminal window. The title bar reads "XLNT Session for User: F...". The terminal text is as follows:

```
XLNT(R) Version U5.0 (Professional - 64-bit)
Copyright (C) 1995-2014 by Advanced Systems Concepts, Inc. All Rights Reserved
Licensed to: (1 license)
$ net ::= $net
$ net use y: \\nas1\pccom1
The command completed successfully.
$
```

Figure 4. NET USE Foreign Command

### 3.3.2 Display and Deleting Symbols

The SHOW SYMBOL command displays the values of symbols. To display the value of a particular symbol, enter the SHOW SYMBOL command followed by the name of the symbol. For example, SHOW SYMBOL DEL would display the abbreviated symbol described above. To display only Global symbols the qualifier /GLOBAL would be used. To display only Local symbols the qualifier /LOCAL would be used. To display all symbols, the /ALL qualifier (or an asterisk) would be specified. For example,

**\$ show symbol \***

```
$JOBID == 0      Hex = 00000000
$LOGIN == "C: \ASCI \XLNT"
$SEVERITY == 3   Hex = 00000003
$USERNAME == "JDoe"
$REMOTE == 0     Hex = 00000000
$PID == 177     Hex = 000000B1
$STDERR == 11   Hex = 0000000B
NOTEPAD == "RUN NOTEPAD/ARG="
$STDIN == 3     Hex = 00000003
$LOCAL_MACHINE == "Machine"
$STDOUT == 7    Hex = 00000007
$STATUS == -805437395   Hex = CFFE002D
$RESTART == "FALSE"
```

The command SHOW SYMBOL \* displays all symbols.

**\$ a=1**

**\$ show symbol \*/local**

**A = 1 Hex = 00000001**

The command SHOW SYMBOL \*/LOCAL displays all local symbols.

To delete a symbol, use the DELETE/SYMBOL command. To delete a global symbol specify the /GLOBAL qualifier. By default, only local symbols are deleted.

### 3.3.3 Declared Symbols

When a symbol is declared the datatype is associated with the symbol and cannot be changed. Declared symbols can have the following datatypes: Signed byte/word/longword/64-bit, Unsigned byte/word/longword/64-bit, Fixed/Dynamic Strings, Handles and Objects. A symbol can also be associated with a Structure. The major advantages gained in declaring symbols (rather than using them implicitly) are: more datatypes to choose from and better syntax and run-time checking of a symbol's usage. Once a symbol is declared the data associated can be a string, binary integer (byte, word, longword, 64-bit), label (special type of string), handle, object and structure. Both string and integer symbols can be used to implement boolean values. Full or partial XLNT commands can be represented by a symbol. The following list describes the characteristic/components of a symbol:

**Name** - The symbol name is a string from 1 to 64 characters in length. Allowable characters are "a" through "z", "A" through "Z", "0" through "9", underscore (\_) and the dollar sign (\$). The first character must be an upper or lower case letter or the underscore. Note: Symbols beginning with a dollar sign are reserved for ASCII use (XLNT built-in symbols begin with a dollar sign, i.e. \$STATUS). A symbol name cannot conflict with a structure name.

**Datatype** - The datatype of a symbol may be: signed/unsigned byte/word/longword/64-bit, fixed/dynamic string, object, handle and structure. Please see the "List of Supported Datatypes" for a list of supported datatypes. The following values may be specified.

**String** - Any combination of displayable ASCII characters. The string may be of length 0 to the hardware/software limitations of the underlying Operating System.

**Binary** - A numeric value whose range is determined by the type of binary datatype chosen and whether the datatype is signed or unsigned. An implicit numeric symbol is always set as an integer (signed longword). This integer value may be Decimal or Hexadecimal. A hexadecimal value is specified by preceding the integer with "%x" (%d is decimal radix which is the default).

**Label** - A string indicating a label within a command procedure. A label is defined by entering the string followed by a colon (e.g. READ\_LOOP:).

**Boolean** - When a boolean test is performed on an integer, 0 equates to FALSE and any non-zero value equates to TRUE. When a string is used, any string with a first character of 'T', 't', 'Y', 'y' is equated to TRUE.

**Level** - Indicates the execution level at which the symbol was created. The level is used to control the scope of accessibility to the symbol. A **LOCAL** symbol is assigned the current level and can be accessed by this level. A **GLOBAL** symbol can be accessed at any level, regardless of its creation level.

To declare an explicit symbol, use the DECLARE SYMBOL command. The DECLARE SYMBOL command has the following format:

```
DECLARE SYMBOL datatype symbol-name [options,...]
```

**datatype** refers to one taken from the list of [Supported Datatypes](#) and represents the symbol's datatype. A [structure](#) name may also be specified indicating that this symbol is an instance of the structure.

**symbol-name** represents the symbol's unique name. A symbol name is limited to sixty four (64) characters.

**options,...** refers to the set of options that may be associated with the symbol. For example, GLOBAL and INITIALIZE are symbol declaration options.

```
XLNT Session for User: XLNTUSER
$ test_symbol = 125
$ new_symbol = test_symbol + 10
$ show symbol test_symbol
TEST_SYMBOL = 125    Hex = 0000007D
$ show symbol new_symbol
NEW_SYMBOL = 135    Hex = 00000087
$
$ test_symbol = "This is a string"
$ test_symbol == "This is a global string"
$ show symbol test_symbol
TEST_SYMBOL = "This is a string"
$ show symbol/global test_symbol
TEST_SYMBOL == "This is a global string"
$
$ declare symbol long x_size
$ x_size = 300
$ show symbol x_size
X_SIZE = 300    Hex = 0000012C
$ x_size = "A new value"
%XLNT-E-INCOMPATSYMTYPES, incompatible symbol types specified
$
```

Figure 5. DECLARE SYMBOL

### 3.3.4 Implicit Symbols

An implicit symbol is simply a symbol which is used (in an Assignment statement) and not previously declared. The symbol's datatype is therefore taken as a result of the value in which it is associated. This type of script writing is referred to as *typeless*. Typeless scripting allows the scriptwriter to avoid declaring symbols and instead to defer such checking until the script actually executes. For example,

```
$ A = "123"
```

indicates that symbol A is a string symbol whose value is "123". Another example:

```
$ B = 52
```

indicates that symbol B is an integer symbol whose value is 52. Finally another example:

```
$ C = Wsscript.CreateObjectName ("Excel.Application")
```

indicates that symbol C is an object symbol.

### 3.3.5 Labels

A label is a special type of local symbol which allows a programmer to transfer control within a command procedure. To define a label, simply enter the symbol name followed by a colon. For example:

#### **\$open\_file:**

The label *open\_file* may be used as the target of a GOTO, ON or error processing qualifier. For improved readability, we suggest that you begin the label immediately after the XLNT dollar sign (there is no requirement to do so).

Labels within an IF/WHILE/UNTIL block may only be accessed by a GOTO or GOSUB within the same block. In other words, the label's scope is limited to the block in which it has been defined. Typically, labels are defined outside of a block and may therefore be accessed anywhere within a command level. If a GOTO or other transfer of control is effected to cause the transfer to continue execution outside of a block, the block context is terminated in its entirety.

### 3.3.6 Supported Datatypes

Symbols and functions maintain a datatype which describes how the value of the symbol or function is to be interpreted. The following list describes all the currently supported datatypes for both symbols and functions (except as noted). Please note that slashes ("/") delimit alternative forms of the same datatype. For example, Integer, Long and SLong all describe the same datatype (which is a signed longword).

#### **Integer/Long/SLong**

This datatype represents a 32-bit signed binary value. The range of this value is from  $-2^{31}$  to  $+2^{31}-1$ . The specification of Integer causes XLNT to use the base integer datatype of the machine it is being executed on. For a 32-bit system, this would be 32-bits.

#### **INT64**

This datatype represents a 64-bit signed binary value. The range of this value is from  $-2^{62}$  to  $+2^{62}$ .

#### **UINT64**

This datatype represents a 64-bit unsigned binary value. The range of this value is from 0 to  $2^{63}$ .

#### **ULong/Dword**

This datatype represents a 32-bit unsigned binary value. The range of this value is from 0 to  $2^{32}-1$ .

#### **Byte/SByte**

This datatype represents an 8-bit signed binary value. The range of this value is from  $-2^7$  to  $+2^7-1$ .

#### **UByte**

This datatype represents an 8-bit unsigned binary value. The range of this value is from 0 to  $2^7$ .

#### **Word/SWord**

This datatype represents an 16-bit signed binary value. The range of this value is from  $-2^{15}$  to  $+2^{15}-1$ .

#### **UWord**

This datatype represents an 16-bit unsigned binary value. The range of this value is from 0 to  $2^{16}-1$ .

#### **String**

This datatype represents a character string. Strings in XLNT may be further defined as Fixed or Dynamic. By default, when no further qualification is made, the string is considered dynamic.

#### **Handle**

This datatype represents a handle or operating system specific description of the viewport for accessing a device or performing other operating system specific functions. Under Windows 2000/NT and Windows Me/9x, the handle is a longword (or DWORD).

#### **Object**

This datatype represents an object. An object is acquired through either the ActiveX scripting environment or directly from the COM/DCOM services.

#### **Void (Functions Only)**

This datatype actually isn't a datatype at all. VOID is limited to function usage. When VOID is specified as a function datatype this indicates that no value will be returned by the function. When VOID is specified as an argument prototype, no datatype checking will be performed.

### 3.3.6.1 Arrays

An array is a list (or table) of variables of a related data type. The variables in an array have a common name. Each individual element in the array is accessed using an *index*. The number of indices that can be specified is referred to as the number of dimensions. A one dimensional array has a single index value. Each dimension is typically declared with a maximum number of elements. XLNT does support *no specific maximum* arrays as well as fixed maximum element arrays. The general format for an array element reference is:

```
array-symbol{symbol|integer|"string"}
```

The above notation means that when you specify an array element, your reference to the element can be via symbol substitution or an integer value (for a numeric indexed array) or a quoted string (for an associative indexed array).

XLNT supports two types of array indices: Numeric and Associative.

#### 3.3.6.1.1 Numeric Indexed Arrays

A numeric indexed array uses a positive number to access each specific element. The numeric index begins with zero and refers to the first element in the array. For example:

```
$ A{5} = 1
```

means that the sixth element in array A is to be accessed and the value one (1) stored. Note that a starting and ending brace "{ }" holds the index (or indices) reference. As a further syntax note, no embedded spaces can occur between the array symbol and the braces. To declare a one-dimensional array issue the following statement:

```
$ DECLARE SYMBOL LONG A OCCURS=10
```

This declares the symbol A as a longword array containing 10 elements (from 0 to 9). A two dimensional array element reference might look like this:

```
$ B{2,4} = 100
```

This means that element 2,4 in array B is assigned a value of 100. To declare B as a two dimensional array you would issue a statement like this:

```
$ DECLARE SYMBOL LONG B OCCURS=(10:10)
```

This declares the symbol B as a longword two-dimensional array containing 100 elements consisting of 10 in one plane and 10 in the other (which index values from 0 to 9 for each dimension).

XLNT also allows multiple array elements to be initialized in a single assignment statement. For example, assume the array WEEKDAYS:

```
$ WEEKDAYS = ("Mon","Tue","Wed","Fri","Sat","Sun")
```

This statement initializes the array WEEKDAYS beginning with numeric index zero (0) and continuing sequentially until all initializers in the list are exhausted. The syntax of the initializer statement for numeric indexed arrays is:

```
$ array-symbol = ([index:] value,...)
```

where *index* is a numeric positive number indicating the element position in the array that *value* should be stored. If *index* is omitted, then XLNT stores the value beginning at zero and ascending sequentially for each value in the list. The value itself can be a quoted string or an integer.

#### 3.3.6.1.2 Associative Indexed Arrays

An associative indexed array uses a string index to access a specific element. For example:

```
$ RECORD{"ken"} = 5
```

means that the element named "ken" in array RECORD is to be accessed and the value five (5) stored. For an associative array, the index is really a key into the array. The key must uniquely reference an element in the array, and the key itself cannot be null (for example, RECORD{""} is illegal). Associative Indexed Arrays are one-dimensional and typically consist of a variable number of elements (as opposed to a fixed number). The major benefit of an Associative Indexed

Arrays is that the array is actually structured using an AVL tree algorithm. This algorithm provides very fast access to many elements in the tree. The other benefit to structuring the array as a tree is that sequential access to the key(s) in the array are possible. For example:

```
$ ADDRESS{"irv"}="1 Sycamore Lane"
$ ADDRESS{"abe"}="214 Main Street"
$ ADDRESS{"don"}="659 Townsend Ave"
```

Assume these keys were established in the order you see them. While you can certainly access any of these elements directly, by just specifying the key index, you might want to access them sequentially. The following code fragment allows you to do just that:

```
$ FOR (,)
$   KEY=F$SEQARRAY(ADDRESS,context)
$   IF KEY .EQS. "" THEN LEAVE
$   WRITE $STDOUT "Address for 'key' is "ADDRESS{"'key'"} "
$ ENDFOR
$ EXIT
```

The lexical F\$SEQARRAY allows you to traverse an associative array sequentially in ascending key index order. Each invocation of F\$SEQARRAY passes back either the next sequential key index value in the named array or a null string is returned which indicates the end of the sequential scan.

Arrays are typically declared using the DECLARE SYMBOL statement although arrays can be used in a typeless manner. As with all symbols once a symbol has a datatype all values associated with the symbol must be compatible with that datatype.

### 3.3.6.2 Structures

A structure provides a mechanism for grouping related fields or symbols. Each symbol may in turn have a standard datatype or represent yet another structure. Under XLNT a structure is first declared and then used in the DECLARE SYMBOL command in the same manner as associating a symbol with a given datatype. Structures are defined through the STRUCTURE command and are formally ended with the ENDSTRUCTURE command. An example provides a good illustration.

```
STRUCTURE RECORD
    STRING NAME 20
    STRING ADDRESS1 30
    STRING ADDRESS2 30
    STRING CITY 20
    STRING STATE 2
    STRING ZIPCODE 5
ENDSTRUCTURE
```

The structure RECORD represents six fields. The datatype of each field can be any legal datatype or structure name. If the datatype is a string, the length of the string follows the field name. To associate a symbol with a given structure you use the DECLARE SYMBOL command. For example,

```
DECLARE SYMBOL RECORD INPUT
```

This command associates the structure RECORD with the symbol INPUT. To reference fields within a structure, the following syntax is used:

```
symbol-name | field-name
```

For example, input|name represents the field *name* for the symbol *input*. The vertical bar (|) is used to access a given field for a specific symbol. Please note that while a space exists in the definition above for clarity, no embedded spaces are allowed between symbol-name, the vertical bar, and the field-name (for example, INPUT|ZIPCODE).

The only option that may be specified for a structure is whether it should be local (default) or global. Such as:  
STRUCTURE RECORD GLOBAL.

To delete a structure, use the DELETE/SYMBOL command (using proper scope, local or global). If a symbol is currently using the structure reference, the structure may not be deleted.

### 3.3.6.3 Symbol Scope

Symbols may be defined as either global or local. A local symbol defined in an outer mode procedure is accessible to all inner command procedure levels. A global symbol is accessible to all command procedures regardless of level.

To define a global symbol using the **Declared Symbols** facility, specify GLOBAL as an option (the default is local). To define a global symbol implicitly, specify double equal signs (==) as part of the initial symbol assignment.

For example:

```
$ A = 1
```

implicitly defines the local integer symbol A with a value of 1.

```
$ A == 1
```

would define this symbol globally.

```
$ DECLARE SYMBOL INTEGER A GLOBAL
```

would define this symbol globally as a declared symbol.

### 3.3.6.4 Symbol Substitution

When a symbol is used (passed to a program or command as an operand in an expression) the value is substituted for the symbol. In certain contexts, XLNT expects a symbol name while in others a quoted string or value is expected. The following list shows the contexts in which implicit substitution takes place. All other contexts require explicit substitution for correct interpretation.

Implicit Substitution Rules:

- When the symbol is the first item on the command line
- On the right side of an assignment operator (=, ==)
- As an argument to a lexical function
- When an expression may be entered

**Examples:**

```
$ TEST = "RUN PROGRAM"
```

```
$ TEST
```

```
$ B = 1
```

```
$ C = 1
```

```
$ A = B + C
```

To force substitution of a symbol that is not in one of the positions listed, enclose the symbol with apostrophes ('), as follows:

```
$ PROG = "D:\USER\MYPROG.EXE"
```

```
$ RUN 'PROG
```

The symbol PROG requires explicit (forced) symbol substitution for proper interpretation (otherwise the string would be RUN PROG which is an acceptable if not desired command).

To force substitution of a symbol within a quoted character string, precede that symbol with two apostrophes (') and follow it with a single apostrophe as follows:

```
$ DAY = "THURSDAY"
```

```
$ STR1 = "TODAY IS ' DAY' "
```

```
$ SHOW SYMBOL STR1
```

```
STR1 = "TODAY IS THURSDAY"
```

In the example above the explicit substitution of DAY uses two quotation marks in succession (no spaces should be present, although the example has a space to better denote the individual single apostrophe marks).

When processing a command line, XLNT replaces symbols using forced substitution and then automatic substitution, in that order.

To include a double quotation mark (") within a quoted string specify two double quotation marks for each single double quotation mark you want interpreted as data. For example:

```
$ DAY = " " " THURSDAY " " "
```

yields a value of "THURSDAY" (which the quotation marks as a part of the value and NOT to denote that DAY is a string symbol). Again, no spaces are allowed between the two double quotation marks (although this has been done above for better readability).

### 3.3.7 Object Reference

XLNT provides the ability to access and invoke COM objects. Typically most XLNT scripts gain access to COM objects, their methods and properties through Microsoft's ActiveX facilities. One of the ActiveX components is the Windows Scripting Host (WSH) which allows scripts to gain access to various methods and properties without requiring IIS use. While users interested in accessing objects should read the section on *XLNT and COM/ActiveX Usage* the syntax of symbol specification is also provided at this point for completeness of our discussion of symbols, their types and usage.

To access an object's methods and/or properties, the object symbol name followed by a period and then the object's method, arguments or property names are specified. For example:

WSSCRIPT.PATHNAME is an object reference. WSSCRIPT is an object symbol and PATHNAME is a method. Some methods require arguments.

Object symbols can be declared or used implicitly. To declare an object simply use the DECLARE SYMBOL statement:

```
$ DECLARE SYMBOL OBJECT WNET
```

The preceding statement declares the symbol WNET as an object. To implicitly associate the object datatype with a symbol you simply use it:

```
$ wnet = WScript.CreateObject ("WScript.Network")
```

This assignment statement associates the object value in the object symbol "wnet".

## 3.4 Operators

XLNT provides a set of operators to be used when performing string, integer, and logical operations.

### 3.4.1 String Operators

Operator	Function
+	String Concatenation
-	String Reduction
.EQS.	Equal to
.GES.	Greater than or equal to
.GTS.	Greater than
.LES.	Less than or equal to
.LTS.	Less than
.NES.	Not equal to

#### Examples:

```
$ if title .eqs. "XLNT" then write $stdout "Good"
$ if title .nes. record
$ then
$     write $stdout record
$     a=a+1
$ endif
$ a = "abc"
$ b = "def"
$ c = a + b           ! c now equals abcdef
$ c = c - "def"      ! c now reduced to abc
```

**Concatenation** refers to an operation where two character strings are joined as one. **Reduction** refers to an operation where the second character string is removed from the first character string. If the second character string occurs more than once in the first character string, only the first occurrence is removed. **String comparison** refers to an operation where two strings are compared character by character. Different length strings are not considered equal.

XLNT supports a capability known as **substring replacement** where a portion of a string symbol can be replaced as part of an assignment statement. The syntax for the substring replacement is:

**symbol-name[offset,size] := replacement-string**

(for global symbol usage replace the := with ==). The *offset* argument refers to an integer that indicates the position of the replacement string relative to the first character in the original string (symbol-name). An offset of 0 refers to the first character. The *size* argument indicates the length of the replacement string. The rules for substring replacement are:

- The square brackets are required notation. No spaces are allowed between the symbol-name and the left bracket.
- Integer values must be positive numbers.
- The replacement string must be a character string
- The symbol name you specify can be undefined initially. The assignment statement creates the symbol name and if necessary, provides the leading or trailing spaces in the symbol value.

Numeric symbols can perform a **numeric overlay**. The numeric overlay operation allows a binary (bit-level) overlay of the current symbol value by using a special form of the assignment statement:

**symbol-name[bit-position,size] = replacement-expression**

(for global symbols, change = to ==). The *bit position* argument is an integer that indicates the location relative to bit 0 at which the overlay is to occur. The *size* argument is an integer that indicates the number of bits to be overlaid. To use numeric overlays, observe these rules:

- The square brackets are required notation. No spaces are allowed between the symbol name and the left bracket.
- Literal values are assumed to be decimal (use the radix operator for other notations).
- The maximum size is 32 bits.
- The replacement-expression must be a numeric expression.

## 3.5 Logical Values and Expressions

Both string and integer symbols can be interpreted logically or contain a logical value.

**True.** A number has a logical value of true if it is non-zero. A character string has a logical value of true if the first character is an uppercase or lowercase T or Y.

**False.** A number has a logical value of false if it is zero. A character string has a logical value of false if the first character is not an uppercase or lowercase T or Y.

One exception to this rule is the special \$STATUS symbol. When \$STATUS is evaluated with an IF statement, XLNT interprets the exit status to determine whether the program was successful or not. A success code is therefore evaluated as true and a failure code is evaluated as false. This prevents the XLNT user from having to interpret the exit status.

In a **numeric expression**, the values involved must be numbers, string numbers (i.e. "16") or numeric symbols. In a **logical expression** the values involved must also be numeric and the result of the expression is numeric as well. If a character string is specified in a numeric or logical expression, the string is converted to an integer value (if the string is not numeric then its value becomes zero). An expression can contain any number of operations and comparisons. The order of operations and how and when they are evaluated are as follows (from highest precedence to lowest): Unary plus (+) and minus (-), Multiplication (\*) and division (/), Concatenation and reduction, all numeric and character comparisons, logical .NOT., logical .AND., and logical .OR. operations. To override the normal order of precedence, simply place parentheses around the operations you would like to be evaluated first.

### 3.5.1 Numeric Operators

Operator	Function
+	Addition
-	Subtraction
+	Unary plus (indicates a positive number)
-	Unary minus (indicates a negative number)
*	Multiplication
/	Division
.EQ.	Equal to
.GE.	Greater than or equal to
.GT.	Greater than
.LE.	Less than or equal to
.LT.	Less than
.NE.	Not equal to

### 3.5.2 Logical Operators

Operator	Function
.AND.	Logically AND's two logical values
.NOT.	Logically NOT's two logical values
.OR.	Logically OR's two logical values
.COM.	Logical One's Complement

Logical Operators can be used to change bits within a number or to request Logical Expression processing between two or more expressions. For example:

\$ A = 1 .AND. 1 yields a value of 1 (or true).

\$ A = 1 .AND. 0 yields a value of 0 (or false).

*The .AND. operator combines two logical values.*

\$ A = .NOT. 1 yields a value of 0 (or false).

*The .NOT. operator reserves a boolean expression.*

\$ A = 1 .OR. 1 yields a value of 1 (or true).

\$ A = 1 .OR. 0 yields a value of 1 (or true).

\$ A = 0 .OR. 0 yields a value of 0 (or false).

*The .OR. operator combines two logical values.*

\$ A = .COM. 1 yields a value of -2 (or true).

*The .COM. operator reverse the bits of an expression.*

## 3.6 Functions

A *function* is a common routine that performs a requested operation and usually returns a result.

XLNT supports functions by declaring a function prototype, loading the DLL the function resides in, and then invoking the function itself. You may declare hundreds (or thousands) of functions and never use them, however, any function which is to be executed *must* reside in a DLL and the DLL must be loaded.

To declare a function you use the DECLARE FUNCTION statement. This statement allows you to build a simple function prototype in which the argument and other characteristics of the function are described. The DECLARE FUNCTION statement has the following format:

DECLARE FUNCTION *datatype name library [arguments,...] [options]*

**datatype** is one taken from a list of [Supported Datatypes](#) and represents the datatype of the value returned by the function. If the function is not to return a value then VOID should be specified.

**name** represents the function's name and must match the exported name in the DLL where the function resides. This name should not conflict with other symbol names. The function name may be quoted if case sensitivity is needed.

**library** represents the DLL filename on disk which is intended to be loaded prior to function usage. For example, if the complete path name of the DLL is "c:\winnt35\win32api.dll" then this parameter would be specified as "win32api" (in other words, omit the path portion and type of the DLL's file specification). The library name may be quoted if case sensitivity is needed.

**arguments,...** represents the datatype of zero or more arguments to be passed to the named function itself. The datatype(s) may be taken from the list of [Supported Datatypes](#). For example,

DECLARE FUNCTION integer "ShowWindow" "win32api" handle, integer  
declares the function "ShowWindow" for possible use (illustration purposes only). The function returns an integer value back to the procedure. The function requires two arguments to be passed; a handle and an integer. To declare a function with no arguments, you may omit the *arguments* parameter entirely. To skip or omit an argument you may specify a null parameter (which is two double quotes with no intervening characters, i.e. ""). The ampersand (&) operator may be placed in front of a numeric datatype to indicate that the argument should be passed by reference (by default, XLNT passes numeric and structure arguments by value and string arguments by reference). The ampersand operator may also be specified when the function itself is actually invoked. If VOID is specified no datatype checking is performed.

**options,...** represents any handling options to be associated with this function. At present, the only options which may be specified are GLOBAL, STDCALL and CDECL. By default, functions are declared locally within the command procedure that defined them. To increase the scope of declared functions so all possible command procedures can maintain visibility to the functions, you should specify the GLOBAL option. By default, functions are declared as CDECL compliant, which means that XLNT will call your specified function through the CDECL calling standard. C functions are, by default, CDECL compliant. STDCALL may be specified for those functions which require that mechanism for function invocation. Please note that all Win32 API functions require that STDCALL be specified.

A *lexical* is a built-in XLNT function that also performs an operation and always returns a result. The general format of a lexical is:

F\$function-name([args,...])

The [Lexical Commands](#) section provides a list of all supported lexicals. You may invoke a lexical function in any context in which you normally use an expression.

---

```
declare function long "SetConsoleTitleA" Kernel32 string stdcall
```

Figure 6. DECLARE FUNCTION statement

## 3.7 Files and Directories

### 3.7.1 UNC Specifications

XLNT supports the Universal Naming Convention (UNC) and Long File Names (LNF) that are provided by Windows 2000/NT/Me/9x. A UNC File Specification has the following format:

***\\machine\share\directory\...filename***

Please note that the ellipsis between directory and filename indicates that additional sub-directories are possible.

All XLNT commands (except EDIT, DOS and DOS related commands) support the UNC method. This means that you aren't limited to the older syntax of:

***drive-letter:\directoryfilename.type***

where both local and network usage require a drive letter.

By default, XLNT users are placed into their "login" directory as indicated through the SET PREFERENCE command (the default is \ASCII\XLNT). To change your default directory, use the SET DEFAULT command. For example, to change to C:\Program Files you would use the following command:

```
$ SET DEFAULT "C:\Program Files"
```

The quotation marks are significant since the directory contains an embedded blank space.

XLNT supports the use of wildcards when searching or specifying directories and/or files.

The filename (and type) portion can include the following wildcard characters:

An asterisk (\*) can be used in the directory name, filename or type. It matches on any number of characters. An output file specification cannot contain both text and wildcard characters inside the directory, filename or type fields.

A question mark (?) can be used in the directory name, filename or type. It matches on one character. Output file specifications cannot contain the '?' character.

The ellipsis (...) is used in Directory and Copy commands to indicate a subdirectory search. The '...' must appear between two '\' characters.

The following are valid examples:

```
$ copy \\test\c$\test.tst \\guest\d$\*. *
$ copy c:\xInt\myprog.exe d:\xInt\*. *
$ delete *.?cp
$ dir d:\xInt\...\*.cpp
$ copy c:\temp\...\*. * c:\temp1\...\*. *
$ delete c:\temp\...\*. *
```

The following are invalid examples:

```
$ rename c:\xInt\myprog.exe his*.exe
$ copy *.xcp *.?
$ dir c:\xInt...
```

In the sections that describe commands, statements and lexicals, **file-name** denotes the complete use of UNC and/or FAT type file specifications (unless otherwise noted, for example, EDIT and DOS). Those commands that denote **directory-name** also allow UNC and/or FAT type directory specifications (but, of course, no filename portion is allowed).

## 3.7.2 FTP URL Specifications

Several XLNT file commands also support the use of FTP URL specifications in addition to UNC specifications. *You must enclose a URL specification within a quoted string.* The following format depicts a valid FTP URL specification for XLNT purposes:

```
ftp://[user [:password] @ ] host [: port] / path
```

The user, password and port information is optional. If omitted, XLNT will assume an anonymous login and standard FTP port. The *path* portion can include full directory and filename, filetype wildcarding as supported by XLNT's UNC specification. When reading the XLNT commands, look for the keyword **ftpurl** as a notation for those file command arguments which support this syntax. For example:

```
ftp://user:password@ftp.advsyscon.com:5051/c:/.../*.*
```

This example shows all the possible FTP URL syntax. A *user* with a password of *password* is accessing machine *ftp.advsyscon.com* using port 5051. The file specification is *c:/.../\*.\** as might be used in a DIRECTORY or COPY command.

Hint: Use the XC command (Professional Edition) to encode and encrypt any embedded passwords within a command procedure.

## 3.8 Date and Time Format

Certain commands and qualifiers accept date and time values. You can specify these values in one of the following formats:

### 3.8.1 Absolute Time

An absolute time is a specific date or time of day. The format for an absolute time is as follows:

`"[dd-mmm-yyyy][:hh:mm:ss.mms]"`

The fields are:

dd	Day of month (range from 1 to 31, depending on month)
mmm	Month (JAN...DEC)
yyyy	Year
hh	Hour (range from 0 to 23)
mm	Minute (range from 0 to 59)
ss	Second (range from 0 to 59)
mms	Milliseconds (range from 0 to 999)

The following rules apply to an absolute time:

- You can truncate the date or the time from the right.
- The date must contain at least one hyphen.
- You can omit any of the fields (other than month if a date is specified) as long as you include the punctuation marks that separate the fields.
- A truncated or omitted date field defaults to the corresponding fields for the current date.
- A truncated or omitted time field defaults to zero.

Example:

1-DEC-1997	Midnight, December 1, 1997.
15-DEC-1997:13:00	1:00 PM, December 15, 1997
20-DEC::30	12:30 AM, December 20, 1997
16:30	4:30 PM today

### 3.8.2 Delta Time

A delta time is an offset from the current date and time to a future time. The format of a delta time is:

“+[dddd-][hh:mm:ss:mms]”

The fields are as follows:

dddd	Number of days (range from 0 to 9999)
hh	Hours (range from 0 to 23)
mm	Minutes (range from 0 to 59)
ss	Seconds (range from 0 to 59)
mms	Milliseconds (range from 0 to 999)

The following rules apply to a delta time:

- You must specify a plus (+) sign.
- You can truncate a delta time on the right.
- If you specify the number of days, include a hyphen
- You can omit fields within the time as long as you include the punctuation that separates the fields.
- If you omit the time field, the default is zero.

Examples:

“+5-“	5 days from now
“+3”	3 hours from now
“+:30”	30 minutes from now

### 3.8.3 Combination Time

To combine absolute and delta times, specify an absolute time followed by a delta time.

“[absolute time][+delta time]”

“[absolute time][-delta time]”

The following rules apply:

- Precede the delta time value by a plus sign for future time or a minus sign for past time.
- Enclose the entire specification in quotation marks if a plus or minus sign precedes the delta time value.
- Omit the absolute time value if you want to offset the delta time from the current date and time.
- Remember that the rules for absolute and delta time values still apply.

You can also specify **TODAY**, **TOMORROW** or **YESTERDAY** (the time is always midnight (00:00:00.000)) as well as **MONDAY**, **TUESDAY**, **WEDNESDAY**, **THURSDAY**, **FRIDAY**, **SATURDAY** or **SUNDAY** (which calculate a forward date to the next day/date, the time is always midnight).

Examples:

“+5”	5 hours from now.
“-1”	Current time minus 1 hour. (Note that the 1 is interpreted as an hour and not a day because it is not followed by a hyphen.)
“-3-“	Current time minus 3 days. (Note the hyphen.)
“+:5”	5 minutes from now.
“-:5”	Current time minus 5 minutes.

## 3.9 Writing Procedures

Simply stated, a procedure or script, in XLNT terms, consists of one or more XLNT statements and/or commands. Normally a procedure is written, using a text editor, when the task to be performed will need to be performed more than once.

To execute a procedure (or command procedure) under XLNT, use the at sign (@) followed by the file specification of the procedure to be invoked. For example, @c:\asci\Xlnt\process means that the *process.xcp* procedure found in *c:\asci\Xlnt* should be executed. Normally, execution of the procedure is silent in regards to the XLNT console window. To cause the XLNT statements to be seen while executing, specify the SET VERIFY command.

Writing a procedure is somewhat similar to writing a program although an XLNT procedure will be much simpler to write and debug. In both cases, however, you must think about the task(s) to be performed. XLNT procedures can be as modularized as normal traditional programs. When you type XLNT commands on the console window you are operating at command level 0. When you invoke a command procedure it operates at level 1. A command procedure can invoke other command procedures, up to a maximum of thirty two (32) levels.

Command procedures exit in four ways: **EXIT**, **ABORT**, **Control/C** and **LOGOUT**. The EXIT statement returns control to the next command level. The ABORT statement terminates the command procedure by script control. Control/C terminates the command procedure by user input and returns control to level 0. LOGOUT terminates both the command procedure and XLNT.

## 3.9.1 Handling Error Conditions

By default, when an XLNT statement results in an error, XLNT executes an EXIT command. This causes the procedure to exit to the previous command level. When a command procedure exits, the built-in symbol \$STATUS is set to the exit status of the procedure. This symbol can be tested by the preceding command procedure.

Another more advanced method of handling an error is through the ON statement. The ON statement allows you to specify a severity which if reached results in the execution of one or more XLNT commands. When such an error occurs, the following actions are taken:

- Performs the actions specified by the ON statement.
- Sets \$STATUS to the error condition.
- Resets to the default error action.

An ON command action is executed only once. It must be reset through another use of the ON statement. In addition, the ON statement only applies to the actions of the current command level. This means that if you issue an ON statement and then invoke another command procedure, that procedure will not be affected by the ON statement. See the [ON Command](#) for more information as to the format of the statement.

To disable error checking, you may use the SET NOON command. This prevents XLNT from executing the actions of any ON statements found. The SET NOON command is in effect for the current command level only.

Examples:

```
$ ON ERROR THEN GOTO ERR_RTN
$ RUN BAD_PROGRAM
$ DIRECTORY
$ EXIT
$ERR_RTN:
$ WRITE $STDOUT "We got an error."
$ EXIT
```

This example sets up an ON error handler for any error which occurs from the point it was invoked. Assuming the execution of BAD\_PROGRAM results in an error (a good assumption for this example!), XLNT will transfer control to the label ERR\_RTN and processing will continue sequentially from that point onward.

```
$ SET NOON
$ RUN BAD_PROGRAM
$ DIRECTORY
$ EXIT
```

This example shows how to disable error handling through the use of the SET NOON statement. Now when BAD\_PROGRAM is run, the procedure will still execute the DIRECTORY command and exit.

Finally:

```
$ SET NOON
$ RUN BAD_PROGRAM
$ IF $STATUS
$ THEN
$     DIRECTORY
$     .
$ ELSE
$     !ERROR PROCESSING
$ ENDIF
$ EXIT
```

This example is a refinement of the preceding one in that BAD\_PROGRAM's exit status is checked (\$STATUS) for success. If it was successful, we invoke the DIRECTORY command. If not, we can take some other type of action more appropriate to the error.

## 3.9.2 Handling Error Conditions from DOS/CMD Programs

By default, when XLNT runs a DOS/CMD command, no ON error detection or handling is performed. XLNT does provide a special global symbol named \$DOSSTATUS which is set to the DOS program's exit status after the command is issued and exits. \$DOSSTATUS is not interpreted by XLNT since the value itself must be analyzed by the script writer as to whether the program's exit status code indicates success or failure.

To enable ON error detection for DOS programs, you may specify the SET DOSERROR command. The SET DOSERROR command allows you to enable and specify a numeric value that is compared against a DOS program's exit status. If the exit status is equal to or greater than that specified with the DOSERROR command, the program is considered to have failed. If an ON statement is currently active, the ON error handler will be initiated. By default, SET NODOSERROR is assumed, which means that ON error handling for DOS programs is disabled. The complete syntax of the SET DOSERROR command is:

**\$ SET [NO]DOSERROR [integer]**

where *integer* is a value which will be compared to the DOS program's exit status. Specifying SET DOSERROR and omitting its parameter establishes a value of one (1). This means that if the DOS program's exit status is equal to or greater than one (1), the program is considered to have failed.

## 3.9.3 Handling Control/C Interruptions

By default, XLNT terminates a command procedure, regardless of level, when a Control/C is typed. Control is passed back to XLNT at level 0.

To prevent the default action, the ON CONTROL\_C statement is available. As with the usual ON syntax, an XLNT command may be specified as the action when this condition occurs. If a program is being run by XLNT, that program will be terminated prior to executing the actions of the CONTROL\_C handler. Unlike the ON statement, a CONTROL\_C handler remains in effect until either the procedure terminates or another CONTROL\_C handler is invoked.

## 3.10 Command Procedure Input

A command procedure can obtain input through several different facilities.

### **Parameter Passing**

When a command procedure is invoked, a user may specify from zero through eight (8) arguments to be used as input parameters to the invoked procedure. These arguments are loaded into the special built-in local symbols named P1 through P8. Passing less than eight arguments causes XLNT to set the remaining Pn parameters to a null string value.

For example:

```
$ @TEST A B C
```

shows the procedure *TEST* with three arguments (A B C). These arguments are loaded into the input parameter symbols such that P1="A", P2="B" and P3="C". All whitespace (extra space, tabs) are reduced to a single space. All input data is converted to uppercase. To preserve whitespace or case, enclose input parameters with quotation marks, for example,

```
$ @TEST "a" "b" "c d"
```

causes a lowercase "a" to be placed into P1. Note that P3 contains "c d" including the space due to the effects of the quotation marks.

### **Symbols**

You can pass global symbols to a command procedure regardless of level.

### **INQUIRE and READ**

The XLNT commands INQUIRE and READ can be used to interactively request data from the console window.

## 3.11 In-Script Data Processing

*In-Script Data* refers to XLNT's ability to have input data appear within a script after an application is started. For example:

```
$ CREATE TEST.TXT
```

```
This is record 1
```

```
This is record 2
```

```
This is record 3
```

```
^Z
```

```
$ DIRECTORY
```

The above sequence creates a file (TEST.TXT) and provides input data that would normally be requested from the console window. In-script data is a very powerful facility for automating many aspects of a repetitive procedure. For in-script data to be available for use, an application must request input from STDIN. If the application issues console specific service calls, in-script data cannot be used.

Note: The In-Script data stream is terminated when a dollar sign (\$) is encountered at the beginning of a line. If you need to begin a line with a dollar sign character, see the \$DECK and \$EOD statements for more details.

## 3.12 Command Procedure Output

Output from XLNT procedures can be directed to the console window or a file.

### TYPE

Use the TYPE command to display text data that does not require symbol substitution.

### WRITE

Use the WRITE command to write data that contains symbols and/or lexical functions. The special handle symbol \$STDOUT is available for writing data to the console. Unless you enclose the data in quotation marks, symbol substitution is implicitly performed. To write a quotation mark, use two sets (""). To force symbol substitution within a quoted string, surround the symbol with single quote marks preceded by a single quote mark. For example,

```
$ WRITE $STDOUT "Hello my name is "name"
```

in this case the symbol *name* is evaluated and substituted into the WRITE command.

### OUTPUT qualifier

Most XLNT commands support the /OUTPUT qualifier. This qualifier allows you to redirect a command or program's output to a disk file of your choosing. A simple way to redirect an entire procedure's output is by specifying the /OUTPUT qualifier along with the invocation of the command procedure itself. For example:

```
$ @TEST/OUTPUT=TEST.TXT
```

This example command invokes the procedure TEST.XCP and redirects all output (\$STDOUT) to the file TEST.TXT in the current directory. Please note that you may also specify parameters as input to TEST.XCP however the placement of the /OUTPUT qualifier must immediately precede the invocation of the command procedure itself. For example:

```
$ @TEST/OUTPUT=TEST.TXT ABC DEF
```

performs the same redirection of output to TEST.TXT and also passes two string constants ABC and DEF.

The following example, however, would not redirect TEST.XCP's output.

```
$ @TEST ABC DEF /OUTPUT=TEST.TXT
```

This example invokes TEST.XCP and passes three parameters. The /OUTPUT qualifier is now interpreted as a parameter for TEST.XCP as opposed to a qualifier for the @ command.

## 3.13 Reading and Writing Files

Several XLNT statements, in addition to commands, are available for accessing data in a file.

The basic steps to reading and writing files within XLNT are:

1. Use the OPEN statement to open the file, device or stream. The OPEN command creates a special global handle symbol which is used by subsequent file statements for accessing the file. This global handle symbol must not conflict with any other symbol usage. Moreover, the same global symbol cannot be used to open more than one file. A warning condition is raised on an attempt to do so.
2. Use the READ or WRITE statements to read or write data from/to the file.
3. Use the CLOSE statement to close files and free up any resources associated with the file.

The READ and WRITE statements both use expressions (which normally contain a symbol). In addition these statements offer their own form of error handling apart from that provided with the ON statement.

## 3.14 Controlling Execution Flow

Normally command procedures are executed in a sequential manner. However, you can also control whether certain statements are executed or the conditions under which the procedure should continue executing.

Execution flow is normally controlled by these statements:

- XLNT control statements: IF, GOTO, GOSUB and CALL.
- Looping statements: FOR, WHILE, UNTIL

### **Block Structured Statements**

XLNT language statements provide for block type structured programming. Many of the statements that offer this capability are explained in the paragraphs which follow. A Block represents a related set of statements which are intended to be executed as a group. Blocks may be conditionally executed as in an IF statement or repetitively executed as in a FOR statement. Labels declared within a block are local to the block.

### **IF statement**

The IF statement is used to test the value of an expression and determine whether the result is true or false. Two forms of the IF statement are provided by XLNT. The IF *expression* THEN and the IF *expression* with THEN/ELSE blocks. When an IF results in a true indication, the THEN portion of the IF statement is executed. When an IF results in a false indication, the THEN portion is not executed and the ELSE block (if provided) is executed.

The IF ... THEN format results in a single action statement if the result of the IF is true. For example:

```
$ IF TRUE THEN WRITE $STDOUT "TRUE"
```

This example is a non-block structured IF statement.

The IF with THEN and/or ELSE blocks results in more structured code and also provides a way of having several statements executed on the results of the IF. The ELSE block provides a way of executing statements where the IF resulted in a false indication. For example:

```
$ IF TRUE
$ THEN
$   WRITE $STDOUT "TRUE"
$ ELSE
$   WRITE $STDOUT "FALSE"
$ ENDIF
```

This example provides both a THEN and an ELSE block. Multiple statements could have been inserted into either block.

Labels defined within an IF-THEN-ELSE block are local to the block in which they are defined. GOTO operations first examine the local block for label definition before increasing the scope to include the current command level. When a GOTO or other operation results in execution outside of the block, all current block contexts (IF-THEN-ELSE, etc) are terminated.

See [Logical Values and Expressions](#) for more information on determining how expressions are evaluated.

### **Goto Statement**

The GOTO statement allows a command procedure to immediately jump to a label within the command procedure. The label can appear anywhere in the current command level but must be present (or an error indication is raised). Please note that labels may be local within the scope of a block as noted in the explanation of the IF statement.

A convenient method of creating a CASE statement is to use the XLNT symbol substitution facility as part of the GOTO statement. For example,

```
$ COMMAND_LIST="/PURGE/DELETE/EXIT/"
```

```

$GET_COMMAND:
$ INQUIRE COMMAND "Command (EXIT, DELETE, PURGE)"
$ IF F$LOCATE("'" + COMMAND + "'", COMMAND_LIST) .EQ. -
    F$LENGTH(COMMAND_LIST) THEN GOTO ERROR1
$ GOTO 'COMMAND

```

This example shows how the object of a GOTO can be a label that is programmatically derived.

### **GOSUB Statement**

The GOSUB statement transfers control to a labeled block within a command procedure. The GOSUB statement does not change the current command level. Therefore, all labels and local symbols defined in the current command level are available to the labeled GOSUB block. Execution within a GOSUB block is normally ended by using the RETURN statement. As the statement implies, execution of the RETURN statement causes control to be passed back to the statement following the GOSUB. For example:

```

$ GOSUB ASK_NAME
$ WRITE $STDOUT "Your name is ' ' name' "
$ EXIT
$ASK_NAME:
$ INQUIRE name "Please enter your name"
$ RETURN

```

This example GOSUBs to a label named ASK\_NAME. The INQUIRE statement prompts for the user's name and returns to the next sequential statement after the GOSUB.

### **CALL Statement**

The CALL statement provides the ability to create a new command level within a command procedure. The CALL statement transfers control to a labeled subroutine (see [SUBROUTINE Command](#) for more information) and creates a new command level. The label of the subroutine (or its name) must be unique within the command procedure. Using the CALL statement you can also pass up to eight (8) parameters as part of calling the subroutine (the usage within the subroutine is the same as invoking a new command procedure - use the Pn symbols). Labels are local within subroutine blocks, although subroutine blocks can be nested. This means that if one subroutine wants to CALL another it must have visibility to the subroutine's label.

### **FOR Statement**

The FOR statement provides a general looping or iteration facility within XLNT. The syntax of the FOR statement is very similar to that of the C programming language. The FOR statement consists of three parameters enclosed in parentheses; an *initial XLNT command*, an *expression* and an *iteration command*. The *initial command* is any valid XLNT command. Normally it will be an assignment statement to initialize a symbol that will be used to control the loop (for example, A=1). The *expression* defines the test to be performed and determines whether the iteration will continue. The expression is evaluated *after* the iteration command is executed. Any valid XLNT expression can be specified. The *iteration command* is also any valid XLNT command. It will be executed prior to evaluating the expression. Normally the iteration command is used to modify a symbol that controls the loop. The body of the loop is delimited by a related ENDFOR. As long as the expression tests true, the body of the loop will be executed. When the expression tests false, control transfers to the statement following the related ENDFOR command. For example:

```

$ for (count=0, count .le. 6, count=count+1)
$     inquire dayofweek{count} "Please enter the days of the week"
$ endfor

```

This example sets up a loop to capture all seven days of the week. The result of the INQUIRE statement is placed in a numeric indexed array using a self-explanatory symbol name. Please read about the [FOR Command](#) for more information.

### **WHILE Statement**

The WHILE statement is essentially a very simplified iteration loop that tests a specified expression and performs the body of the loop as long as the expression is true. If the expression is false, control transfers to the statement following the related ENDWHILE command. Since the expression test is performed at the *top* of the loop, it is possible that the loop itself will never be executed.

### **UNTIL Statement**

The UNTIL statement is almost exactly the same as the WHILE statement except that the specified expression is tested and the body of the loop is executed as long as the expression is false. This is the exact opposite of the WHILE statement. The body of the loop is delimited by a ENDUNTIL command.

### **LEAVE Statement**

The LEAVE statement is used to immediately exit the current loop block. If the procedure is not executing a loop block, the command is ignored.

### **ITERATE Statement**

The ITERATE command is used to bypass the remaining statements within a loop block *but* to continue the loop with the next iteration.

# 4 Interactive Development Environment and Debugging Facility

The XLNT Professional Edition offers a specialized environment for writing XLNT command procedures (or scripts). This chapter details the XLNT Interactive Development Environment (or IDE) as well as the in-line XLNT Script Debugger.

The IDE is a Windows GUI application which provides a language sensitive script editing capability similar in scope to Microsoft's Developer Studio environment. The IDE provides a "one-stop" application for creating, maintaining, printing, searching and debugging XLNT scripts.

## 4.1 Starting the IDE

When the Professional Edition is installed, an icon named *XIntIDE* is placed on the desktop. The IDE can also be found within the XLNT Program Group. Double clicking on the XIntIDE icon or selecting the program will cause the GUI to begin running. Figure 7 displays the IDE's initial window and start view.

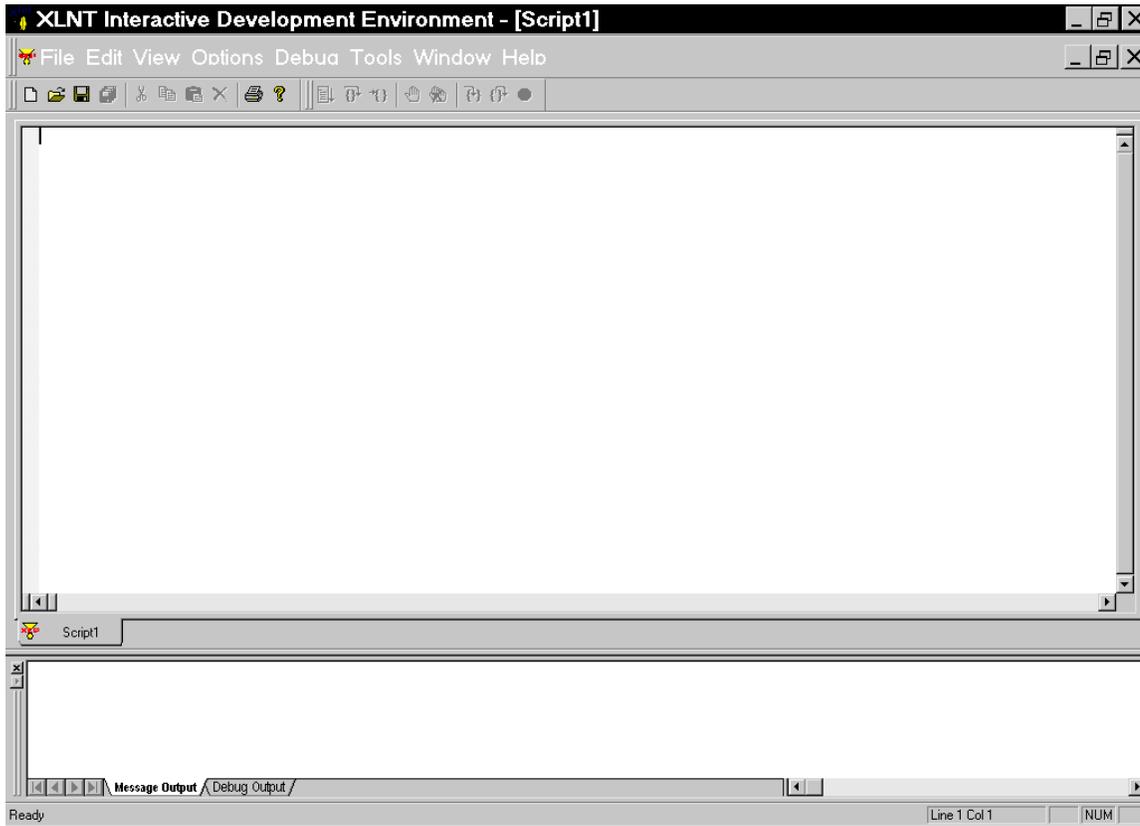


Figure 7. IDE Initial Window

The XIntIDE provides both a menu and tool button bar for requesting various operations. The File Menu allows a scriptwriter to create a new script, open an existing script, print, print preview as well as quickly open the last five (5) most recently accessed script files. Figure 8 displays a sample script file that has been opened by the XIntIDE. Note that the script is displayed with line numbers for ease in locating statements that may be in error. You can elect to remove the line numbers through IDE Customization (see "Options..." under the "Tools" menu).

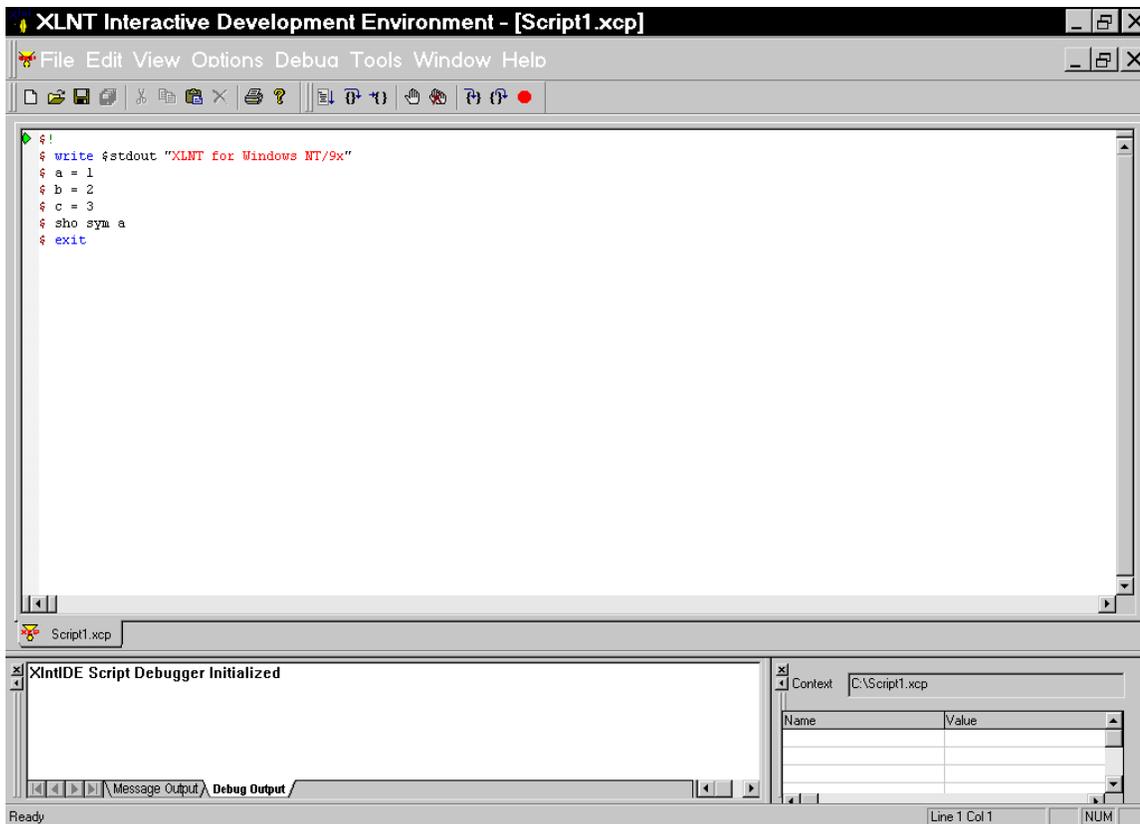


Figure 8. IDE Script View

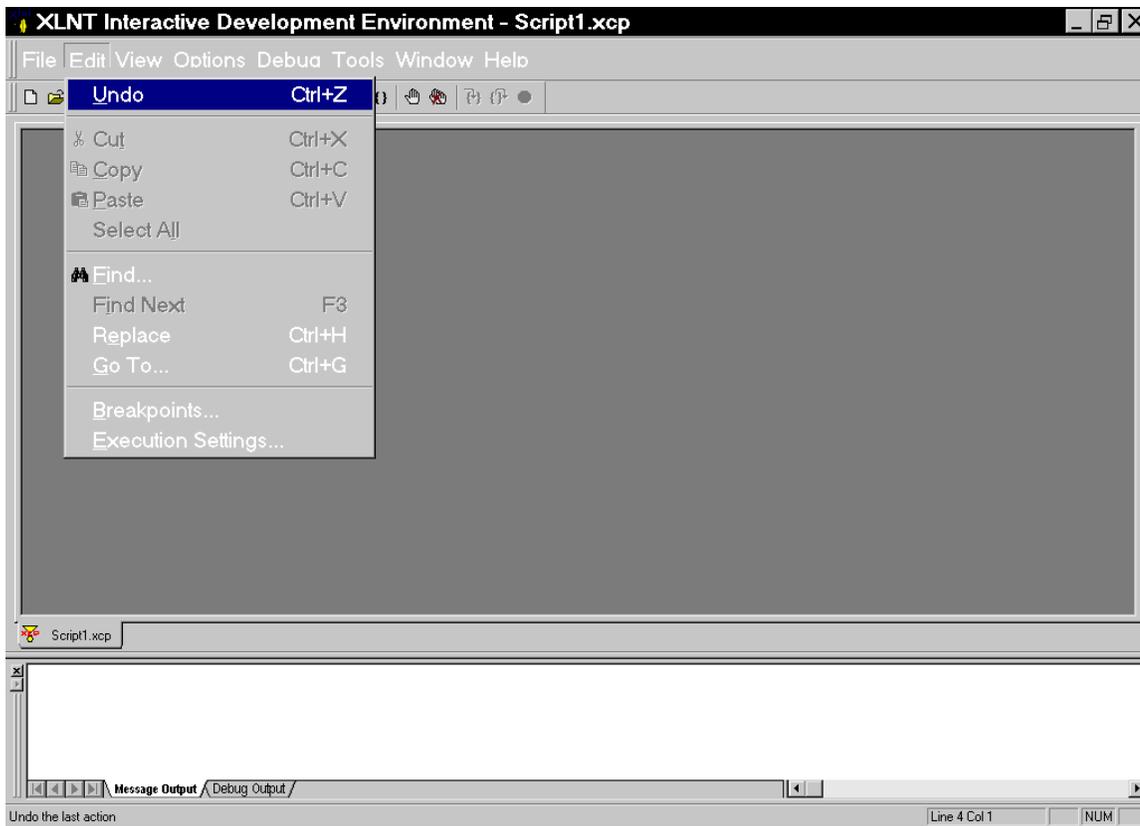
The IDE uses color to denote the different types of statements and commands that XLNT supports. The following table provides the color scheme used for the different statements:

Type	Color
Comments	Green
Commands	Blue
Lexicals	Purple
Arguments	Red

Use of color helps you quickly identify whether the statements and/or commands being coded will work. Of course, depending on the extent that you abbreviate commands or use symbols for command substitution, the IDE may not be able to always color a statement properly. The IDE creates a typical editing environment with the ability to enter and edit text. You can further customize the editor by examining the "Options..." command under the "Tools" menu. Once you have created a script you will need to save it using the *Save* or *Save As* commands. The default file type for an XLNT script is .XCP. If your script is to be invoked through ActiveX scripting then a file type of .XCS should be selected.

## 4.2 Creating and Editing a Script

Creation of an XLNT script through the IDE is fast, simple and straightforward. If you need to create a new script, click on the *New* command under the *File* Menu. This will create a new script document. When the IDE is initially executed, this is the default action. Once you are presented with a script document view, start typing. As you enter XLNT statements, commands and other script syntax, watch for the proper syntax highlighting as described above. The IDE is like most text editors and does provide for copy, cut and paste as well as a text search capability. So you can *Find* and/or *Replace* text by invoking those commands. Figure 9 depicts the *Edit* menu.



**Figure 9. IDE Edit Menu**

When you're finished editing your script, remember to Save (or Save As...) your procedure. The IDE also provides the ability to Print, Print Preview and Send your script as a mail attachment.

## 4.3 Debugging an XLNT Script

When you've coded an XLNT script you will most likely want to execute and/or debug your script especially if scripting errors have been detected. The IDE allows you to simply execute your script using the *Execute* command (found under the "Tools" menu) or to actually begin a debug session. Unlike traditional programming languages you don't have to compile your script with special debug settings. With the selected script file opened, just access the "Debug..." sub-menu.

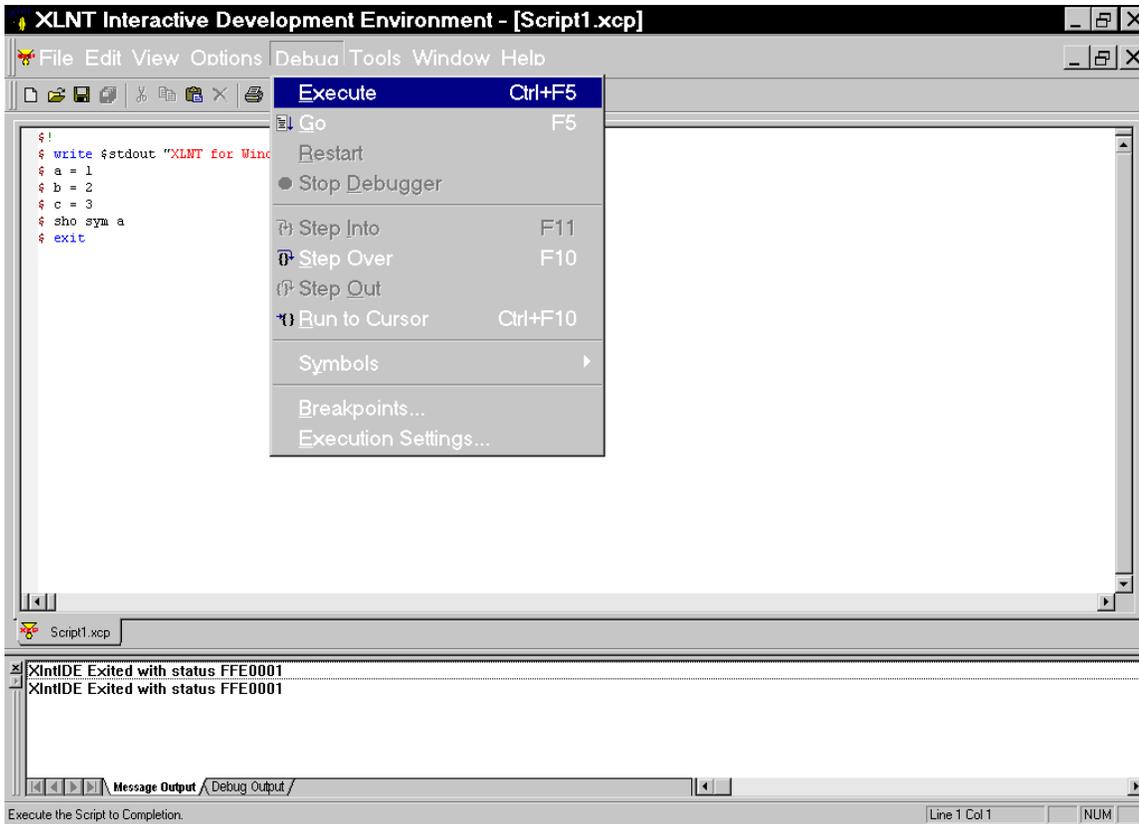


Figure 10. IDE Debug Menu

The Debug menu provides operations which set/clear breakpoints, single statement execution (of varying modes), symbol examination and modification as well as the ability to set *Execution Settings...* . Execution Settings allows you to set both the working (or current) directory when the script is to be executed as well as the input parameters (if any) which are to be passed to the executing script.

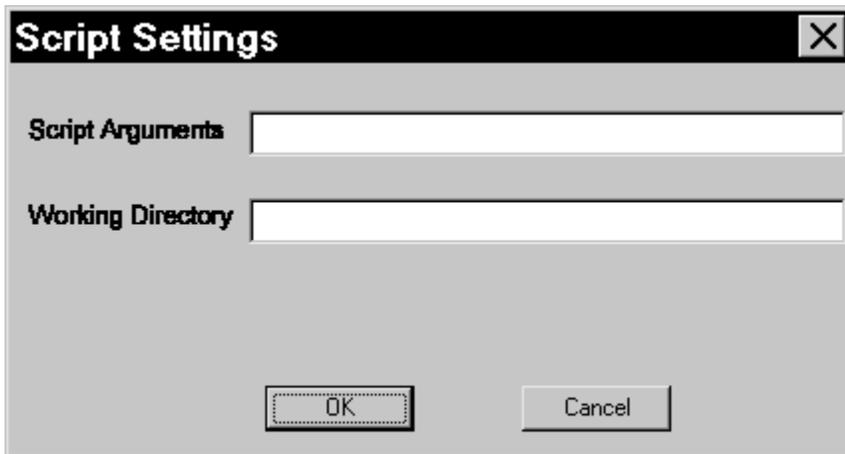
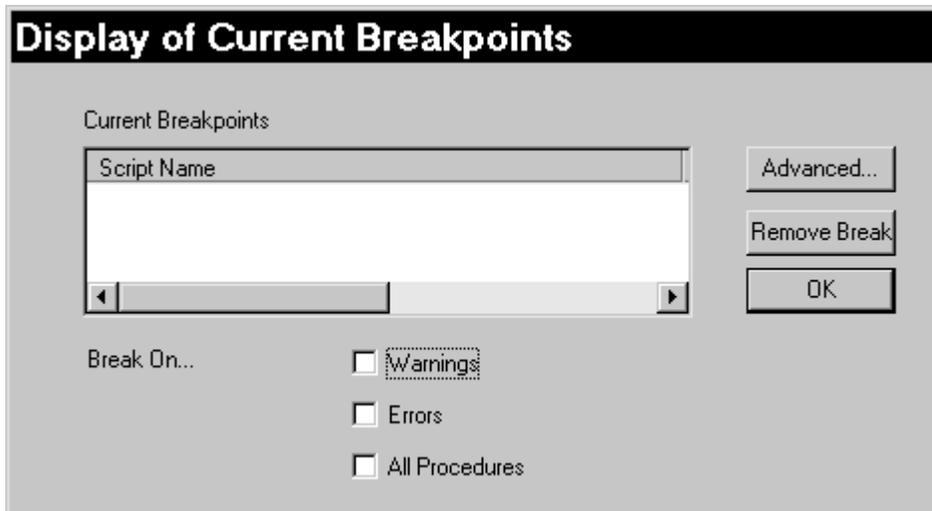


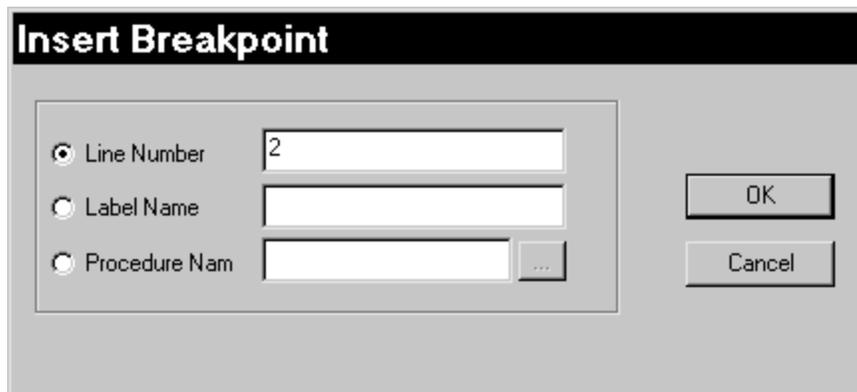
Figure 11. Script Execution Settings Dialog Box

Breakpoints can be set in a variety of different ways. The simplest approach is to place the cursor on the line you want to set the breakpoint and click on the *Set Breakpoint* tool button. To disable a breakpoint, the same approach would be used except the *Clear Breakpoint* tool button would be invoked. To set/clear more complicated breakpoints, click on the *Breakpoints...* sub-menu command (found in both the *Edit* and *Debug* menus).



**Figure 12. Breakpoints Dialog Box**

The Breakpoints dialog box provides an area listing all active breakpoints. Below the list box are several checkboxes which, when enabled, allow general purpose breakpoints based on a specific event. For example, a breakpoint can be generally set whenever a new nested command procedure is invoked. You can also request a breakpoint whenever your script encounters a warning or an error. The dialog box also provides three (3) buttons on the right-side which allow the advanced insertion and/or removal of breakpoints.



**Figure 13. Insert Breakpoint Dialog Box**

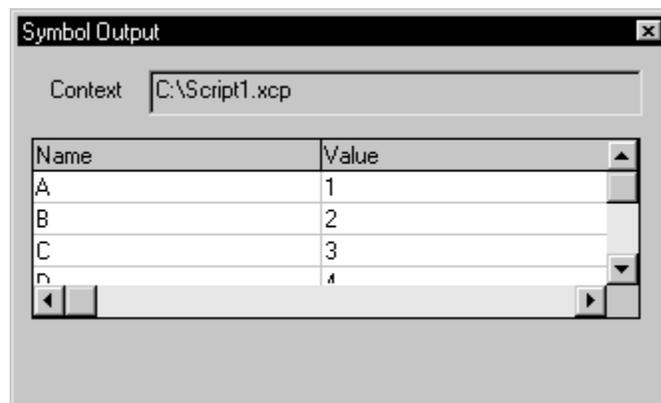
Figure 13 depicts the Advanced Breakpoint Insertion Dialog Box. The "advanced" part is due to the types of breakpoints which can be set. This dialog box allows you specific control of where a breakpoint can be set including the ability to set a breakpoint in an as-yet-to-be-executed script. This is a handy feature when the problem is within a procedure that is nested and will be executed. In this example, a breakpoint is being set on Line 2. If we look at the script view (Figure 14) we can see the effect.



**Figure 14. Script View with Breakpoint**

When a breakpoint has been inserted into a procedure (script), a small red circle is drawn to the left of the script line the breakpoint applies to. The Arrow ( $\Rightarrow$ ) symbol is used to indicate the current line of execution within the script. In addition to inserting and removing breakpoints, you can also enable or disable a breakpoint. This very useful feature allows you to disable a breakpoint temporarily while still retaining the location of the breakpoint. This can be useful when you need to keep executing a particular statement but don't want to service the resulting breakpoint until a specific time within in your script. You can access the breakpoint features via the Debug menu, Tool Bar or by right clicking the mouse in the Script Document View. The IDE also supports *step* execution of the script. This means that you can execute the script a line at a time and determine where things are going wrong. The Step Execution facility provides for the ability to step over, step into or step out any of the XLNT statements or commands. So if you need to step over a GOSUB or a CALL, you have that ability.

As the script proceeds, all script output is written to the Message Output view. Typically a script writer will also want to examine symbols to determine whether the script is executing properly. The XlntIDE provides a Symbol view for that purpose.



**Figure 15. Symbols View**

The Symbol view displays the current command level's symbols. The Symbol view is updated automatically as the script executes and symbol values change. The IDE also allows some control of the scope of symbols displayed. You can elect to display local and/or global symbols. To modify the value of a symbol, execute the Symbols... sub-menu command (under the Debug... menu). This command allows any symbol to be updated manually and can be used to test various conditions or even temporarily correct an errant program.

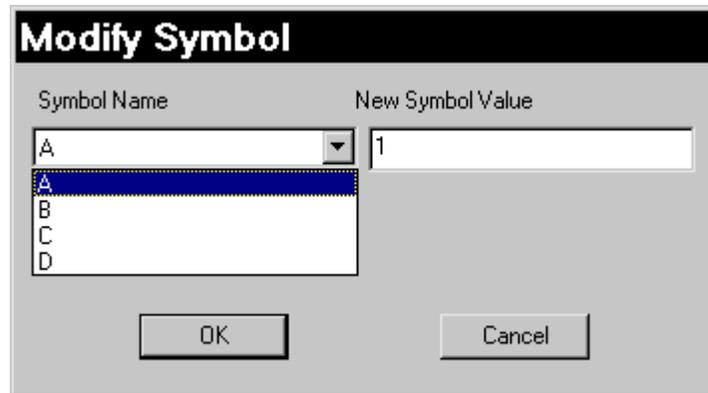


Figure 16. Symbols Modification Dialog Box

## 4.4 Tools and Customization of the IDE

The *XIntIDE* provides various tools and preferential control of the various document views and execution. Examining the *Tools* menu (Figure 17) you can generate an executable image of your script. You can launch a separate interactive XLNT session while your debugging. The *Set Verification* command is can be enabled or disabled to provide a complete audit trail of which statements were executed and the actual symbol substitution performed.

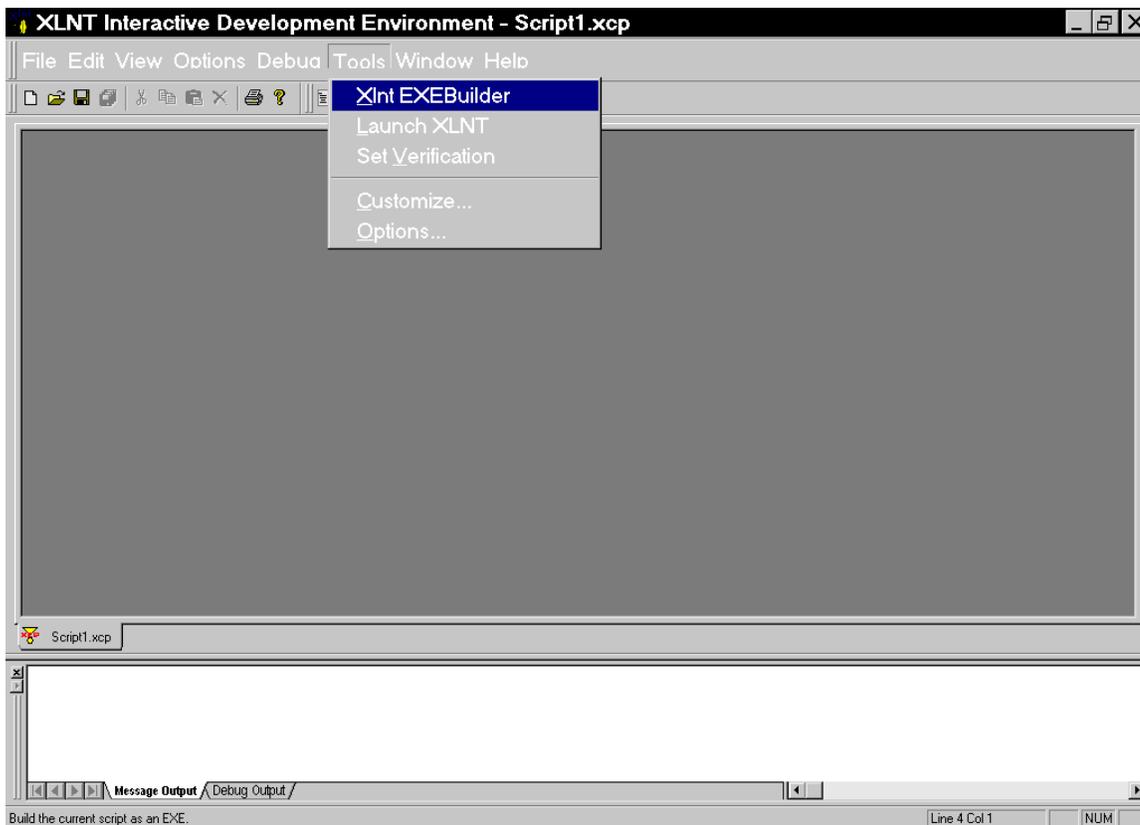


Figure 17. IDE Tools Menu

The Customize Dialog Box (Figure 18) allows you to change the Toolbar buttons and menu commands presented within the IDE.

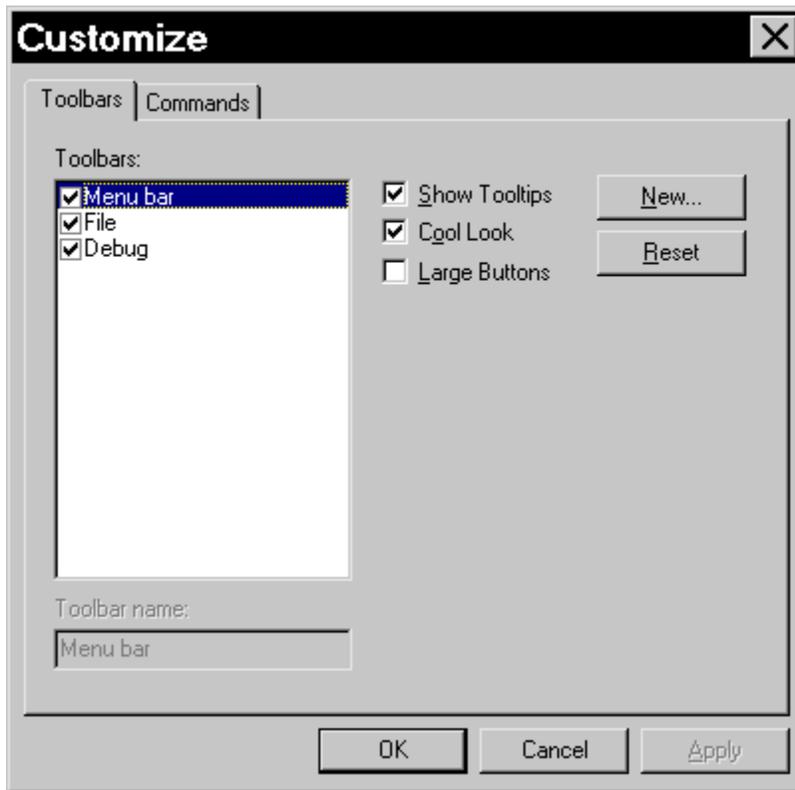


Figure 18. IDE Customize

Finally, the Editor Options Command starts a similarly named dialog box which provides preferential settings of the colors used when syntax highlighting, font and sizes used when displaying a script and other user set-able features of the IDE.

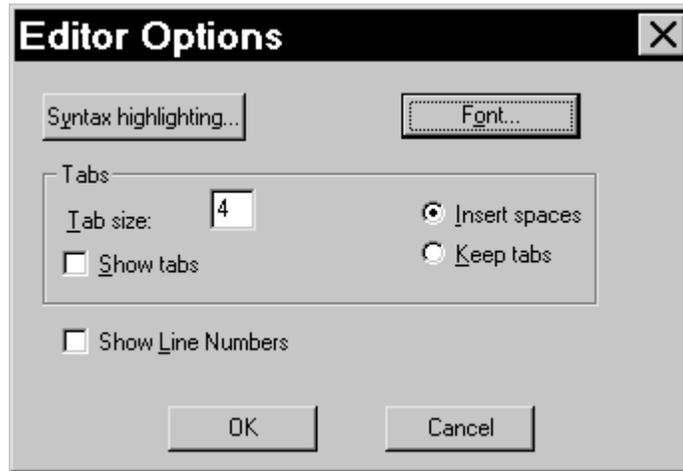


Figure 19. IDE Options (Preferences)

## 5 XLNT and DOS/WIN CMD

XLNT and the Windows DOS/CMD command interpreter are similar in that both are shells which support a command language.

XLNT supports the use of DOS commands through automatic transparent foreign commands as well as the XLNT DOS command. This command allows a user to specify any DOS command with XLNT symbol substitution. While most DOS commands are provided within the XLNT system, several are not. For example, the NET command.

To execute a DOS command under XLNT simply precede the command with the verb DOS. For example,

```
$ DOS TITLE "My window"
```

This command changes the title of the XLNT console window. Please note that changing your directory via the DOS command will *not* work since all DOS commands are executed within their own context.

XLNT also provides a similar approach for executing XLNT commands under the DOS/CMD environment. To execute an XLNT command within a DOS Batch file (.BAT) or within the DOS shell, simply precede the XLNT command with the verb XLNT. For example,

```
c:\> XLNT DIR
```

would execute the XLNT Directory command (and not the DOS version). To execute an XLNT command procedure under DOS, simply specify the XLNT verb followed by the procedure name (including path, if necessary).

```
c:\> XLNT @c:\asci\sample
```

The command procedure *sample.xcp* is executed. Control is always returned back to DOS/CMD after the command or procedure is completed. To transfer control to XLNT while executing within DOS, type the XLNT verb alone. This command causes XLNT to begin interactively. Logging off XLNT causes control to return to DOS. The XLNT command is available in the Professional and Standard Product Editions.

## 6 Using XLNT as a CGI

XLNT provides very powerful facilities for string handling and text searching that far exceeds those of other scripting languages. To invoke XLNT as a part of your CGI script you can employ one of three methods:

1. An executable built by the XLNT XC command (XC is available in the XLNT Professional Edition), or
2. An action statement that executes a batch (.BAT) file that then contains the proper XLNT statement or
3. If either IIS (Internet Information Server) or PWS (Peer Web Services) is installed, you can add a CGI ScriptMap for a .XGI (XLNT CGI Script).

### 1. XC Command.

The XC command is used to create secure CGI scripts. A by product of this command is that a single encoded executable that will run your XLNT procedure as well as XLNT itself. XC has the following format:

```
$ XC input-procedure-file [output-procedure-file]
```

*input-procedure-file* represents a filename for the procedure that should be converted to an XLNT encoded executable.

*output-procedure-file* is an optional parameter that represents the filename of the XLNT encoded executable. If this argument is omitted, the filename of the procedure is used for the executable with a .EXE extension.

For example:

```
$ XC myproc.xcp
```

will cause a file named *myproc.exe* to be created.

### 2. Batch File.

For example, *action="http://a.a.com/cgi-dos/execxInt.bat"*

The batch file would contain the following command:

```
install_path\xInt xInt-procedure-file %1 %2 %3 %4 %5 %6 %7 %8
```

The rules for creating a CGI script apply to XLNT as well. This means the first text written to \$STDOUT must be the content type (for example, "content-type: text/html") followed by two blank lines.

The first three writes of any CGI should be:

```
$ WRITE $STDOUT "content-type: text/html"  
$ WRITE $STDOUT ""  
$ WRITE $STDOUT ""
```

The CONTENT\_LENGTH environment variable value can be obtained through the F\$GETVARIABLE lexical.

### 3. IIS or PWS Script Map.

Note: The XLNT installation procedure automatically adds an XLNT Script Map if you performed the installation and IIS or PWS was already present. This section is only required if you installed IIS or PWS after the XLNT installation.

To create an XLNT Script Map for IIS or PWS use, invoke the XLNT provided script, MAPXGI.XCP located in the XLNT installation directory. You must have Administrative rights to properly execute this procedure.

The information which follows is not presented in a tutorial manner and assumes the reader is familiar with developing

CGI scripts. To that end, a discussion of all pertinent environmental variables is not presented, except as it directly relates to how a user would create a CGI script using XLNT.

## 6.1 Extracting QUERY\_STRING

The QUERY\_STRING environmental variable is a string with the various components delineated by specialized characters. As a direct result of the process some components are represented in a "%hex" format to avoid conflicts or to fill white space such as the "+" to equal a space or "%22" to equal to quote. The delineator for a name and value pair is the "&" the delineator for a value is the "=" . Thus if we break down the QUERY\_STRING first by the "&" we will extract a name and the associated value pair. Now if we break down the extracted string on the "=" the net result is a name and value. To provide a safe way to declare a name we should use some mechanism to prefix all names such as "XLNT\_". The resulting variable would be XLNT\_name = value.

## 6.2 Installing the Sample CGI Script

In order to use this sample program you will need to copy the file to the Web Servers CGI directory. For Microsoft's IIS Server this would normally be INETPUB\SCRIPTS for Netscape Servers this would be \CGI-BIN.

1. Copy "SAMPLE.XGI" to your CGI directory.
2. Change the following line to reflect the correct path to your CGI directory.  

```
$ echo "<form action=/scripts/sample.exe method=put><pre>"
```
3. Type the following command in the CGI directory: **XC Sample.xgi** this will create the file Sample.exe that we will use in our URL.
4. In your Internet Client Browser type the fully qualified URL example `http://www.advsyscon.com/scripts/sample.exe`

*Note: An encoded script built using XC runs SYLOGIN.XCP on startup. You must not include any commands in SYLOGIN.XCP that print anything to \$STDOUT before you print the necessary content type. See **Batch File** above for the necessary lines.*

Please note that this Sample.Xgi procedure has been reformatted to fit within the constraints of the documentation. Therefore, every line that begins with a "\$" denotes a new line. Occasionally, a procedure line wraps and should be considered as a continuation of the previous line.

```
#! Sample.xgi
#! This sample was created by Advanced Systems Concepts and may be
#! freely used.
#!
#! This program provides a working sample CGI.
#! By accessing the CGI you will be provided a new form if the QUERY_STRING $! is NULL. If the QUERY_STRING is greater than NULL the provided
#! information will be processed
#!
$ echo = "write $stdout"
#! echo "Content-type: text/html"      !Turn these four items on to assist in debug
#! echo ""
#! echo ""
#! set ver
$ ON error then goto ERROR_HANDLER
#!
#! ----- Get Information form client -----
#!
$ Agent = F$GETVAR("HTTP_USER_AGENT")
$ Host = F$GETVAR("REMOTE_HOST")
$ IP_Name = F$GETVAR("REMOTE_HOST")
$ IP_Address = F$GETVAR("REMOTE_ADDR")
#!
$ Qs = F$GETVAR("QUERY_STRING")
$ if Qs .eqs. "" Then goto NEW      !Query string is empty so let's send a new form
#!
```

```

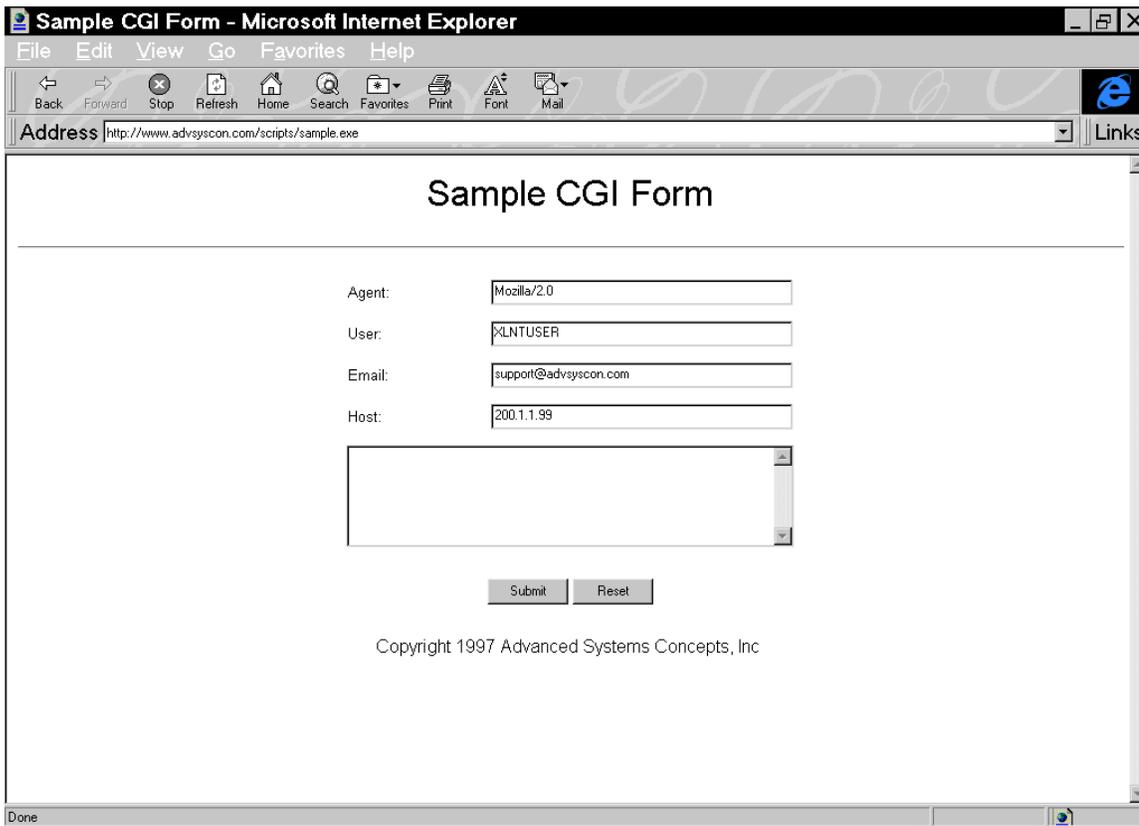
$! ----- Set and Clear Variables -----
$!
$ ElementCounter =0
$ FirstPosition =0
$ Ms == ""
$ Ns == ""
$ VS == ""
$ Hs == ""
$ Rs == ""
$!
$! ----- Extract the information provided -----
$!
$! -- Expand and convert "%hh" --
$!
$ while FirstPosition .ne. F$Length(Qs)
$   FirstPosition = F$Locate("%", Qs)
$   if FirstPosition .eq. F$Length(Qs) then leave
$   Hs = F$Extract(FirstPosition,3,Qs)
$   As = F$Extract(1,2,Hs) ! Convert %hh to %xhh so we can format text
$   As = "%x" + As
$   Qs = F$Replace(Hs,F$Format("%c",'As),Qs)
$ endwhile
$!
$! -- End Expand and convert "%xx" --
$!
$ for( , , )
$   Ms= F$Element(ElementCounter,"&",Qs)
$   if Ms .eqs. "&" then leave
$   Ms = F$Replace("+",",",Ms) ! Expand all spaces
$   Ns = F$Element(0,"=",Ms)
$   Vs = F$Element(1,"=",Ms)
$   XLNT_'Ns = Vs ! Store all name as XLNT_name = Value
$   ElementCounter = ElementCounter +1
$ endfor
$!
$! ----- END Extract the information provided -----
$!
$ goto Process
$ NEW: ! New Form Start
$! Required Line must be followed by a blank line
$ echo "Content-type: text/html"
$ echo ""
$ echo ""
$ echo "<html><head><title>Sample CGI Form</title></head>"
$ echo "<BODY BGCOLOR=#000000 TEXT=#CACA85>"
$ echo "<center><font size=+3>Sample CGI Form</font></center><p>"
$ echo "<hr>"
$ echo "<form action=/scripts/sample.exe method=put><pre>"
$ echo "Agent: <input name=agent size=50 maxsize=60 Value='Agent'><br>"
$ echo "User: <input name=user size=50 maxsize=60 ><br>"
$ echo "Email: <input name=email size=50 maxsize=60><br>"
$ echo "Host: <input name=host size=50 maxsize=60 value='Host'><br>"
$ echo "IP Name: <input name=ipname size=50 maxsize=60 value='IP_NAME'><br>"
$ echo "<TEXTAREA name=comments ROWS= 5 COLS=80></TEXTAREA></pre><br>"
$ echo "<p>"
$ echo "<input type=submit name=submit value=Submit> <input type=reset name=reset value=Reset><br>"
$ echo "Copyright 1997 Advanced Systems Concepts, Inc"
$ echo "</form></body></html>"
$ exit
$ logout
$!
$ Process: ! Process the information provided
$!
$!
$ echo "Content-type: text/html"
$ echo ""
$ echo ""
$ echo "<html><head><title>Returned Sample CGI Form</title></head>"
$ echo "<BODY BGCOLOR=#000000 TEXT=#caca85><hr>"
$ echo "<center><font size=+3>Returned Sample CGI Form</font></center><p>"

```

```

$ echo "<hr>"
$ echo "Agent:      "XLNT_Agent'<br>"
$ echo "User:       "XLNT_User'<br>"
$ echo "Email:      "XLNT_Email'<br>"
$ echo "Host:       "XLNT_Host'<br>"
$ echo "IP Name:    "XLNT_IP_NAME'<br>"
$ echo "Comments:  "XLNT_comments'<br>"
$ echo "<p>"
$ echo "Copyright 1997 Advanced Systems Concepts, Inc"
$ echo "</form></body></html>"
$ exit
$ logout
$!
$!  Error handler routine
$!
$ ERROR_HANDLER:
$ echo "Content-type: text/html"
$ echo ""
$ echo ""
$ echo "<html><head><title>Sorry I've encountered an ERROR</title></head>"
$ echo "<body>CGI - FAILURE<p>REF: SAMPLE.EXE <p>Contact <a href=mailto:webmaster@mysite.com>Webmaster</a></body></html>"
$ exit
$ logout

```



**Figure 20. CGI Screen Powered by XLNT**

```
sample.cgi - Notepad
File Edit Search Help
$ echo "Content-type: text/html"
$ echo ""
$ echo ""
$ echo "<html><head><title>Sample CGI Form</title></head>"
$ echo "<BODY BGCOLOR=#FFFFFFA TEXT=#000000>"
$ echo "<center><font size=+3 face=arial>Sample CGI Form</font><p>"
$ echo "<hr>"
$ echo "<form action=/scripts/sample.exe method=put><pre>"
$ echo "<font face=arial>Agent:</font>          <input name=agent size=50 maxsize=60"
Value='Agent'><br>"
$ echo "<font face=arial>User:</font>          <input name=user size=50 maxsize=60 ><br>"
$ echo "<font face=arial>Email:</font>         <input name=email size=50 maxsize=60><br>"
$ echo "<font face=arial>Host:</font>          <input name=host size=50 maxsize=60"
value='Host'><br>"
$ echo "<TEXTAREA name=comments ROWS= 5 COLS=80></TEXTAREA></pre><br>"
$ echo "<p>"
$ echo "<input type=submit name=submit value=Submit>  <input type=reset name=reset"
value=Reset><br>"
$ echo "<font face=arial><BR><BR>Copyright 1997 Advanced Systems Concepts, Inc</font>"
$ echo "</form></center></body></html>"
$ exit
$ lo
$?
$?
Process:
? Process the information provided
$?
$?
$?
$ echo "Content-type: text/html"
$ echo ""
$ echo ""
```

Figure 21. XLNT Source for Sample CGI

## 7 Using XLNT for ActiveX and Windows Scripting Host Processing

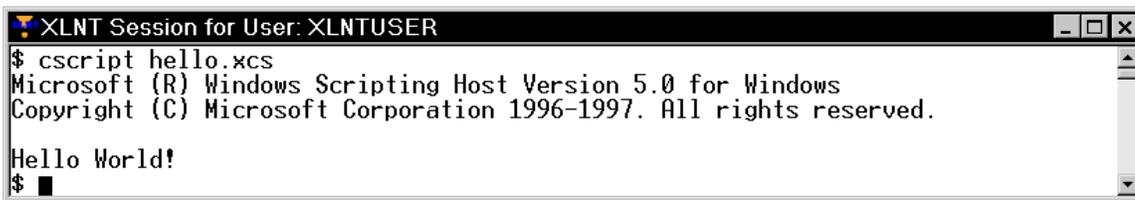
ActiveX Scripting uses OLE Automation components and provides a “client/server” based facility with COM technology as a foundation. Rather than use the words “client/server”, ActiveX Scripting uses the words “host” and “engine”. XLNT is the Script Engine and Microsoft’s Internet Information Server (IIS) and Windows Scripting Host (WSH) are the Script Hosts. The XLNT installation procedure registers itself as an ActiveX scripting engine for scripts that use the .XCS extension. Please note that XLNT registers .XCP for interactive and non-interactive script uses while .XCS is applicable for ActiveX scripting use.

A major benefit to using XLNT for ActiveX scripting is that, unlike VBScript, Jscript or Perl, you can insert XLNT commands as well as scripting statements in your script. So you might read an Excel spreadsheet and create security accounts using the SECURITY CREATE command.

The topic of ActiveX scripting and the use of COM technology is much larger than could be addressed in this chapter and manual. The reader is encouraged to examine various books on the subject as well as Microsoft’s web site for more information. The next several paragraphs will form the very basic definitions of some of the required terms when using ActiveX scripting.

Any given application can be broken down into content and functionality. An *object* refers to a discrete unit of related content and functionality. For Microsoft Word, a document would be an object. An application’s object can be further categorized into an *object hierarchy*. The top-level object is usually called the Application object. For Microsoft Word, the document object is beneath the Application object. As with most hierarchies, the higher the level the broader the scope. As the object hierarchy moves to a lower-level, the object becomes more specific. You can imagine that many complex applications can easily contain large object hierarchies. Manipulation of the object requires that the script access the object’s *methods* and *properties*. Generally, methods help you gain access to an object’s functionality and an object’s properties describe its content. Some properties are meant to be read-only and some allow modification.

While XLNT supports ActiveX usage, most administrators and developers will use XLNT in conjunction with Microsoft’s Windows Scripting Host facility. IIS and WSH both provide ActiveX scripting with WSH requiring much less memory and system resources. WSH also provides additional objects that administrators will most likely want to use. WSH provides a built-in object named *Wscript* that exports several methods. Several other objects are also provided by the WSH facility (for example, SHELL, NETWORK, etc). While most of this manual has the user, in some fashion, directly run XLNT, that is not the case when IIS and WSH are used. When a user wishes to invoke WSH, one of two programs are used: WSCRIPT or CSCRIPT. WSCRIPT is a Windows oriented version of WSH and CSCRIPT is a console command-line version of WSH. Aside from the user-interface orientation both programs are identical in so far as WSH and COM are concerned.



```
XLNT Session for User: XLNTUSER
$ cscript hello.xcs
Microsoft (R) Windows Scripting Host Version 5.0 for Windows
Copyright (C) Microsoft Corporation 1996-1997. All rights reserved.

Hello World!
$
```

For example:

```
$ CSCRIPT hello.xcs
```

uses the command-line version of WSH to invoke the XLNT script “hello”. (Note that the extension is .XCS and not .XCP since .XCS is the type registered for WSH usage. Since .XCS is registered with the operating system for this purpose you can invoke an XLNT WSH script by also double-clicking on the file as well). When CSCRIPT runs it looks up the .XCS extension and determines that a special XLNT component needs to be invoked to process the “hello” script. When XLNT is run in such a manner it inherits the WSCRIPT object. For convenience XLNT creates a special symbol named WSCRIPT that contains that object. So if you wanted to invoke the WSCRIPT method “CreateObject” (to gain access to another object) you would code an XLNT script line like this:

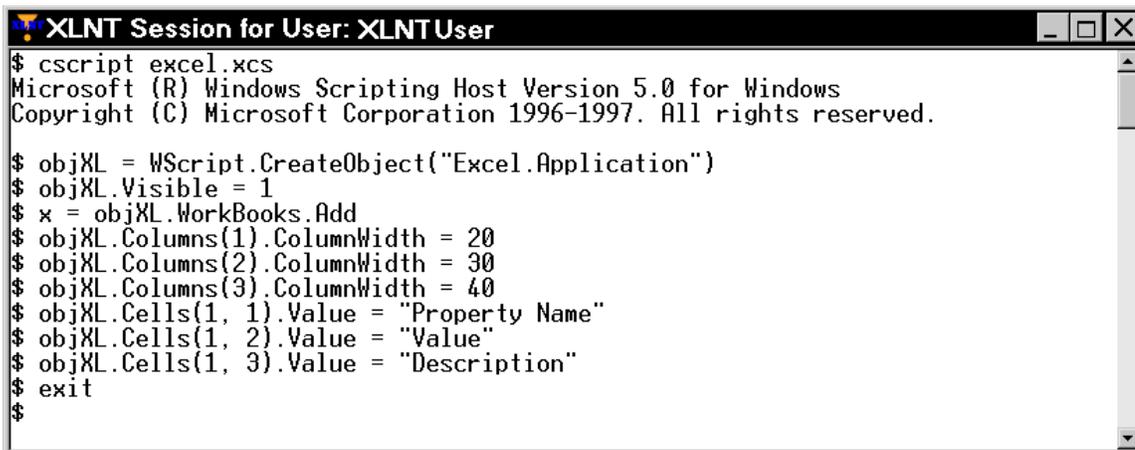
```
$ EXCEL = Wscript.CreateObject ("EXCEL.APPLICATION")
```

In other words to gain access to an object's methods and properties you need to specify the object symbol (WSCRIPT in this case) followed by a period followed by the object's method (CreateObject). This syntactical approach allows you to specify the object's hierarchy. So an object reference is typically zero or more objects (zero only because a default object can be used), followed by a method and/or property. Each reference is separated by a period (or dot). It is important to note that an object's property can also be defaulted. By the way, the actual case (upper/lower) used isn't significant. The above statement creates new object to the Microsoft EXCEL application. The object symbol EXCEL could be used to invoke EXCEL's methods. While you will typically see object symbols and their methods delimited with one or more periods, XLNT does support a default object. The default object is essentially the initial COM component that invoked XLNT. So the following statement is the same as the one above when XLNT has been invoked by WSH:

```
$ EXCEL = CreateObject ("EXCEL.APPLICATION")
```

The reason this disclosure is important is that when XLNT is invoked by IIS, no special object symbol is provided. You simply specify the built-in object's methods. (XLNT provides the WSCRIPT symbol purely as a convenience since VBScript and JScript depend on the symbol and Microsoft manuals and examples use the symbol).

Before we continue it is appropriate at this time to discuss how a script can determine whether it has been invoked by a COM component. Even though the initial script type must be .XCS, an .XCS script can still invoke other .XCP procedures. To determine how your script was executed use the F\$MODE lexical. F\$MODE() will return a string indicating how the procedure was invoked. "ACTIVEEX" or "WSH" are mode strings that will be returned specific to ActiveX scripting.



```
XLNT Session for User: XLNTUser
$ cscript excel.xcs
Microsoft (R) Windows Scripting Host Version 5.0 for Windows
Copyright (C) Microsoft Corporation 1996-1997. All rights reserved.

$ objXL = WScript.CreateObject("Excel.Application")
$ objXL.Visible = 1
$ x = objXL.WorkBooks.Add
$ objXL.Columns(1).ColumnWidth = 20
$ objXL.Columns(2).ColumnWidth = 30
$ objXL.Columns(3).ColumnWidth = 40
$ objXL.Cells(1, 1).Value = "Property Name"
$ objXL.Cells(1, 2).Value = "Value"
$ objXL.Cells(1, 3).Value = "Description"
$ exit
$
```

Figure 22. WSH EXCEL Script

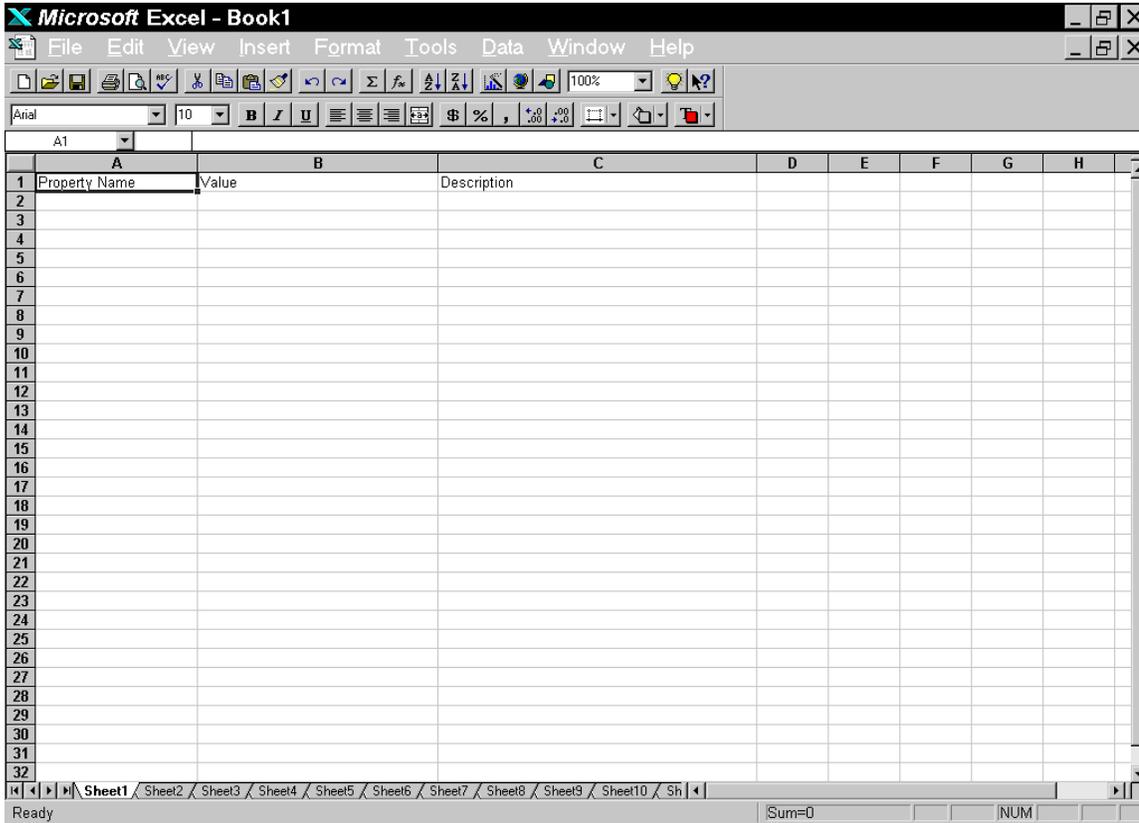
Object references can occur anywhere a symbol or function can be specified within the XLNT language. This example illustrates some of the object references that can be performed.

```
$ on error then goto ExitRtn
$ objXL = WScript.CreateObject("Excel.Application")
$ objXL.Visible = -1
$ x = objXL.WorkBooks.Add
$!
$ objXL.Columns(1).ColumnWidth = 20
$ objXL.Columns(2).ColumnWidth = 30
$ objXL.Columns(3).ColumnWidth = 40
$!
$ objXL.Cells(1, 1).Value = "Property Name"
$ objXL.Cells(1, 2).Value = "Value"
$ objXL.Cells(1, 3).Value = "Description"
$!
$ x = objXL.Range("A1:C1").Select
$ objXL.Selection.Font.Bold = 1
$ objXL.Selection.Interior.ColorIndex = 1
```

```

$ objXL.Selection.Interior.Pattern = 1
$ objXL.Selection.Font.ColorIndex = 2
$!
$ x = objXL.Columns("B:B").Select
$ExitRtn:
$ wscript.quit(1)

```



**Figure 23. EXCEL Application Window**

If you examine the preceding WSH/Excel sample XLNT script, you'll notice that object references can occur on either side of an assignment statement. XLNT also allows an object reference anywhere an expression is specified. Methods and properties may also support arguments depending on the object invoked. A property may retrieve selected information or a property may be used to store a value. An Object can actually represent a group or collection of similar objects. This is called a *Collection Object*. Microsoft Excel contains many collection objects; worksheets, columns, rows, etc. When a collection object is specified, you can iterate through the object to retrieve additional information about the group. This is referred to as *Enumeration*. For example, if you examine the script at the end of this manual named "Display Environmental Variables" you'll find a small procedure which is extracted below:

```

$ Shell = WScript.CreateObject("WScript.Shell")
$ SysEnv = Shell.Environment("PROCESS")
$ write $stdout "Number of Processors: "SysEnv("NUMBER_OF_PROCESSORS")"
$ write $stdout "Processor Architecture: "SysEnv("PROCESSOR_ARCHITECTURE")"

```

The object symbol *SysEnv* is actually a collection object. The object represents the names of each environmental variable. When an object is specified without any accompanying methods or properties, the default property is used. So while *SysEnv* is a collection object the specification of an argument indicates that a specific object in the collection is requested. To enumerate through all the objects of a collection object, the following code would be used:

```

$ SysEnv = Shell.Environment("PROCESS")
$ for (j=0, j .lt. 'SysEnv.Count, j=j+1)
$ Write $STDOUT SysEnv(j)

```

\$ endfor

The *.Count* property returns the number of objects in the collection. Many collection objects are zero based for the purposes of iteration (COM implementors are free to use a zero or one based index for the initial object -- so you do need to check).

XLNT scripts invoked through WSH gain several additional advantages. One benefit is that the XLNT scripting engine will automatically convert the first eight (8) command-line arguments specified to WSH (WSCRIPT or CSCSCRIPT) into P1 through P8 input parameters. This feature allows an additional level of transparency in that XLNT symbol substitution and parameter passing mechanisms can be used during invocation of the XLNT scriptlet.

## 8 Using XLNT for Logon Script Processing

XLNT is the perfect script language for executing various scripts at logon time. First, XLNT provides many built-in symbols which provide perfect context for describing the user who is logging in. Second, XLNT provides many built-in functions which allow User/Group access, Registry access, Domain and Machine enumeration. Finally, XLNT provides [SECURITY commands](#) which are perfect for retrieving and maintaining user/group security. Logon scripts will most likely use the F\$USERINFO and F\$GROUPINFO lexicals for determining various security characteristics of the user who is logging in. F\$ENUMUSER and F\$ENUMGROUP are useful for enumerating user and group (global or local) accounts on either a domain or a local machine.

To associate a given XLNT logon script with a user simply enter the complete login script path name using the SECURITY MODIFY USER username /SCRIPT\_PATH=file-specification.

Hint: (Professional and Run-Time Editions) After writing your XLNT Login Script you might want to encode your script into an executable image using the XC command. You would then specify the .EXE version as the qualifier value for /SCRIPT\_PATH. Using an encoded script means that you don't have to be concerned about unauthorized changes to the script.

In actuality the "Logon Domain Controller" designation for RPC can be *any* Windows NT system. The important requirement for that WinNT system is that it be running the appropriate RPC Services and the XLNT Service. To facilitate that process, another command procedure has been provided named *<XLNT-installation-path>WINNTRPC.XCP*. This procedure is meant to execute only on WinNT systems and will first check whether the appropriate RPC and XLNTservices are installed and/or started. If they are not installed or started, a confirmation message will appear requesting permission to install or start the services. Of course, this procedure can only be executed by users who have the appropriate rights to install and/or start services. A sample execution of the procedure is listed below:

```
***** SCRIPT Start: The script WINNTRPC.XCP has started *****
```

```
XLNT RPC Service not installed on this computer.  
Would you like to install it? (Y/N): Y
```

```
XLNT RPC Service is installed, but not active on this computer.  
Would you like to start it? (Y/N): Y  
XLNT RPC Service is START PENDING
```

```
***** SCRIPT COMPLETION: WINNTRPC.XCP has completed successfully *****
```

## 9 File-Processing Command Summary

This section summarizes the syntax of the XLNT file-processing commands. As the name implies, the commands in this category are used to manipulate files

### 9.1 APPEND Command

The APPEND command will add the contents of one or more specified input files to the end of the specified output file.

**Format:**

**APPEND** input-file[,...] output-file

**Parameters:**

**input-file[,...]**

Specifies the names of one or more input files to be appended. If more than one input file is specified, each file name must be separated by a comma. Wild cards may be specified.

**output-file**

Specifies the name of the file to which the input files will be appended.

**Description:**

The APPEND command is similar syntactically to the COPY command. The APPEND command adds the contents of one or more files to the end of a specified output file. The files are appended in the order specified as part of the *input-file* parameter.

**Qualifiers:**

**/ACCESS**

Selects the date the file was last accessed. For use with the /SINCE and /BEFORE qualifiers.

**/BEFORE[=time]**

Selects only those input files dated prior to the specified time. You can specify an absolute time or a combination time as per the [Date and Time Format](#) rules. The current date and time are the default.

**/[NO]CONFIRM**

Controls whether a request is issued before each append operation to confirm that the operation should be performed on that file. Valid responses are: YES or NO. The default response is NO.

**/CREATE** (default)

Selects the file creation date. For use with the /SINCE and /BEFORE qualifiers.

**/EXCLUDE=(filename[,...])**

Excludes the specified files from the append operation.

**/[NO]LOG**

Controls whether the APPEND command displays the file names of each file appended.

**/[NO]NEW**

Controls whether the APPEND command creates a new output file if the specified output file does not exist.

**/SINCE[=time]**

Selects only those input files dated after the specified time. You can specify an absolute time or a combination time as per the **Date and Time Format** rules. TODAY is the default.

**/WRITE**

Selects the date the file was last written to. For use with the /SINCE and /BEFORE qualifiers.

**Examples:**

**\$ APPEND A.1,A.2 A.A**

This command will append the files A.1 and A.2 to A.A.

**See also:**

**COPY CREATE DELETE**

## 9.2 CLOSE Command

The CLOSE command closes a file opened with the OPEN command and deletes the associated file-symbol.

**Format:**

**CLOSE file-symbol**

**Parameter:**

**file-symbol**

Specifies the symbol assigned to the file when it was opened with the OPEN command.

**Description:**

CLOSE is used to close files that have been opened via the OPEN command. Please note that even after a command procedure exits the file remains open until either XLNT is terminated (via LOGOUT) or the CLOSE command is issued. This has the effect of allowing discrete command procedures to have access and retain context to files opened by other command procedures.

**Qualifiers:**

**/ERROR=label**

Specifies a label in the command procedure to receive control if the close operation results in an error.

**Example:**

```
$ OPEN/READ INFILE TEST.DAT
$ FOR (,,)
$ READ/END_OF_FILE=DONE INFILE INFILE_RECORD
.
.
.
$ ENDFOR
$ DONE:
$ CLOSE INFILE
```

This example illustrates the use of the CLOSE command. First a file named TEST.DAT (in the current directory) is opened for read access. The file symbol handle INFILE is associated with the opened file. All future XLNT access will use that file handle symbol when referring to TEST.DAT. A FOR loop is established to read all the records in INFILE. Each READ of INFILE causes the data to be placed in the symbol INFILE\_RECORD. An *end-of-file* condition causes control to be transferred to the label DONE where the file referenced by INFILE is closed.

**See Also:**

**OPEN READ WRITE**

## 9.3 COPY Command

The COPY command creates a new file from one or more existing files.

### Format:

**COPY input-file[,...] output-file**

### Parameters:

**input-file[,...]**

Specifies the names of one or more input files to be copied. If more than one input file is specified, each file name must be separated by a comma. Wild cards may be specified. (**ftpurl**)

**output-file**

Specifies the name of the file to which the input files will be copied. (**ftpurl**)

### Description:

The COPY command is used to copy one or more files from one location to another. The location could be a different directory, device or machine (or any combination). COPY can do the following:

- Copy an input file to an output file.
- Concatenate one or more input files into an output file.
- Copy a group of input files into a group of output files.
- Copy a group of input files (and sub-directories) into a group of output sub-directories.
- Copy files using existing permissions for use on a single machine.

By default, COPY creates one output file. However, if you select more than one input file, the first file is copied to the output file and each subsequent input file is appended to the end of the output file. To create multiple output files, simply specify either an asterisk (\*) or other wildcard designation in the *output-file* parameter or use the /NOCONCATENATE qualifier. When wildcarding is used in the *output-file* parameter, COPY will merge any missing file specification information from the first named input file. COPY, as with most XLNT file commands, provides both name, date and other criteria that can be used to determine which files are to be copied.

### Permissions

By default, COPY will copy NTFS files and apply the inherited permissions associated with the destination directory. Use of the /PERMISSIONS qualifier allows you to override that default, and retain the original permissions associated with the input file(s). However, please note that if the input file(s) contain local machine permissions, the COPY command will fail. So this feature is most useful when the copied files are expected to be used within the machine on which the COPY command is issued.

### Directory and Sub-directories

If you copy a file that is a directory, COPY will create a new empty subdirectory of the named file. For example:

```
$ COPY C:\*. * D:\*. *
```

will copy all subdirectories of C:\ to D:\ as empty directories (if you wanted the contents of the subdirectories you would have specified a directory wildcard (i.e. C:\...\ and D:\... ). This feature is useful when you want to copy the directory structure but not the files within the structure.

### Qualifiers:

**/ACCESS**

Selects the date the file was last accessed. For use with the /SINCE and /BEFORE qualifiers.

**/ASCII**

This qualifier, when specified with an FTP URL, indicates that an ASCII file transfer is to be performed. By default, a

binary transfer is performed.

### **/ASK\_REPLACE**

Indicates that should a file exist (and therefore conflict) under the output specification, to prompt the user prior to replacing the file for confirmation. /REPLACE (the default) will override this qualifier so /NOREPLACE must also be specified. The prompt requires an answer of one of the following; YES, NO, ALL and QUIT. ALL means that the answer is YES and that no further prompting for file conflicts should occur.

### **/ATTRIBUTES**

If omitted, the compression attribute is not enabled for the destination file. This is the default action for this command and for file copying using the Windows GUI facilities. If specified, the compression and all original attributes are applied at the destination file.

### **/BEFORE[=time]**

Selects only those input files dated prior to the specified time. You can specify an absolute time or a combination time as per the [Date and Time Format](#) rules. The current date and time is the default.

### **/BINARY**

This qualifier, when used with an FTP URL, indicates that a binary file transfer is to be performed. This is the default.

### **/[NO]CONCATENATE**

Controls whether multiple input files (particularly a wildcard operation) should automatically append to a specified output file (no wildcard specifications). By default, the operation is allowed.

### **/[NO]CONFIRM**

Controls whether a request is issued before each copy operation to confirm that the operation should be performed on that file. Valid responses are: YES or NO. The default response is NO.

### **/CREATE** (default)

Selects the file creation date. For use with the /SINCE and /BEFORE qualifiers.

### **/EXCLUDE=(filename[,...])**

Excludes the specified files from the copy operation.

### **/[NO]LOG**

Controls when the COPY command displays the file names of each file copied.

### **/[NO]PASSIVE**

This qualifier, when used with an FTP URL, indicates whether the FTP session should be established with or without a passive port assignment. The default is NOPASSIVE. Some firewalls require that /PASSIVE be asserted.

### **/PERMISSIONS**

This qualifier, when specified, causes COPY to copy files and maintain existing permissions. By default, destination files observe the default permissions established by the directory.

### **/[NO]REPLACE**

This qualifier allows or prevents the overwriting of an existing file.

### **/SAVE=filename**

If the output file currently exists, this qualifier allows it to be saved, under the filename specified, before the COPY operation overwrites it.

### **/SINCE[=time]**

Selects only those input files dated after the specified time. You can specify an absolute time or a combination time as per the [Date and Time Format](#) rules. TODAY is the default.

### **/[NO]SUBDIRECTORY**

This qualifier, /SUBDIRECTORY, if specified, means that subdirectories should be copied if encountered. By default, subdirectories are not copied when encountered.

### **/WRITE**

Selects the date the file was last written to. For use with the /SINCE and /BEFORE qualifiers.

#### **Examples:**

```
$ COPY PROD.DAT PROD.OLD  
$ COPY/CONFIRM *.* D:\XLNT\*.OLD
```

The first command is a simple file COPY where the destination file has a different extension specified. The second command allows manual selection of all input files (due to the wildcards) prior to copying them to an output area. As each selected file is copied, the extension will be changed as well.

```
$ IF F$MODE() .EQS. "INTERACTIVE" THEN COPY == "COPY/ASK"  
$ COPY *.* C:\TEMP\*.*
```

This example shows how XLNT can be used to dynamically adapt the COPY command depending on whether you are logged in interactively or as a batch job. The first command checks whether you are interactive and if so modifies the COPY verb through a symbol named COPY such that the /ASK\_REPLACE qualifier is appended as the default action. Such a default might not be wanted in a non-interactive procedure hence the check.

```
$ COPY \\NTSRV1\D$\TEST\*.* -  
  "FTP://FTP.HITECH.ORG/TEMP/*.*"
```

This example shows an input UNC specification used to copy files from machine NTSRV1 share D\$ directory TEST (all files) via FTP access to FTP.HITECH.ORG location C:\TEMP. The FTP specification assumes an anonymous login is possible on the node. If not, something like "FTP://user:password@FTP.HITECH.ORG..." could be specified (the lowercase is for better illustration, it has no effect on the command. Please note that the FTP URL must be enclosed in quotation marks.

#### **See also:**

[APPEND CREATE DELETE RENAME TYPE](#)

## 9.4 CREATE Command

The CREATE command creates a sequential disk file.

**Format:**

**CREATE file-name**

**Parameter:**

**file-name**

Specifies the name of the files to be created. Wildcard characters are not allowed.

**Description:**

The CREATE command is used to create a new raw disk file. CREATE is interactive and expects you to enter a line of data followed by the ENTER key. Each line of data becomes a record in the file. To terminate the input process and close the file, enter Control/Z on the keyboard.

**Qualifier:**

**/[NO]LOG**

Displays the name of each new file created as the command executes.

**Examples:**

**\$ CREATE SAMPLE.DAT**

This is line 1

This is line 2

**^Z**

This command creates a file named SAMPLE.DAT in the current directory. The input data is retrieved from STDIN. A Control/Z signifies the end of input data and the file SAMPLE.DAT is closed.

**See also:**

[APPEND COPY DELETE RENAME](#)

## 9.5 CREATE/DIRECTORY Command

The CREATE/DIRECTORY command creates a new directory or sub-directory.

### Format:

**CREATE/DIRECTORY directory-name**

### Parameter:

**directory-name**

Specifies the name of the directory or sub-directory to create. See [Files and Directories](#) for more information.

### Description:

The CREATE/DIRECTORY command can create both directories and subdirectories. You may need special permissions or rights depending on the volume on which you want to create the directories. You can create a nested directory in a single command. For example, if you wanted to create C:\SOFTWARE\V2.0\SOURCE and neither C:\SOFTWARE or C:\SOFTWARE\V2.0 exists, simply specify:

```
$ CREATE/DIRECTORY C:\SOFTWARE\V2.0\SOURCE
```

CREATE will search for C:\SOFTWARE and if it doesn't exist will create it. The same is true for C:\SOFTWARE\V2.0. This nesting approach is useful in that you don't need to worry about whether a particular directory path already exists or not, just specify the directories you want created and the CREATE/DIRECTORY command will do the rest. Remember that CREATE/DIRECTORY creates directories while the CREATE command creates files.

### Qualifiers:

**/[NO]LOG**

Controls whether the CREATE/DIRECTORY command displays the name of each directory after creating it.

### Examples:

```
$ CREATE/DIRECTORY D:\TEST
```

This command creates the directory TEST on device D:.

```
$ CREATE/DIRECTORY \\SERVER4\C$\MASTER\NA
```

This command creates the directory \MASTER\NA on \\SERVER4\C\$.

### See also:

[DELETE](#)

## 9.6 DELETE Command

The DELETE command deletes one or more files from a disk device.

The DELETE command deletes directories and/or sub-directories.

### Format:

**DELETE file-name[,...]**

### Parameter:

**file-name[,...]**

Specifies the names of one or more files to be deleted. (**ftpurl**)

### Description:

The DELETE command is used to delete one or more files and/or directories. You may need special permissions or rights to actually delete files and/or directories. To delete a directory you must specify the /DIRECTORY qualifier.

### Qualifiers:

#### **/ACCESS**

Selects the date the file was last accessed. For use with the /SINCE and /BEFORE qualifiers.

#### **/BEFORE[=time]**

Selects only those files dated prior to the specified time. You can specify an absolute time or a combination time as per the [Date and Time Format](#) rules. The current date and time is the default.

#### **/[NO]CONFIRM**

Controls whether a request is issued before each delete operation to confirm that the operation should be performed on that file.

#### **/CREATE** (default)

Selects the file creation date. For use with the /SINCE and /BEFORE qualifiers.

#### **/DIRECTORY**

If specified indicates that the directory should be deleted. When coupled with a wildcard operation, this qualifier will delete all eligible files and subdirectories and then delete the top-level portion of the directory that was actually specified. /SUBDIRECTORY is enabled when this qualifier is present.

#### **/EXCLUDE=(file-name[,...])**

Excludes the specified files from the delete operation.

#### **/[NO]FILES**

Includes or excludes files when used in conjunction with the /SUBDIRECTORY qualifier. By default, /FILES is assumed.

#### **/[NO]LOG**

Controls whether the DELETE command displays the name of each file after its deletion.

#### **/SINCE[=time]**

Selects only those files dated after the specified time for deletion. You can specify an absolute time or a combination time as per the [Date and Time Format](#) rules. TODAY is the default.

#### **/[NO]SUBDIRECTORY**

This qualifier, /SUBDIRECTORY, if specified, means that subdirectories should be deleted if encountered. By default, subdirectories are not deleted when encountered.

### **/WRITE**

Selects the date the file was last written to. For use with the /SINCE and /BEFORE qualifiers.

#### **Examples:**

```
$ DELETE A.A
```

```
$ DELETE/CONFIRM A*.*
```

The first DELETE command will delete the file A.A from the current directory (assuming it exists). The second DELETE command processes an input wildcard specification and then requests manual confirmation prior to the file deletion.

#### **See also:**

**CREATE**

## 9.7 DELETE/SYMBOL Command

The DELETE/SYMBOL command deletes a symbol and frees any associated memory storage.

### Format:

**DELETE/SYMBOL symbol**

### Parameter:

**symbol**

Specifies the name of a symbol or structure definition that is to be deleted and removed from the symbol table. If a structure definition is specified, the structure must not be referenced by an active symbol. You cannot delete a specific array element however you can delete an array symbol (and thus the array in its entirety).

### Description:

The DELETE/SYMBOL command deletes an XLNT symbol and any associated storage or resources from the symbol table. Symbols can be local or global and if you don't specify any qualifiers, the command assumes the symbol is located in the local symbol table. Remember that local symbols are allocated on a command level basis so you must be at the appropriate level to delete a local symbol. To delete a global symbol use the /GLOBAL qualifier. To delete all symbols regardless of whether they are local or global use the /ALL qualifier. Note that the /SYMBOL qualifier must immediately follow the DELETE command to avoid an ambiguity error.

### Qualifiers:

**/ALL**

If specified, the symbol parameter may be omitted. This qualifier indicates that all symbols (either local or global, depending on the presence of the /GLOBAL qualifier) are to be deleted from the symbol table.

**/GLOBAL**

If specified, indicates that the symbol specified is a global symbol and that symbol table is searched. By default, and when this qualifier is omitted, the local symbol table of the current command level is used.

### Examples:

```
$ A="123"
```

```
$ DELETE/SYMBOL A
```

This example shows how symbol A is deleted and its storage is deallocated. Deleting a symbol is useful for freeing object references, structure symbols, arrays and especially "context" symbols that are used in XLNT built-in functions (lexicals) for sequential access through various informational data structures.

## 9.8 DIFFERENCES Command

The DIFFERENCES command compares the contents of two files and displays a listing of the records that don't match.

### Format:

**DIFFERENCES input-file1 input-file2**

### Parameters:

#### **input-file1**

Specifies the first file to be compared. This file is considered the master input file. Wildcard characters are not allowed.

#### **input-file2**

Specifies the second file to be compared against the first file. This file is considered the revision input file. Wildcard characters are not allowed.

### Description:

The DIFFERENCES command (or DIFF for short) is used to determine whether two files are identical or whether they differ. Each file is compared on a line-by-line basis and an output file can be produced listing the differences. DIFF provides many qualifiers including selection criteria qualifiers consistent with other XLNT file commands. DIFF also provides special qualifiers for its own processing which can be categorized as follows:

- Qualifiers that request DIFF to ignore data in each record: /COMMENT\_DELIMITERS and /IGNORE. By default, DIFF compares every character in each line.
- Qualifiers that control DIFF's output file formatting: /IGNORE, /MERGED, /NUMBER, /PARALLEL, /WIDTH. By default, DIFF merges the differences it finds. Each difference record is listed that has no match in the other input file and a single line is listed that does match up against the other input file.
- Qualifiers that control the extent of the comparison: /MATCH, /MAX\_DIFFERENCES. By default, DIFF reads each line in the master input file and looks for a matching line in the revision input file. The search for a match continues until the end of the revision input file is reached or a match is found. By default, three lines must match for the files to be considered identical again.

### Qualifiers:

#### **/COMMENT\_DELIMITER[=(character[,...])]**

This qualifier indicates that those lines starting with the specified comment characters are to be ignored for DIFF purposes. This qualifier is used with or without the /IGNORE=COMMENTS qualifier. If a single character is specified then the parenthesis can be omitted.

#### **/IGNORE=(keyword[,...])**

Inhibits the comparison of the specified characters, strings or records; also controls whether the comparison records are output to the listing file as edited records or exactly as they appeared in the input files. Valid keywords are:

<b>BLANK_LINES</b>	Blank lines between data lines.
<b>COMMENTS</b>	Data following a comment character. (Use the /COMMENT_DELIMITER qualifier to designate one or more comment characters)
<b>HEADER[=n]</b>	First n records of the files are ignored.
<b>SPACING</b>	Extra blank spaces or tabs within data lines.
<b>TRAILING_SPACES</b>	Space and tab characters at the end of a data line.
<b>EDITED</b>	Omits ignored characters from the output file.

#### **/MATCH=size**

Specifies the number of records that should indicate matching data after a difference is found. The default of three (3) indicates that when 3 sequential records are found that match, synchronization is considered achieved for the purposes of detecting differences.

**/MAXIMUM\_DIFFERENCES=n**

Terminates the DIFFERENCES command after the specified number of unmatched records are detected. By default, no maximum number of differences is established.

**/MERGED[=n]**

Specifies the number of matched records that should be included at the end of a differences section. The value "n" must be less than or equal to that of the /MATCH qualifier. By default, one merged record is included at the end of a differences section or /MERGED=1.

**/[NO]NUMBER**

Includes line numbers in the output file of differences.

**/OUTPUT[=file-spec]**

Specifies an output file where potential differences records are written. By default, differences records are written to \$STDOUT (usually the console window).

**/PARALLEL[=n]**

Lists the records with differences side by side. The value of the parameter n specifies the number of matched records to merge after each unmatched record. By default, DIFF does not list records after each list of unmatched records.

**/WIDTH=n**

Specifies the width of the lines in the output file.

**Example:**

**\$ DIFF c:\file2.txt c:\file3.txt**

```
*****
C: \FILE2. TXT
 1 $ *b=f$length(' p1)
 2 $ write $stdout "' 'b' "
 3 $ exit
*****
C: \FILE3. TXT
 1 $ a == "1"
 2 $ @tmp2 a
 3 $ exit
*****
```

Number of difference sections found: 1  
Number of difference records found: 3

```
DIFFERENCES
  C: \FILE2. TXT
  C: \FILE3. TXT
```

This example illustrates a simple DIF (short for DIFFERENCES) command between two files. The single difference section is reported by first displaying the difference in the first file specified and then the second file.

**\$ DIFF/PARALLEL c:\file2.txt c:\file3.txt**

```
-----
C: \FILE2. TXT | C: \FILE3. TXT
-----1-----1-----
$ b=f$length(' p1) | $ a == "1"
$ write $stdout "' 'b' " | $ @tmp2 a
$ exit | $ exit
-----
```

Number of difference sections found: 1  
Number of difference records found: 3

```
DIFFERENCES/PARALLEL
```

C: \FILE2. TXT  
C: \FILE3. TXT

This example is similar to the one above except that the difference section is reported by showing the differences in a side-by-side view.

## 9.9 DIRECTORY Command

The DIRECTORY command provides a list of files or information about a file or group of files.

### Format:

**DIRECTORY [file-name[,...]]**

### Parameters:

#### **file-name**

Specifies one or more files to be listed. The syntax of a file name determines which files will be listed, as follows:

If a file name is omitted, the DIRECTORY command will list all files in the current directory.

If only a device name is specified (e.g., C:), the DIRECTORY command will list all files in the default directory for that device. (**ftpurl**)

### Description:

The DIRECTORY command searches and lists files contained in a directory or folder. Like all XLNT file commands, DIRECTORY provides various search criteria qualifiers which allow you to refine your file search. The DIRECTORY command at its simplest will produce a brief list display across the console window (or output file) of files that must meet the selection criteria. By default, DIRECTORY will list all files (except those with the hidden attribute) in the current directory. DIRECTORY also provides qualifiers which control the formatting of the list: /HEADING and /TRAILING specifically alter the formatting of the list. /NOHEADING produces a single column of files that are fully qualified (in terms of UNC and path specification). This qualifier is particularly suited to reading and simple parsing by a command procedure. However, note that many qualifiers as a result of usage also change the formatting. Specifying /DATE or /SIZE will cause a single file to be listed on each line. Specifying /FULL causes several informational lines to be displayed with the various file characteristics and permissions (if applicable). The /FULL qualifier bears some additional explanation. /FULL means that you want a complete listing of the file's attributes and permissions to the extent that they exist (for example, a FAT volume/file has no permissions).

DIRECTORY supports full UNC specifications including directory and filename wildcarding. To wildcard a directory specification specify \...\ at any position within the directory specification that you want to wildcard from that point. For example, to search for all CPP type files on the entire C: drive specify:

```
$ DIRECTORY C:\...\*.CPP
```

To search for the same files from a specific point downward, specify:

```
$ DIRECTORY C:\SOFTWARE\2.0\...\*.CPP
```

this causes a search from C:\SOFTWARE\2.0 and all subdirectories downward for files that have the CPP type. The same search entered with a UNC specification would appear as follows:

```
$ DIRECTORY \\SERVER4\D$\SOFTWARE\2.0\...\*.CPP
```

If the directory and/or file specification contains any embedded blank spaces the file-specification must be contained in quotation marks. For example:

```
$ DIRECTORY "C:\Program Files\...\*.TXT"
```

The same is true for FTP URL specifications. For example:

```
$ DIRECTORY "FTP://FTP.ADVSYSCON.COM/XLNT/*.*"
```

### Qualifiers:

#### **/ACCESS**

Selects the date the file was last accessed. For use with the /SINCE and /BEFORE qualifiers.

#### **/ATTRIBUTE**

Displays the attributes of each file listed.

**/BEFORE[=time]**

Selects only those files dated prior to the specified time. You can specify an absolute time or a combination time as per the [Date and Time Format](#) rules. TODAY is the default.

## **/CREATE**

Selects the file creation date. For use with the /SINCE and /BEFORE qualifiers.

## **/DATE = (ACCESS | CREATE | WRITE | ALL)**

Displays the file's name and date, according to the specified criterion. *Access* displays the date the file was last accessed. *Create* displays the file's creation date. *Write* (default) displays the date on which the file was last updated.

## **/DIRECTORY**

Displays only directory files in the requested directory.

## **/EXCLUDE=(filename[,...])**

Excludes the selected files from the DIRECTORY command.

## **/FULL**

Displays full information for each file selected. This includes all filenames (the default filename as well as the MS-DOS filename in 8.3 format if applicable), the file size, the file date and the file attributes. For Windows NT systems, this also includes the owner and any security information (such as Access Control Entries) associated with the file or directory.

## **/GRAND\_TOTAL**

Displays only the totals for all files and directories that have been specified.

## **/[NO]HEADING**

Controls whether heading lines consisting of device description and directory specification are displayed. When /NOHEADING is specified, DIRECTORY creates a single column of directory information where each file is listed separately on a single line in fully path qualified form. This qualifier is particularly useful when a command procedure is expected to read and analyze the results for subsequent processing. By default, /HEADING is assumed.

## **/[NO]HIDDEN**

Controls whether files marked with the *Hidden* attribute are displayed. By default, hidden files are not displayed.

## **/OUTPUT[=filename]**

Controls where the output of the command is sent. By default, the display is written to the console window.

## **/OWNER**

Controls whether the owner of the file or directory is displayed (Windows NT only)

## **/PAGE**

Displays the output of the command one page at a time.

## **/ROOT\_DIRECTORY**

Determines whether the specification entered is meant to describe the directory itself. By default, DIRECTORY assumes a directory specification to actually refer to the files beneath the specified directory and not the directory itself.

## **/SEARCH=keyword**

Provides additional search capability. Currently the only keyword available is SIZE. The format is SIZE=(min[,max]). This keyword allows you to restrict DIRECTORY to displaying files that have either a certain minimum size or fall within a range (such as when you specify both the minimum and maximum values). By default, DIRECTORY imposes no size limitation when displaying files.

## **/SECURITY**

Controls whether security information associated with the file or directory is displayed (Windows NT only)

## **/SINCE[=time]**

Selects only those files dated after the specified time. You can specify an absolute time or a combination time as per the [Date and Time Format](#) rules. TODAY is the default.

## **/SIZE**

Displays the file's name and size in bytes.

## **/SORT [= SIZE | DATE | NAME]**

Sorts the output of the DIRECTORY command into collating sequence order based on the keyword specified. NAME is the default keyword if none is specified.

## **/TOTAL**

Displays only the directory name and total number of files.

## **/[NO]TRAILING**

Controls whether trailing summary information lines are displayed.

## **/WRITE (default)**

Selects the date the file was last written to. For use with the /SINCE and /BEFORE qualifiers.

### **Examples:**

#### **\$ DIRECTORY/OUT=DEVICE.LIS C:\...\\*.EXE**

This command searches the C device looking for files with an extension of .EXE. All sub-directories will be examined. All files found in the search are placed in the output file DEVICE.LIS. This file could be later examined in a number of ways, perhaps the most interesting being XLNT itself. An XLNT command procedure could OPEN and parse the file.

#### **\$ DIR/SIZE/DATE**

```
Directory C:\SCRATCH
206_nt.exe          2/29/96 2:32:44 PM      37005
206_nt2.exe         2/29/96 5:39:48 PM      38215
206_nt31.exe        2/29/96 2:32:56 PM      42562
BEN.BAT             8/7/95 8:46:58 AM        0
BETA2BUG.TXT        4/10/95 11:22:44 AM      2478
CHKLIST.MS          11/6/94 3:44:04 PM        27
CNTLFIL.E           2/9/94 3:14:36 PM      16801
CTRLFIL.E           2/8/94 11:37:48 AM      4852
INSTALL.BAT         1/19/95 6:09:08 PM       134
LMS206.SYS          1/20/95 8:37:28 AM      33984
OEMSETUP.INF        5/11/94 9:59:54 AM      20877
README.1ST          4/10/95 10:08:14 AM      3404
TXTSETUP.OEM        1/17/95 12:29:18 PM      436
```

Total of 13 files, 200775 bytes

This command searches the current directory and will display the file specification, file size and creation date for each file found.

#### **\$ DIR/FULL PRODUCT.BAT**

```
Directory C:\SCRATCH
PRODUCT.BAT
File Size : 26 bytes
File Creation Date: 2/21/96 5:04:13 PM
Last Access Date : 2/26/96 12:58:04 PM
Last Write Date : 2/21/96 5:04:13 PM
```

File Attributes: Archive  
Owner: Administrator  
Access Control List:  
Domain Admins Full Control (All)  
Everyone Full Control (All)

Total of 1 files

This command provides a complete directory listing of the file PRODUCT.BAT.

## 9.10 DISABLE

The DISABLE command (and its complement ENABLE) allow you to disable file redirection or registry reflection. These commands are useful on 64-bit systems when running in a 32-bit environment. By default, file system access, for example, “Program Files” is automatically redirected to “Program Files (x86)”. This command prevents that redirection. Likewise, when issuing Registry commands or lexicals on 64-bit system running in a 32-bit environment, Windows, by default, will reflect access to WOW64 key. This command prevents that reflection.

### Format:

**DISABLE** parameter

### Description:

The DISABLE command prevents file redirection or registry reflection when issued on a 64-bit platform within a 32-bit process or environment.

### Parameter:

**FILEREDIRECTION** – used to prevent file system redirection

**REGISTRYREDIRECTION** – used to prevent registry redirection.

## 9.11 DISCONNECT FILE Command

The DISCONNECT FILE command will close a file opened on a server. It should be used only when the file cannot be closed by any other means. This operation does not write any data cached on the client system.

### Format:

**DISCONNECT FILE/qualifiers**

### Description:

The DISCONNECT FILE command is used to force a file to be closed that is opened on a server. Files that are opened locally cannot be closed using this command. The /USER qualifier is useful for disconnecting files for a specific user and the /ALL qualifier is particularly useful when a file is to be closed as a prerequisite for backup.

### Qualifiers:

#### /ALL

Specifies that all open files are to be closed.

#### /ID=number

Specifies the identification number assigned to the file when it was opened. The SHOW SERVER/FILES command will display this number.

#### /[NO]LOG

Controls whether the file name is displayed after being successfully disconnected.

#### /ON=servername

Specifies the name of the server computer on which the operation will execute. If omitted, the local computer will be used.

#### /USER=username

Specifies the name of the user whose opened files are to be closed. This qualifier is mutually exclusive with the /ID qualifier.

Note: If neither /ALL, /ID or /USER qualifiers are specified, an error condition is raised.

### Example:

#### \$ SHOW SERVER/FILES

Open Files on TEST 08-Jul-1997 16:11:56

ID	Path	User Name	Locks
--	----	-----	-----
18	C:\ascii\xlnt\tmp.txt	guest	0

**\$ DISCONNECT FILE/USER=GUEST**

## 9.12 DISCONNECT SESSION Command

The DISCONNECT SESSION command ends a session between a server and a workstation.

### Format:

**DISCONNECT SESSION/qualifiers**

### Qualifiers:

**/ALL**

Specifies that all sessions are disconnected.

**/CLIENT=clientname**

Specifies the name of the computer of the client to disconnect. If omitted, all the sessions of the user specified by the */USER* qualifier will be disconnected.

**[/NO]LOG**

Specifies whether a successful disconnect message is displayed.

**/ON=servername**

Specifies the name of the server computer on which the operation will execute. If omitted, the local computer will be used.

**/USER=username**

Specifies the name of the user whose session is to be terminated. If omitted, all the user sessions from the client computer will be terminated.

### Example:

#### \$ SHOW SESSION

Active Sessions on TEST 08-Jul -1997 16:16:09

Client	Type	User Name	Active	Idle
BIGMACHINE	Windows NT	1381 guest	00 00:04:33	00 00:03:40

**\$ DISCONNECT SESS/USER=GUEST**

## 9.13 DUMP Command

Displays the contents of a file or disk volume in hexadecimal and ASCII format.

### Format:

**DUMP filespec [,...]**

### Parameter:

**file-spec**

Specifies the file or device name to be dumped.

### Description:

The DUMP command is used to obtain a formatted or raw dump of the contents of a file or volume. DUMP is useful when a file may have suffered corruption or cannot be otherwise read using normal or alternative means. DUMP is also useful for looking at internal file system or database structures. By default, DUMP produces an output list containing both longwords and ASCII characters listed horizontally. You can specify other formatting qualifiers to alter the defaults. The /SECTORS qualifier is useful for beginning or limiting a file or volume dump display to either a specific location or a specific duration.

### Qualifiers:

#### **/BYTE**

Formats the dump in bytes. The /BYTE, /LONGWORD, and /WORD qualifiers are mutually exclusive. The default format is composed of longwords.

#### **/DECIMAL**

Dumps the file in decimal radix.

#### **/HEXADECIMAL (default)**

Dumps the file in hexadecimal radix.

#### **/LEFT\_TO\_RIGHT**

Formats and displays selected data from left to right.

#### **/LONGWORD (default)**

Formats the dump in longwords. The /BYTE, /LONGWORD, and /WORD qualifiers are mutually exclusive.

#### **/NUMBER[=n]**

Specifies how byte offsets are assigned to the lines of output. If you specify the /NUMBER qualifier, the byte offsets increase continuously through the dump, beginning with n; if you omit the /NUMBER qualifier, the first byte offset is zero. By default, the byte offset is reset to zero at the beginning of each block or record.

#### **/OUTPUT[=filespec]**

Specifies the output file for the dump. If you do not specify a file specification, the default is the file name of the file being dumped and the file type .DMP. If the /OUTPUT qualifier is not specified, the dump goes to \$STDOUT.

#### **/RIGHT TO LEFT**

Formats and displays selected data from right to left.

#### **/SECTORS[=(option[,...])]**

Dumps the specified blocks one block at a time, which is the default method for all devices except network devices.

Block numbers are specified as integers relative to the beginning of the file. Typically, blocks are numbered beginning with 1. If a disk device is specified, blocks are numbered beginning with zero. Select a range of blocks to be dumped by specifying one of the following options:

**START:n** Specifies the number of the first block to be dumped; the default is the first block.

**END:n** Specifies the number of the last block to be dumped; the default is the end-of-file (EOF) block.

**COUNT:n** Specifies the number of blocks to be dumped. The COUNT option provides an alternative to the END option; you cannot specify both.

If you specify only one option, you can omit the parentheses.

### **/WORD**

Formats the dump in words. The /BYTE, /LONGWORD, and /WORD qualifiers are mutually exclusive.

#### **Example:**

#### **\$ DUMP LOGIN.XCP**

Dump of file LOGIN.XCP on 08/29/96 at 12:20:32

Virtual sector number 1 (00000001), 512 (0200) bytes

22206874	61702072	61762074	65732024	\$ set var path "	000000
69775C3A	633B3533	746E6E69	775C3A63	c:\winnt35;c:\wi	000010
633B3233	6D657473	79735C35	33746E6E	nt35\system32;c	000020
6E69775C	3A633B73	776F646E	69775C3A	:\windows;c:\win	000030
6D5C3A63	3B6D6574	7379735C	73776F64	dows\system;c:\m	000040
66666F73	6D5C3A63	3B656369	66666F73	soffice;c:\msoff	000050
3B5C3A63	3B64726F	776E6977	5C656369	ice\winword;c\;	000060
0A0D22				"..	000070

This example shows a simple use of the DUMP utility. ASCII data is shown on the right and hexadecimal data is shown on the left.

## 9.14 EDIT Command

The EDIT command invokes the standard MS-DOS text editor.

**Format:**

**EDIT file-name**

**Parameter:**

**file-name**

Specifies the name of the file to be edited.

**Note:** *Since EDIT invokes the MS-DOS text editor, it adheres to 8.3 filenames (and does not support the UNC specification).*

Note: If you want XLNT to run an editor of your own choosing, see the [SET PREFERENCES](#) command and its EDITOR qualifier.

## 9.15 ENABLE

The ENABLE command (and its complement DISABLE) allow you to enable file redirection or registry reflection. These commands are useful on 64-bit systems when running in a 32-bit environment. By default, file system access, for example, “Program Files” is automatically redirected to “Program Files (x86)”. This command enables that redirection when disabled. Likewise, when issuing Registry commands or lexicals on 64-bit system running in a 32-bit environment, Windows, by default, will reflect access to WOW64 key. This command enables that reflection.

### Format:

**ENABLE parameter**

### Description:

The ENABLE command enables file redirection or registry reflection when issued on a 64-bit platform within a 32-bit process or environment.

### Parameter:

**FILEREDIRECTION** – used to enable file system redirection

**REGISTRYREDIRECTION** – used to enable registry redirection.

## 9.16 MAIL SEND Command

The MAIL command sends a file to one or more recipients or configures your Windows Messaging Profile. The file can be sent in plain text or RTF (Rich Text Format) format.

### Format:

**MAIL SEND file-name recipient[,...]**

### Parameters:

#### **file-name**

Specifies the name of the file to include as the body of the mail message.

#### **recipient[,...]**

Specifies one or more recipients to send the mail message. A recipient can be a fully qualified internet address or a recipient that has been entered in an address book.

### Description:

The MAIL SEND command provides the ability for a command procedure to send mail or some type of alert to an individual or mailing list. MAIL SEND is not expected to be used interactively as with many of the mail clients available for Windows. MAIL does support the MAPI interfaces and in this mode expects to use an existing profile containing the connection and other user information that would be present in a mail profile. In the MAPI mode, the /PROFILE qualifier is required. When a mail message is sent, in MAPI mode, the message is simply queued for transmission. The attributes of the profile are used by the mail system to determine when the message is actually sent. Further, in MAPI mode, it is important to remember that a send-only connection is not possible. Mail is always sent and received. Received mail is placed in the user's in-box.

### Qualifiers:

#### **/[NO]LOG**

Specifies whether a message is displayed if the command executes successfully.

#### **/PASSWORD=profile-password**

This optional qualifier contains the password for an Exchange Profile.

#### **/PROFILE=profilename**

This qualifier contains the name of the Windows Messaging Profile

#### **/SUBJECT=string**

This optional qualifier provides a Subject line to include with the mail message.

### Example:

```
$ MAIL SEND C:\SAMPLE.TXT support@advsyscon.com -  
$_ /SUBJECT="Sample File"/PROFILE="NTUser"
```

This example shows how to send a file named *sample.txt* to *support@advsyscon.com* with a subject of *Sample File* using the profile *NTUser* that has no password.

## 9.17 MAIL SEND/SMTP Command

The MAIL command sends a file to one or more recipients using the SMTP mail protocol. This command differs from MAIL SEND in that the MAPI protocol (or Microsoft Exchange profile) is not used.

### Format:

**MAIL SEND/SMTP file-name recipient[,...]**

### Parameters:

#### **file-name**

Specifies the name of the file to include as the body of the mail message.

#### **recipient[,...]**

Specifies one or more recipients to send the mail message. A recipient must be a fully qualified internet address.

### Description:

The MAIL SEND/SMTP command provides the ability for a command procedure to send mail or some type of alert to an individual or mailing list. MAIL SEND/SMTP is not expected to be used interactively as with many of the mail clients available for Windows. Unlike the MAPI version of MAIL SEND, SEND/SMTP uses the SMTP protocol and does not require any MAPI profiles. Consequently, SEND/SMTP is perfect for non-interactive procedures or services that need to send mail and reception of mail is not desired. SEND/SMTP will also not cause any new mail to be placed in the user's inbox.

### Qualifiers:

#### **/ATTACHMENT=(file,...)**

If specified, one or more files may be included as attachments to the sent message.

#### **/CC=(cc-list | recipient,...)**

If specified, one or more recipients which will also be sent the mail message OR a text file containing the list of recipients to be cc'd.

#### **/[NO]LOG**

Specifies whether a message is displayed if the command executes successfully.

#### **/PORT=number**

This optional qualifier is used to specify a non-standard TCP port number for SMTP transmission.

#### **/SERVER=server-name**

This qualifier indicates the fully qualified machine name that will receive the SMTP transmission (in other words your SMTP mail server machine).

#### **/SIGNATURE=file-spec**

This optional qualifier may be used to specify a file where signature information will be appended to the transmitted message.

#### **/SUBJECT=string**

This optional qualifier provides a Subject line to include with the mail message.

#### **/USERNAME=user-name**

This qualifier allows you to indicate who is sending the message.

### Example:

```
$ MAIL SEND/SMTP C:\SAMPLE.TXT support@advsyscon.com -
```

**`$_/SUBJECT="Sample File"/SERVER=hera /USERNAME=tester`**

This example shows how to send a file named *sample.txt* to *support@advsyscon.com* with a subject of *Sample File*. The message will be sent to *hera* for mail processing and the sender's name is *tester*.

## 9.18 OPEN Command

The OPEN command opens a file for reading, writing, or both, and assigns a symbol name to the file.

### Format:

**OPEN file-symbol file-name**

### Parameters:

**file-symbol**

Specifies the symbol handle to be assigned to the file. This symbol is created as a global symbol and must not conflict with any other symbol or usage.

**file-name**

Specifies the name of the file being opened for input or output.

### Description:

The OPEN command will open any Windows file for reading, writing or both. After the file is opened, you can invoke the READ or WRITE statement (as appropriate) to read or write to the file. To close an opened file you must use the CLOSE statement. Please note that when a file is opened in this manner it will not be closed unless either you logout of XLNT or use the CLOSE statement. Particular attention should be paid to the /CREATE qualifier. You can indicate whether a file should be opened, always created or created only if the file doesn't exist. The /ERROR qualifier allows you to direct XLNT to transfer control to a specific label (within the current command procedure and level) in the event the OPEN fails. The file-symbol specified represents a file handle that must not be changed or otherwise modified.

### Qualifiers:

**/APPEND**

Opens an existing file for writing and positions to the end of the file. New data is added at the end-of-file.

**/ATTRIBUTE=attributes**

Specifies the attributes to be applied to the file. The values can be any of the following:

- ARCHIVE - file to be archived
- NORMAL - normal file
- HIDDEN - hide file from directory display
- TEMPORARY - temporary work file
- READ\_ONLY - read only file
- SYSTEM - system file

**/CREATE=create\_attribute**

Specifies the creation/open attributes to be applied to the file. One of the following values can be specified:

- CREATE\_ALWAYS - always create this file
- CREATE\_NEW - create only if file doesn't exist
- OPEN\_ALWAYS - always open this file
- OPEN\_EXISTING - only open if file exists

TRUNCATE\_EXISTING - truncate existing file

**/ERROR=label**

Transfers control to the label specified (in a command procedure) if the open operation results in an error.

**/FLAGS=values**

Specifies file-processing options:

WRITE\_THRU - write-through file cache

RANDOM\_ACCESS - file access is random

SEQUENTIAL\_ACCESS - sequential file scan

DELETE\_ON\_CLOSE - file is deleted when all handles are closed

**/READ (default)**

Opens the file for reading.

**/SHARE=(R | W)**

Specifies file sharing options. Default is READ.

**/WRITE**

Opens the file for writing. The following restrictions apply to the /WRITE qualifier:

Use the /WRITE qualifier to create a new file.

Use the /READ qualifier with the /WRITE qualifier to open an existing file.

The /WRITE and /APPEND qualifiers are mutually exclusive.

**Examples:**

**\$ OPEN/ERROR=OPEN\_ERROR FILE1 "C:\XLNT\TEST.DAT"**

This command attempts to open the file C:\XLNT\TEST.DAT. If the open is successful a file handle will be placed in the symbol FILE1. If the open is not successful, then control will be transferred to the label OPEN\_ERROR.

**See also:**

CLOSE INQUIRE READ TYPE WRITE

## 9.19 PRINT Command

The PRINT command sends the contents of a file or group of files to an output device or printer.

### Format:

**PRINT file-name[,...]**

### Parameter:

**file-name[,...]**

Specifies the names of one or more files to be displayed.

### Description:

The PRINT command sends the contents of a file or group of files to an output device or printer. The printer must support the same format data or provide translation software to convert file data to whatever format the printer supports. For example, you shouldn't send ASCII data to a Postscript printer unless the printer or remote printer/system supports inline translation. The printer, specified with the /DEVICE qualifier, can be an LPT device or a UNC specified printer. For example, \\SERVER\POSTPRINTER designates a network printer. You might want to examine the MANAGE Printer Management Commands for more information on setting up and maintaining printers.

### Qualifiers:

#### **/ACCESS**

Selects the date the file was last accessed. For use with the /BEFORE and /SINCE qualifiers.

#### **/BEFORE[=time]**

Selects only those files dated prior to the specified time. You can specify time as absolute time, as a combination of absolute and delta times.

#### **/CREATE (default)**

Selects the file creation date. For use with the /BEFORE and /SINCE qualifiers.

#### **/DEVICE[=printer-name]**

Specifies the printer-name (in UNC style syntax) or device-name (i.e. LPT1: ).

#### **/EXCLUDE=(filename[,...])**

Excludes the specified files from the type operation.

#### **/SINCE[=time]**

Selects only those files dated after the specified time. You can specify time as absolute time, as a combination of absolute and delta times. If the time value is omitted, TODAY is assumed.

#### **/WRITE**

Selects the date the file was last written to. For use with the /BEFORE and /SINCE qualifiers.

### Example:

```
$ PRINT/DEVICE=\\SERVER\ANSI17 c:\sylogin.xcp
```

This command will print the file c:\sylogin.xcp on remote printer \\server\ansi17.

## 9.20 READ Command

The READ command reads data from a specified input file and assigns it to a specified symbol name.

### Format:

**READ file-symbol symbol-name**

### Parameters:

#### **file-symbol**

Specifies the symbol of the input file from which the data is to be read. The symbol was assigned by the OPEN command when the file was opened.

#### **symbol-name**

Specifies the name of the symbol to be equated to the contents of the data that was read. The name must be 1 to 128 alphanumeric characters and must start with an alphabetic letter, an underscore (\_), or a dollar sign (\$).

### Description:

The READ command can be used on any file opened with the OPEN statement to read data either sequentially or randomly. In the sequential mode, the READ statement reads data into a symbol (specified by *symbol-name*) and sets the current position to the next record. You can control the amount of data read using the /COUNT qualifier. By default, READ will read a record up a Return/Line Feed. You can alter this by using either the /COUNT or /DELIMITER qualifiers. Once data has been read, the specification of the /BYTES\_READ qualifier allows you to retrieve the actual bytes read. The /END\_OF\_FILE and /ERROR qualifiers provide the ability to selectively trap either the specific end-of-file error or any general error that might occur. When used in this manner, control is transferred to a label that must be present at the same command level as the READ statement. When READ is used on a console device, you may also specify the /ECHO qualifier. This qualifier, in particular /NOECHO, allows you to disable echo as might be done when requesting a password. You can also specify the /TIMEOUT qualifier to prevent a script hang when a user doesn't enter any data after a specified period of time.

### Qualifiers:

#### **/BYTES\_READ=symbol**

Creates a symbol containing the number of bytes read from the input files.

#### **/COUNT=integer**

Specifies an integer value that indicates the number of bytes that should be read from the file.

#### **/DELIMITER=(value[,value])**

Specifies one or more decimal ASCII values that will be used to delimit the read operation. Default is *Carriage Return, Line Feed* or /DELIMITER=(13,10),

#### **/[NO]ECHO**

By default, the user is able to see keystrokes entered. When /NOECHO is specified, the user's keystrokes are not echoed back. This qualifier is useful when passwords are prompted.

#### **/END\_OF\_FILE=label**

Transfers control to the location specified by the label keyword (in the current command procedure) when the end of the file is reached.

#### **/ERROR=label**

Transfers control to the location specified by the label keyword (in the current command procedure) when a read error occurs.

### **/GLOBAL**

Indicates that the specified symbol name is a *global* symbol.

### **/LOCAL (default)**

Indicates that the specified symbol name is a *local* symbol.

### **/POSITION=position**

Specifies the file position from which to begin the read operation. One of the following values can be specified:

**BEGIN** - read from the beginning of the file

**CURRENT** (default) - read from wherever the file is currently positioned

**END** - read from the end of the file

**OFFSET=n** - read from specified position

### **/PROMPT=[prompt-string]**

Specifies a prompt to be written prior to requesting input. By default the string "Data: " is used.

### **/TIMEOUT=value**

Specifies a timeout value for the read operation, in seconds.

### **/[NO]UPPERCASE**

By default, case is not converted (the opposite of INQUIRE which converts to uppercase by default). To convert data read to uppercase, specify the /UPPERCASE qualifier.

**Note:** *the /COUNT and /DELIMITER qualifiers and the /LOCAL and /GLOBAL qualifiers are mutually exclusive.*

#### **Example:**

```
$ OPEN/ERROR=OPEN_ERROR FILE1 "C:\XLNT\TEST.DAT"  
$READ_LOOP:  
$ READ/END=DONE_READ FILE1 INPUT_RECORD
```

These commands open a file named C:\XLNT\TEST.DAT and place the corresponding file handle into the symbol FILE1. That file-handle symbol is used in the READ command to read the contents of that file. Each record read is placed in the symbol INPUT\_RECORD. If an "End of File" condition is raised, control will be transferred to the symbol DONE\_READ.

#### **See also:**

**CLOSE OPEN WRITE**

## 9.21 RENAME Command

The RENAME command renames an existing file or directory.

### Format:

**RENAME** input-file[,...] output-file

### Parameters:

**input-file[,...]**

Specifies the names of one or more files to be renamed.

**output-file**

Provides the new name to be applied to the input file. The new name must not be the name of an already-existing file. A new file may be placed on a different file system or device. A new directory must be on the same device.

### Description:

The RENAME command is used to change the directory, filename, type or any combination of a file. An important distinction between COPY and RENAME is that the file cannot be renamed to another device or location that differs from the input file(s) original device. Of course, COPY creates a second file while RENAME simply changes the directory and/or filename of an existing file.

### Qualifiers:

**/ACCESS**

Selects the date the file was last accessed. For use with the /SINCE and /BEFORE qualifiers.

**/BEFORE[=time]**

Selects only those files dated prior to the specified time.

**/[NO]CONFIRM**

Controls whether a request is issued before each rename operation to confirm that the operation should be performed on that file.

**/CREATE**

Selects the file creation date. For use with the /SINCE and /BEFORE qualifiers.

**/EXCLUDE=(filename[,...])**

Excludes the specified files from the rename operation.

**/[NO]LOG**

Controls whether the name of each file is displayed as it is renamed.

**/SINCE[=time]**

Selects only those files dated after the specified time.

**/[NO]SUBDIRECTORY**

This qualifier, /SUBDIRECTORY, indicates that if a subdirectory is encountered it will be subject to the RENAME command. This is the default action. Specifying /NOSUBDIRECTORY causes any subdirectories that are encountered to be skipped from the RENAME command.

**/WRITE**

Selects the date the file was last written to. For use with the /SINCE and /BEFORE qualifiers.

**Example:**

```
$ RENAME C:\TEMP\*.EXE C:\TEMP\*.OLD_EXE
```

This example depicts a simple rename operation. Files that have a .EXE extension are renamed to a .OLD\_EXE extension.

## 9.22 SCHEDULE Command

The SCHEDULE command (Windows NT only) provides an XLNT interface to the Windows NT Schedule Service. This command allows you to schedule the execution of XLNT commands and command procedures at specified times in the future. Users who have licensed BQMS will want to examine the SUBMIT command for increased scheduling, execution and security flexibility.

### Format:

**SCHEDULE option /qualifiers**

### Parameters:

#### ADD

This parameter is specified to schedule a new job for execution. Subsequent qualifiers specify the job's particulars.

#### DELETE

This parameter is specified to delete a scheduled job *prior* to its execution.

#### SHOW

This parameter is specified to display the jobs currently scheduled for execution.

### Qualifiers:

#### /ALL

This qualifier is used in conjunction with the DELETE parameter to specify that all currently scheduled jobs are to be deleted. If this qualifier is specified in an interactive XLNT session, a message asking the user to confirm the deletion will be displayed.

#### /COMMAND=command\_string

This qualifier is used in conjunction with the ADD parameter to specify the command to be executed. If the user wishes to execute an XLNT command procedure, this qualifier must supply the full file specification of the command procedure and must be preceded by the "@" sign. In addition, the string must be enclosed within quotation marks. For example, to execute the command procedure *mycommands.xcp* in the *usercmds* directory on device *c:*, specify the qualifier as follows:

```
/command="@c:\usercmds\mycommands.xcp"
```

#### /EVERY=(list)

This qualifier is used in conjunction with the ADD parameter to schedule the job on a repeating basis. Its value is a parenthesized list containing the days of the week (specified as three-character names - MON, TUE, WED, THU, FRI, SAT, SUN) or the days of the month (specified as decimal digits 1 through 31) on which the job is to be scheduled. For example, */EVERY=(MON,WED,FRI)* will schedule the job for execution on every Monday, Wednesday, and Friday of the week. The qualifier */EVERY=(1,15,26)* will schedule the job for execution on the 1<sup>st</sup>, 15<sup>th</sup>, and 26<sup>th</sup> days of every month.

#### /ID=job\_identifier

This qualifier is used in conjunction with the DELETE parameter to specify the identification number of the job to be deleted. This identifier is displayed whenever a new job is added and may also be discerned through use of the SHOW function.

#### /NEXT=(list)

This qualifier is used in conjunction with the ADD parameter to schedule the job in the future. Its value is a parenthesized list containing the next days of the week (specified as three-character names - MON, TUE, WED, THU, FRI, SAT, SUN) or the next days of the month (specified as decimal digits 1 through 31) on which the job is to be scheduled. For example, */NEXT=(TUE,SAT)* will schedule the job for execution on next Tuesday and Saturday. The qualifier */NEXT=(3,18)* will schedule the job for execution on the next 3<sup>rd</sup> and 18<sup>th</sup> of the month.

### **/ON=computername**

This qualifier is used to specify the name of the computer on which the schedule operation is to take place. If omitted, the operation will be performed on the local computer.

### **/TIME=hh:mm**

This qualifier is used in conjunction with the ADD parameter to specify the time of day (specified as hours and minutes of a 24 hour clock) at which the scheduled job is to execute. This is a required qualifier.

Note: The SCHEDULE command provides a direct interface to the Windows NT Schedule Service. As such, it is subject to the same limits and restrictions that apply to the AT command. That is, the job that is scheduled will be run in the account in which the Schedule Service is installed.

### **Examples:**

```
$ SCHEDULE ADD/COMMAND="@C:\ASCIXLNT\DIR.XCP" -/TIME=10:00
```

```
Job 0 successfully scheduled
```

This command schedules a job for execution at 10:00 AM. If the current time is prior to 10:00 then the job will be scheduled today. If the current time is after 10:00 then the job will be scheduled for tomorrow at 10:00.

### **\$ SCHEDULE SHOW**

Status ID	Day	Time	Command Line
0	Today	10:00 AM	C:\ASCIXLNT\xlntcli.exe @C:\ASCIXLNT\DIR.XCP

This command displays the current scheduled jobs that are on queue.

## 9.23 SEARCH Command

Searches one or more files for the specified strings and displays the lines containing those strings.

### Format:

**SEARCH file-name[,...] search-string[,...]**

### Parameters:

#### file-name

Specifies one or more files to be searched. You must specify at least one file name. If you specify more than one file name, separate the file specifications with commas (,).

You can use the asterisk (\*) and the percent sign (%) wildcard characters in the file specification.

#### search-string

Specifies the character string to be located in the specified files. Enclose strings containing lowercase letters, blanks, or other nonalphanumeric characters (including spaces) in quotation marks (" "). (You may use up to 10 search strings)

You can use the /MATCH and /EXACT qualifiers to alter the way that SEARCH matches search strings.

### Description:

The SEARCH command searches through one or more files looking for a specified character string. Each time the string is found, the record is displayed.

### Qualifiers:

#### /ACCESS

Selects the date the file was last accessed. For use with the /SINCE and /BEFORE qualifiers.

#### /BEFORE[=time]

Selects only those files dated prior to the specified time. You can specify time as absolute time, as a combination of absolute and delta times.

#### /[NO]CONFIRM

Controls whether a request is issued before each search operation to confirm that the operation should be performed on that file.

The following responses are valid:

YES	NO	QUIT	TRUE	FALSE	Ctrl/Z
1	0	ALL	<Enter>		

You can use any combination of uppercase and lowercase letters for word responses. Word responses can be abbreviated to one or more letters (for example, T, TR, or TRU for TRUE), but these abbreviations must be unique. Affirmative answers are YES, TRUE, and 1. Negative answers include: NO, FALSE, 0, and pressing the Enter key. Entering QUIT or pressing Ctrl/Z indicates that you want to stop processing the command at that point. When you respond by entering ALL, the command continues to process, but no further prompts are given. If you type a response other than one of those in the list, XLNT issues an error message and redisplay the prompt.

#### /CREATE (default)

Selects the file creation date. For use with the /SINCE and /BEFORE qualifiers.

### **/[NO]EXACT**

Controls whether the SEARCH command matches the search string exactly or treats uppercase and lowercase letters as equivalents. By default, SEARCH ignores case differences in letters.

### **/EXCLUDE=(filespec[,...])**

Excludes the specified files from the search operation. You can include a directory but not a device in the file specification. The asterisk (\*) and the percent sign (%) wildcard characters are allowed in the file specification. However, you cannot use relative version numbers to exclude a specific version. If you specify only one file, you can omit the parentheses.

### **/[NO]HEADING**

Includes file names in the output file and displays a line of 30 asterisks(\*) as a window separator between groups of lines that belong to different files. With the default heading format, file names are printed only when more than one file is specified or when the asterisk (\*) and the percent sign (%) wildcard characters are used.

The /WINDOW qualifier displays a line of 15 asterisks to separate each window within a file.

### **/[NO]HIGHLIGHT**

This will show your search strings in reverse video.

### **/[NO]LOG**

Outputs a message for each file searched. The message includes the file name, the number of records, and the number of matches for each file searched.

### **/MATCH=option**

Interprets and matches multiple search strings in one of the following ways:

- AND** A match occurs only if the record contains all the strings.
- NOR** A match occurs only if the record contains none of the strings.
- NAND** A match occurs only if the record does not contain all of the strings.
- OR** A match occurs if the record contains any of the strings.
- XOR** A match occurs if any of the search strings are in the record but not if all or none of them are in the record.

When only one search string is specified, the OR and AND options produce identical results. Similarly, NOR and NAND produce identical results for a single search string. If you specify none of these options, the default is /MATCH=OR.

### **/[NO]NUMBERS**

Controls whether the source line number is displayed at the left margin of each line in the output.

### **/[NO]OUTPUT[=filespec]**

Controls whether the results of the search are output to a specified file. The output is sent to the current default output device (\$stdout) if you omit the /OUTPUT qualifier or omit the file specification with the qualifier. The

/NOOUTPUT qualifier means that no matching records are output as a result of the SEARCH command.

**/SINCE[=time]**

Selects only those files dated after the specified time. You can specify time as absolute time, as a combination of absolute and delta times. If the time value is omitted, TODAY is assumed.

**/[NO]STATISTICS**

Controls whether the following statistics about the search are displayed:

- Number of files searched
- Number of characters searched
- Number of lines printed (for each string)

**/[NO]WINDOW[=(n1,n2)]**

Specifies the number of lines to be displayed with the search string.

If you specify n1 and n2, the /WINDOW qualifier displays n1 lines above the search string, the search string, and n2 lines below the search string. Either of these numbers can be zero.

If you specify the /WINDOW qualifier without the values n1 and n2, two lines above the search string, the search string, and the two lines below the search string are included in the output.

If you specify the /WINDOW qualifier with a single number (n1), n1 specifies the number of lines to display including the search string. Half the lines precede the matched search string and half follow it. (If n1 is even, one line is added to the lines following the matched search string.)

For example, if you specify /WINDOW=10, nine additional lines are listed along with the line containing the search string. Four lines are listed above the line containing the search string and five lines are listed below it, for a total of 10 lines.

If you specify /WINDOW=0, the file name of each file containing a match (but no records) is included in the output.

If you omit the /WINDOW qualifier, only the line containing a match is displayed.

The maximum values for n1 and n2 are 100.

**/WRITE**

Selects the date the file was last written to. For use with the /SINCE and /BEFORE qualifiers.

**Examples:**

**\$ SEARCH READMEFT4.TXT ADV**

\*\*\*\*\*

C:\VASC\XLNT\READMEFT4.TXT

Copyright © 1995, 1996. Advanced Systems Concepts, Inc.

A mailing list for XLNT named xlnt\_beta@advsyscon.com has been established

Internet: Sales and Distribution: sales@advsyscon.com

Information: info@advsyscon.com

Technical Support: support@advsyscon.com

World-Wide-Web: <http://www.advsyscon.com>

Anonymous FTP: ftp.advsyscon.com

ASCI is a registered trademark of Advanced Systems Concepts.

XLNT is a trademark of Advanced Systems Concepts.

This example searches a text file looking for the subphrase *adv*. The search is made looking for the caseless match of *adv* (if an exact match was needed the /EXACT qualifier would be used).

**\$ SEARCH READMEFT4.TXT/MATCH=AND ADV,SYSTEM**

\*\*\*\*\*

C: \ASCI \XLNT\READMEFT4.TXT

Copyright © 1995, 1996. Advanced Systems Concepts, Inc.

ASCI is a registered trademark of Advanced Systems Concepts.

XLNT is a trademark of Advanced Systems Concepts.

This example further refines the first example and searches for records which contain both *adv* and *system*. Note that the domain name records are dropped out of the search.

## 9.24 TYPE Command

The TYPE command displays the contents of a file or group of files on the current output device.

### Format:

**TYPE file-name[,...]**

### Parameter:

**file-name[,...]**

Specifies the names of one or more files to be displayed.

### Description:

The TYPE command is used to display a printable file on an XLNT console session. Use the /PAGE qualifier to cause TYPE to display the specified file(s) a page at a time.

### Qualifiers:

#### **/ACCESS**

Selects the date the file was last accessed. For use with the /BEFORE and /SINCE qualifiers.

#### **/BEFORE[=time]**

Selects only those files dated prior to the specified time. You can specify time as absolute time, as a combination of absolute and delta times, or as one of the following keywords: TODAY (default), TOMORROW, or YESTERDAY.

#### **/CREATE (default)**

Selects the file creation date. For use with the /BEFORE and /SINCE qualifiers.

#### **/EXCLUDE=(filename[,...])**

Excludes the specified files from the type operation.

#### **/OUTPUT[=filename]**

Specifies where the output of the command is sent.

#### **/PAGE**

Displays the file one page at a time and waits for input before displaying the next page.

#### **/SINCE[=time]**

Selects only those files dated after the specified time. You can specify time as absolute time, as a combination of absolute and delta times. If the time value is omitted, TODAY is assumed.

#### **/WRITE**

Selects the date the file was last written to. For use with the /BEFORE and /SINCE qualifiers.

### Example:

```
$ TYPE \\SERVER\CDISK\TEST\README.TXT
```

This command causes the above file to be displayed on the console window (or more correctly to Standard Out).

## 9.25 WRITE Command

The WRITE command writes the specified data as one stream to an open file specified as a symbol name.

### Format:

**WRITE symbol expression**

### Parameters:

**symbol**

Specifies the symbol assigned to the file by an OPEN command.

**expression**

Specifies data to be written as a single stream to the output file. Can be either a string symbol, a quoted string or a formal expression

### Description:

The WRITE command is used to write data to a file that has been previously opened via the OPEN statement. Data contained in the *expression* parameter is written to the file. You can control how much data is written via the /COUNT qualifier. By default, WRITE will write the entire contents of the *expression* parameter to the file.

### Qualifier:

**/COUNT=value**

Specifies the count, in bytes, of the data to be written to the file.

**/DELIMITER=value**

Specifies the value(s) used to delimit the write.

**/ERROR=label**

Transfers control to the location specified by the *label* keyword (in a command procedure) if an error occurs during the Write operation.

**/POSITION=position**

Specifies the file position from which to begin the write operation. One of the following values can be specified:

**BEGIN** - write to the beginning of the file

**CURRENT** (default) - write to wherever the file is currently positioned

**END** - write to the end of the file

**OFFSET=n** - write from specified position

### Example:

```
$ WRITE $STDOUT "Now is the time for all good...."
```

This command writes a record to a file using an expression specified as the third parameter of the command (in this case, a simple quoted string). Note that the WRITE command is using the built-in file handle \$STDOUT (if run interactively, this would be a console window, however, \$STDOUT can be redirected under XLNT using the RUN/OUT or SPAWN/OUT commands).

### See also:

**CLOSE OPEN READ**

# 10 Distributed File System Commands

The following Distributed File System (Dfs) commands provide the Windows NT administrator with the ability to add, remove and show Dfs junction points. These commands are available for Windows NT systems only.

## 10.1 DFS ADD

This command allows an administrator the ability to add a junction point to an existing Dfs tree.

### Format:

**DFS ADD dfs-entry-path storage-path,...**

### Parameters:

#### **dfs-entry-path**

This parameter represents a new or existing Dfs entry path that is to be used when accessing files. The string must be of the form: \\server\dfsname\path-to-junction-point.

#### **storage-path[,...]**

This parameter (one or more of which can be specified, delimited by commas) represents the name of the server and the share on the server that combine to form a storage-path and is associated with the junction point. The parameter must be of the form: \\server\sharename. When more than one storage-path is specified for a junction point, Dfs treats the additional storage-paths as alternate paths.

### Qualifiers:

#### **/COMMENT=string**

This optional qualifier when used associates a comment with the junction point added.

#### **/LOG**

This qualifier causes the display of a message indicating the operation's success. By default, only errors and warnings are displayed.

### Examples:

```
$ DFS ADD \\HERA\TESTDFS\TEST1 \\ORION\C$, \\ODIN\C$, - \\HERA\C$ /LOG
XLNT-I-DFSADDSUC, \\ORION\C$ successfully added.
XLNT-I-DFSADDSUC, \\ODIN\C$ successfully added.
XLNT-I-DFSADDSUC, \\HERA\C$ successfully added.
```

This command allows the user to add multiple storage paths to either an existing junction point or a new junction point. In this case, the junction point TEST1 is being newly created.

```
$ DFS ADD \\HERA\TESTDFS\TEST1 \\HERA\D$, \\ZEUS\C$, -\\HERA\C$ /LOG
XLNT-I-DFSADDSUC, \\HERA\D$ successfully added.
XLNT-I-DFSADDSUC, \\ZEUS\C$ successfully added.
XLNT-W-STOPATHEXIST, Storage path \\HERA\C$ already exists.
```

Here we see that we can add more storage paths to the existing junction point. Those storage paths that already exist on the Dfs tree will not be added.

## 10.2 DFS REMOVE

This command allows an administrator the ability to remove a junction point from an existing Dfs path.

### Format:

```
DFS REMOVE dfs-entry-path [storage-path,...]
```

### Parameters:

#### **dfs-entry-path**

This parameter represents the existing Dfs entry path that is used when accessing files. The string must be of the form: \\server\dfsname\path-to-junction-point.

#### **storage-path[,...]**

This optional parameter (one or more of which can be specified, delimited by commas) represents the name of the server and the share on the server that combine to form a storage-path and are removed from the junction point. The string must be of the form: \\server\sharename. Each storage-path specified will be removed from the junction point. To remove the entire junction point use the /ALL qualifier.

### Qualifiers:

#### **/ALL**

This qualifier indicates that the entire junction point is to be removed from the specified Dfs tree.

#### **/LOG**

This qualifier causes the display of a message indicating the operation's success. By default, only errors and warnings are displayed.

### Examples:

```
$ DFS REMOVE \\HERA\TESTDFS\TEST1 \\HERA\D$, -\\HERA\C$ /LOG  
XLNT-I-DFSRMVSUC, \\HERA\D$ successfully removed.  
XLNT-I-DFSRMVSUC, \\HERA\C$ successfully removed.
```

This command allows the user to remove a multiple storage paths from a junction point. If no other storage paths exist after the remove is complete, the junction point will be removed.

```
$ DFS REMOVE \\HERA\TESTDFS\TEST1 /LOG  
XLNT-I-DFSRMVSUC, \\HERA\TESTDFS\TEST1 successfully removed.
```

This command allows the user to remove a junction point and all associated storage paths.

## 10.3 DFS SET

This command allows an administrator the ability to change characteristics for a junction point.

### Format:

**DFS SET dfs-entry-path**

### Parameters:

**dfs-entry-path**

This parameter represents the existing Dfs entry path that is used when accessing files. The string must be of the form: \\server\dfsname\path-to-junction-point.

### Qualifiers:

**/COMMENT=string**

This optional qualifier when used associates a new comment with the junction point.

**/LOG**

This qualifier causes the display of a message indicating the operation's success. By default, only errors and warnings are displayed.

### Example:

```
$ DFS SET \\HERA\TESTDFS\TEST1 -  
  /COMMENT="THIS IS THE TEST JUNCTION POINT" /LOG  
XLNT-I-DFSSETSUC, \\HERA\TESTDFS\TEST1 successfully modified.
```

This command changes the comment for the Dfs entry path.

## 10.4 DFS SHOW

This command allows an administrator to display the contents of a Dfs path.

**Format:**

**DFS SHOW server-name | dfs-entry-path**

**Parameters:**

**server-name**

This parameter specifies the server that is publishing the Dfs path. If specified, the complete tree is displayed including all junction points. The syntax for the server is \\server.

**OR**

**dfs-entry-path**

This parameter represents the dfs entry point and specific junction point that is to be displayed. The string must be of the form: \\server\dfsname\path-to-junction-point.

**Note: Only one of the parameters listed above can be displayed.**

**Qualifiers:**

**/GRAPHIC**

When specified displays a tree-like graphic representation of the Dfs root and associated junction points.

**/FULL**

When specified provides full details of the Dfs volume. If /GRAPHIC and /FULL are omitted, a brief display of the Dfs volume(s) is performed.

**/OUTPUT=file-spec**

When specified indicates that output display should be sent to the file specified.

**/PAGE**

When specified the information is displayed a page at a time. The user would then press ENTER to continue to the next page.

**Examples:**

**\$ DFS SHOW \\HERA**

Entry Path	Server	Comment
-----	-----	-----
\\HERA\testdfs	\\HERA\testdfs	
\\HERA\TESTDFS\TEST1	3 Servers	

This command displays a brief listing of the Dfs tree on machine HERA. Three storage paths are associated with junction point \\HERA\TESTDFS\TEST1 and one storage path is associated with \\HERA\testdfs (which is why it is directly listed).

**\$ DFS SHOW \\HERA\TESTDFS\TEST1 /GRAPHIC**

```
--Root Volume
|  |--\\HERA\TESTDFS\TEST1 (Normal)
|  |  |--\\ZEUS\C$ (Online)
|  |  |--\\HERA\D$ (Online)
|  |  |--\\HERA\C$ (Online)
```

This command allows the user to view the Dfs tree.

# 11 Printer Management Commands

The following Printer Management commands provide the Windows NT administrator or Windows 95 user with the ability to add, maintain and delete printer queues and the various print jobs enqueued on those printers.

Please note that Windows NT does require special rights in order to manipulate or otherwise maintain printer queues.

## 11.1 MANAGE ADD DRIVER

This command provides power users with the ability to add a printer driver to a network or local machine.

### Format:

**MANAGE ADD DRIVER version driver-name environment driver-path driver-data driver-config**

### Parameters:

#### version

This parameter indicates the printer's version number.

#### driver-name

This parameter specifies the name of the driver (for example, "QMS 810").

#### environment

This parameter specifies the environment the driver was written for (for example, "Windows NT x86").

#### driver-path

This parameter specifies the filename or full path and filename for the file that contains the printer device driver (for example, "C:\DRIVERS\PSCRIPT.DLL").

#### driver-data

This parameter specifies a filename or full path and filename for the file that contains the printer device driver data (for example, "C:\DRIVERS\QMS810.PPD").

#### driver-config

This parameter specifies a filename or full path and filename for the file that contains the printer device driver's configuration DLL (for example, "C:\DRIVERS\PSCRPTUI.DLL").

### Description:

This command can be used to add a printer driver to a local or remote machine. Typically a printer vendor will provide an installation procedure for adding a driver at the same time a printer is being added. Many of the parameters and qualifiers are heavily dependent on the actual printer driver being added.

### Qualifiers:

#### **/DATA\_TYPE=string**

This qualifier can be specified to indicate the data type of submitted print jobs. The default is RAW.

#### **/DEPENDENT\_FILE=(file-spec,...)**

This qualifier indicates the file(s) that this driver is dependent on.

#### **/HELP\_FILE=file-spec**

This qualifier specifies a filename or full path and filename for the printer device driver's help file.

#### **/[NO]LOG**

This qualifier causes a message to be displayed indicating successful operation. By default, messages are displayed only when an error occurs.

#### **/MONITOR\_NAME=string**

This qualifier indicates a language monitor (for example, "PJL monitor").

#### **/ON=machine-name**

This optional qualifier allows you to direct the execution of this command to another system. By default, commands are executed on the local system that they are issued on.

**Example:**

```
$ MANAGE ADD DRIVER 2 "HP Deskjet 4Si" "Windows NT x86" -  
"C:\WINNT\System32\spool\DRIVERS\W32X86\RASDD.DLL" -  
"C:\WINNT\System32\spool\DRIVERS\W32X86\PCL5EMS.DLL" -  
"C:\WINNT\System32\spool\DRIVERS\W32X86\RASDDUI.DLL" -/HELP= -  
"C:\WINNT\System32\spool\DRIVERS\W32X86\RASDDUI.HLP" -/DEPENDENT_FILE= -  
(C:\WINNT\System32\spool\DRIVERS\W32X86\RASDDUI.DLL) /DATA_TYPE="RAW" /LOG  
XLNT-I-ADDDRIVERSUC, Printer driver installation successful.
```

## 11.2 MANAGE DELETE DRIVER

This command deletes an existing printer driver. Please note that special rights may be required to delete a printer driver, and further that the actual files associated with the driver are not deleted by this command.

### Format:

**MANAGE DELETE DRIVER environment driver-name**

### Parameters:

#### **environment**

This parameter specifies the environment the driver was written for (for example, "Windows x86").

#### **driver-name**

This parameter specifies the name of the driver to be deleted (for example, "QMS 810").

### Qualifiers:

#### **/[NO]LOG**

This qualifier causes a message to be displayed indicating successful operation. By default, messages are displayed only when an error occurs.

#### **/ON=machine-name**

This optional qualifier allows you to direct the execution of this command to another system. By default, commands are executed on the local system that they are issued on.

### Examples:

```
$ MANAGE DELETE DRIVER "GENERIC / TEXT ONLY " -  
"WINDOWS NT X86" /ON=ORION /LOG
```

The specified printer driver is currently in use.  
XLNT-E-DELDRI VERFAIL, Printer driver deletion failed.

This command allows the user to delete a printer driver from the local machine or a remote machine. The printer driver must be a valid printer driver name that currently exists on the machine and the driver must not be used by any installed printer.

## 11.3 MANAGE SHOW DRIVER

This command displays characteristics of all printer drivers on the local or networked machine.

### Format:

**MANAGE SHOW DRIVER**

### Qualifiers:

#### **/FULL**

This qualifier displays complete information about the printer drivers. By default, brief information is displayed.

#### **/ON=machine-name**

This optional qualifier allows you to direct the execution of this command to another system. By default, commands are executed on the local system they are issued.

#### **/OUTPUT=file-spec**

This optional qualifier when specified directs the command output to the specified file.

#### **/PAGE**

This qualifier provides the interactive user with the ability to read the information provided a page at a time.

### Example:

#### **\$ MANAGE SHOW DRIVER /FULL**

```
Name : HP DeskJet 600
Version : 2
Path : C:\WINNT\System32\spool\DRIVERS\W32X86\2\RASDD.DLL
Environment : Windows NT x86
Data File : C:\WINNT\System32\spool\DRIVERS\W32X86\2\HPDSKJET.DLL
Config File : C:\WINNT\System32\spool\DRIVERS\W32X86\2\RASDDUI.DLL
Help File : C:\WINNT\System32\spool\DRIVERS\W32X86\2\RASDDUI.HLP
Dependent Files : C:\WINNT\System32\spool\DRIVERS\W32X86\2\HPDSKJET.DLL
Monitor File :

Name : Generic / Text Only
Version : 2
Path : C:\WINNT\System32\spool\DRIVERS\W32X86\2\RASDD.DLL
Environment : Windows NT x86
Data File : C:\WINNT\System32\spool\DRIVERS\W32X86\2\TXTONLY.DLL
Config File : C:\WINNT\System32\spool\DRIVERS\W32X86\2\RASDDUI.DLL
Help File : C:\WINNT\System32\spool\DRIVERS\W32X86\2\RASDDUI.HLP
Dependent Files : C:\WINNT\System32\spool\DRIVERS\W32X86\2\TXTONLY.DLL
Monitor File :
```

This command allows the user to show the currently installed printer drivers on the local machine or on a remote machine.

#### **\$ MANAGE SHOW DRIVER /ON=ORION**

Driver Name	Version
Generic / Text Only	2
Digital PrintServer 17 v48.3	2
Digital PrintServer 17 12mb/L2	2
Canon Bubble-Jet BJC-4000	2
Acrobat PDFWriter	2
Acrobat Distiller 3.01	2
Digital PrintServer 17 12mb/L2	1

This command provides a brief display of the drivers that are currently loaded on system ORION.

## 11.4 MANAGE SHOW MONITOR

This command provides users with the ability to show print monitors.

**Format:**

**MANAGE SHOW MONITOR [server-name]**

**Parameter:**

**server-name**

This optional parameter represents the desired machine name that hosts the print monitor(s) you would like displayed.

**Qualifiers:**

**/OUTPUT=file-spec**

This optional qualifier when specified directs the command output to the specified file.

**/PAGE**

This qualifier provides the interactive user with the ability to read the information provided a page at a time.

**Example:**

```
$ MANAGE SHOW MONITOR \\ORION
```

```
Monitor Name  
-----
```

```
LPR Port  
Local Port
```

This command allows the user to show the print monitors that are installed on the local machine or a remote machine.

# 11.5 MANAGE SHOW PORT

This command provides users with the ability to show printer ports.

**Format:**

**MANAGE SHOW PORT [server-name]**

**Parameter:**

**server-name**

This parameter represents the desired machine name that hosts the printer port(s) you would like displayed. If omitted, the local machine is used.

**Qualifiers:**

**/FULL**

This qualifier directs the command to present a complete display of the port's characteristic. By default, only a brief display is presented.

**/OUTPUT=file-spec**

This optional qualifier when specified directs the command output to the specified file.

**/PAGE**

This qualifier provides the interactive user with the ability to read the information provided a page at a time.

**Example:**

```
$ MANAGE SHOW PORT \\ORION
```

Port Name	Port Type
-----	-----
LPT1:	Write
LPT2:	Write
LPT3:	Write
COM1:	Write
COM2:	Write
COM3:	Write
COM4:	Write
FILE:	Write

This command allows the user to display the ports that are installed on the local machine or on a remote machine.

```
$ MANAGE SHOW PORT /FULL
```

```
Port Name : LPT1:  
Monitor Name : Local Monitor  
Description : Local Port  
Port Type : Write
```

```
Port Name : COM1:  
Monitor Name : Local Monitor  
Description : Local Port  
Port Type : Write
```

```
Port Name : FILE:  
Monitor Name : Local Monitor  
Description : Local Port  
Port Type : Write
```

```
Port Name : \\MARS\LITESHOW
Monitor Name : LAN Manager Print Monitor
Description : LAN Manager Printer Port
Port Type : Write
```

This command produces a full description of the ports on the local machine.

## 11.6 MANAGE ADD PRINTER

This command provides power users with the ability to add a printer to a network or local machine.

### Format:

**MANAGE ADD PRINTER printer-name port-name driver-name**

### Parameters:

#### printer-name

This parameter specifies the unique name of the printer you would like to add. Please enter the name in quotes if you would like to either preserve the case of the string or enter embedded spaces within the printer name.

#### port-name

This parameter designates the port(s) used to transmit data to the printer. If a printer is connected to more than one port, the names of each port must be separated by commas (for example, "LPT1:;LPT2:").

#### driver-name

This parameter designates the name of the print driver that is to be used to support this printer.

### Description:

The MANAGE ADD PRINTER command is used to add a printer to a local or remote machine. Most characteristics of the printer can also be specified when the printer is added or may be changed at a later time through the MANAGE SET PRINTER command. Printer permissions may be set through either the /ACCOUNT qualifier when the printer is added and/or the SET PERMISSIONS command. The newly added printer can be established as the default printer through the /ATTRIBUTE=DEFAULT qualifier.

### Qualifiers:

#### **/ACCOUNT=(account[:perm],...)**

This qualifier provides the ability to associate permissions with a printer. One or more permissions may be specified delimited by commas. The account portion may be specified as *machine\accountname* or *domain\accountname* or just simply *accountname*. When used with /REVOKE only account(s) are specified. The account portion is delineated by a colon followed by the permission to be assigned to the account. Valid permissions are: FULL (or ALL), MANAGE, PRINT and NONE.

#### **/ATTRIBUTES=(keyword,...)**

This qualifier allows the specification of various attributes that are to be associated with the printer. More than one keyword can be specified by separating each keyword with a comma.

<b>DEFAULT</b>	designates the printer as the default printer for windows use.
<b>DEVQ</b>	special printer handling
<b>DIRECT</b>	print directly to the printer
<b>DO_COMPLETE_FIRST</b>	print complete jobs first
<b>KEEPPRINTEDJOBS</b>	keep documents after they have printed
<b>QUEUED</b>	print spooled jobs first

#### **/[NO]COLLATE**

This qualifier determines whether collating of multiple copies is to be performed. By default, collating is not enabled.

#### **/[NO]COLOR**

This qualifier determines whether the printer supports color. By default, monochrome is enabled.

#### **/COMMENT=string**

This qualifier allows a comment to be entered and associated with the printer. The default is no comment.

**/DATA\_TYPE=string**

This qualifier can be specified to indicate the data type of submitted print jobs. The default is RAW.

**/[NO]DUPLEX=(HORIZONTAL | VERTICAL)**

This qualifier determines whether the printer supports simplex or duplex printing. If duplex is specified then horizontal or vertical double-side printing can be further specified. By default, simplex printing is enabled.

**/FORM\_NAME=string (Windows NT Only)**

This qualifier can be used to associate a known form name with this printer.

**/LOCATION=string**

This qualifier allows you to associate a location comment with the printer.

**/[NO]LOG**

This qualifier causes a message to be displayed indicating successful operation. By default, messages are displayed only when an error occurs.

**/ON=machine-name**

This optional qualifier allows you to direct the execution of this command to another system. By default, commands are executed on the local system that they are issued on.

**/ORIENTATION=(PORTRAIT | LANDSCAPE)**

This qualifier is used to set the new printer's orientation. By default, the setting is Portrait mode. Portrait mode causes the printer to print across the width portion of the page. You may specify Landscape to print across the length portion of the page.

**/PAPER\_SIZE=(LETTER | LEGAL | A4 | CSHEET | DSHEET | ESHEET | LETTERSMALL | TABLOID | LEDGER | STATEMENT | EXECUTIVE | A3 | A4SMALL | A5 | B4 | B5 | FOLIO | QUARTO | 10X14 | 11X17 | NOTE | ENV\_9 | ENV\_10 | ENV\_11 | ENV\_12 | ENV\_14 | ENV\_DL | ENV\_C3 | ENV\_C4 | ENV\_C5 | ENV\_C6 | ENV\_C65 | ENV\_B4 | ENV\_B5 | ENV\_B6 | ENV\_ITALY | ENV\_MONARCH | ENV\_PERSONAL | FANFOLD\_US | FANFOLD\_STD\_GERMAN | FANFOLD\_LGL\_GERMAN)**

This qualifier determines the size of the paper that is loaded in the printer. The default is LETTER.

**/PARAMETERS=string**

For Advanced Users Only: This qualifiers allows you to specify various printer specific parameters. The default is no parameters.

**/PRIORITY=number**

This qualifier sets the default print job priority for this printer. Default is one (1).

**/PROCESSOR=string**

This qualifier sets the print processor to be associated with this printer. The default is WINPRINT.

**/QUALITY=(HIGH | MEDIUM | LOW | DRAFT | nnn)**

This qualifier determines the quality of the printed output. The *nnn* parameter indicates a numeric DPI setting (for example, 300 for 300 dpi). The default is MEDIUM.

**/REVOKE**

This qualifier when coupled with /ACCOUNT causes each named account to have its permissions revoked (deleted) for the named printer.

**/SEPARATOR=file-spec**

This qualifier allows you to specify a file which is to be used as a separator page. By default there is no separator page.

### **/[NO]SHARE=share-name**

This qualifier, when specified, indicates that you want the added printer to be a shared resource. The printer's *share-name* will become available to other machines of the workgroup or domain. By default, the printer is not shared.

### **/[NO]START\_TIME=hh:mm**

This qualifier indicates the time at which this printer will start printing. The time entered must be in 24-hour format. By default, the printer is not time limited.

### **/TAKE\_OWNERSHIP**

This qualifier when specified indicates that the user issuing the command is to now take ownership of the printer. This qualifier requires either a valid permission allowing the operation or a right indicating the operation can be performed.

### **/[NO]UNTIL\_TIME=hh:mm**

This qualifier indicates the latest time at which this printer will print a job. The time entered must be in 24-hour format. By default, the printer is not time limited.

### **Examples:**

```
$ MANAGE ADD PRINTER "Test Printer 1" "lpt1:" -  
  "generic / text only" /LOG  
XLNT-I-ADDPNSUC, Test Printer 1 successfully added.
```

This command allows the user to add a printer either locally or remotely.

```
$ MANAGE ADD PRINTER "Test Printer 2" "lpt1:" -  
  "HP Deskjet 600" /ON=ORION -  
  /SHARE="Printer 2 ShareName" -  
  ATTRIBUTES=(Default,Queued) -  
  /ORIENTATION=LANDSCAPE /PAPER_SIZE=A5 /LOG  
XLNT-I-ADDPNSUC, Printer 2 successfully added.
```

Here is an example of adding a printer to a remote machine. The driver "HP Deskjet 600" must exist on the machine on which the printer is to be installed. MANAGE SHOW DRIVER can be used to determine what drivers are installed and ADD DRIVER can be used to add the appropriate printer driver to the target system.

```
$ MANAGE ADD PRINTER "XLNT PRINTER" "LPT1:,LPT2:" "GENERIC / TEXT ONLY" /ORIENTATION =  
LANDSCAPE /QUALITY=LOW /ACCOUNT=(EVERYONE:FULL,"CREATOR OWNER:FULL") /LOG  
XLNT-I-ADDPNSUC, XLNT Printer successfully added.
```

This command creates a new printer queue and modifies the security permissions to allow "EVERYONE" and "CREATOR OWNER" full permissions to the printer.

## 11.7 MANAGE CONNECT PRINTER

This command will add a printer connection for the current user.

**Format:**

**MANAGE CONNECT PRINTER printer-name**

**Parameters:**

**printer-name**

This parameter represents a known printer on the network. Please specify UNC style syntax for accessing the printer. For example, \\server1\printer.

**Description:**

This command is typically used within a login script or interactively to connect to a known network printer and make it available for use.

**Qualifier:**

**/[NO]LOG**

This qualifier causes a message to be displayed indicating successful operation. By default, messages are displayed only when an error occurs.

**Example:**

**\$ MANAGE CONNECT PRINTER "\\ORION\TEST PRINTER 2" /LOG**  
XLNT-I -ADDPRNCONSUC, \\ORION\TEST PRINTER 2 successfully connected.

This command allows the user to connect to remote printers. By using the MANAGE SHOW PRINTER command, the user will be able to identify those printers that are remotely accessible.

## 11.8 MANAGE DELETE PRINTER

This command deletes an existing printer.

### Format:

**MANAGE DELETE PRINTER printer-name**

### Parameters:

**printer-name**

This parameter represents a known printer on the system which will be deleted. You may specify a known printer or a UNC style printer name. Please note that Administrator level rights may be required to delete the printer.

### Qualifiers:

**/[NO]LOG**

This qualifier causes a message to be displayed indicating successful operation. By default, messages are displayed only when an error occurs.

### Examples:

**\$ MANAGE DELETE PRINTER "\\ORION\TEST PRINTER 2" /LOG**

XLNT-I-DELPRNSUC, \\ORION\TEST PRINTER 2 successfully deleted.

This command allows the user to delete a printer queue either remotely or locally.

**\$ MANAGE DELETE PRINTER "TEST PRINTER 1" /LOG**

XLNT-I-DELPRNSUC, TEST PRINTER 1 successfully deleted.

**\$ MANAGE DELETE PRINTER "TEST PRINTER 4" /LOG**

XLNT-E-OPENPRINTERFAIL, Failed to open printer TEST PRINTER 4.

This command deletes a local printer queue. The second DELETE PRINTER command displays an error when an attempt to delete a non-existent queue is performed.

## 11.9 MANAGE DISCONNECT PRINTER

This command disconnects an existing printer connection for the current user.

**Format:**

**MANAGE DISCONNECT PRINTER printer-name**

**Parameters:**

**printer-name**

This parameter represents an existing printer name which was previously connected for this user.

**Qualifiers:**

**/[NO]LOG**

This qualifier causes a message to be displayed indicating successful operation. By default, messages are displayed only when an error occurs.

**Example:**

**\$ MANAGE DISCONNECT PRINTER "\\ORION\Test Printer 2" /LOG**

XLNT-I-DELPRNCONSUC, \\ORION\XLNT PRINTER 2 successfully disconnected.

This command allows the user to disconnect remote connections to printers.

## 11.10 MANAGE SET JOB

This command sets or changes the execution state of a current or queued print job.

### Format:

**MANAGE SET JOB printer-name**

### Parameters:

**printer-name**

This parameter represents a known printer on the system on which the selected job(s) will be changed. You may specify a known printer or a UNC style printer name.

### Description:

This command is set to modify the run-time status of an enqueued print job. You may CANCEL, PAUSE, RESUME or RESTART either the selected print job(s) or all print jobs. Print Job selection includes both the print job id number as well as the User who entered the print job.

### Qualifiers:

**/ALL**

This qualifier indicates that all jobs that are queued for printing are eligible for the selected operation (i.e. CANCEL).

**/CANCEL**

This qualifier indicates that the selected job(s) are to be canceled.

**/JOBID=(id,...)**

This qualifier indicates that the job-id's specified are eligible for the selected operation. You can specify multiple job-id's separated by commas and/or a range of job-id's separated by a hyphen (for example, /JOBID=(1,3,5-7) would denote jobs 1,3 and 5 through 7).

**/[NO]LOG**

This qualifier causes a message to be displayed indicating successful operation. By default, messages are displayed only when an error occurs.

**/PAUSE**

This qualifier indicates that the selected job(s) are to be paused.

**/RESTART**

This qualifier indicates that the selected job(s) are to be restarted.

**/RESUME**

This qualifier indicates that the selected job(s) are to be resumed.

**/USER=(username,...)**

This qualifier indicates that the print jobs submitted by specified user names are eligible for the selected operation.

### Examples:

```
$ MANAGE SET JOB "\\ORION\TEST PRINTER 2" /JOBID=(11,12,13-19,21-24) /RESUME /LOG
```

```
XLNT-E-SETJOBFAIL, Error modifying settings for job #11.
```

```
XLNT-I-SETJOBSUC, Job #12 successfully modified.
```

```
XLNT-I-SETJOBSUC, Job #13 successfully modified.
```

```
XLNT-I-SETJOBSUC, Job #14 successfully modified.
```

```
XLNT-I-SETJOBSUC, Job #15 successfully modified.
```

XLNT-I-SETJOBSUC, Job #16 successfully modified.  
XLNT-I-SETJOBSUC, Job #17 successfully modified.  
XLNT-I-SETJOBSUC, Job #18 successfully modified.  
XLNT-I-SETJOBSUC, Job #19 successfully modified.  
XLNT-I-SETJOBSUC, Job #20 successfully modified.  
XLNT-I-SETJOBSUC, Job #21 successfully modified.  
XLNT-E-SETJOBFAIL, Error modifying settings for job #22.  
XLNT-E-SETJOBFAIL, Error modifying settings for job #23.  
XLNT-E-SETJOBFAIL, Error modifying settings for job #24.

## 11.11 MANAGE SET PRINTER

This command changes the characteristics of the selected printer. You must have sufficient rights or permissions to change printer settings permanently.

### Format:

**MANAGE SET PRINTER printer-name**

### Parameters:

**printer-name**

This parameter represents a known printer on the system. You may specify a known printer or a UNC style printer name.

### Description:

The MANAGE SET PRINTER command allows an Administrator to change various printer characteristics on a local or remote printer. Permissions or ownership of the printer may be changed through the /ACCOUNT and /TAKE\_OWNERSHIP qualifiers. The /ATTRIBUTES=DEFAULT qualifier allows a printer to be set as the default printer.

### Qualifiers:

**/ACCOUNT=(account:perm,...)**

This qualifier provides the ability to associate permissions with a printer. One or more permissions may be specified delimited by commas. The account portion may be specified as *machine\accountname* or *domain\accountname* or just simply *accountname*. The account portion is delineated by a colon followed by the permission to be assigned to the account. Valid permissions are: FULL (or ALL), MANAGE, PRINT and NONE.

**/ATTRIBUTES=(keyword,...)**

This qualifier allows the specification of various attributes that are to be associated with the printer. More than one keyword can be specified by separating each keyword with a comma.

<b>DEFAULT</b>	designates the printer as the default printer for windows use.
<b>DEVQ</b>	special printer handling
<b>DIRECT</b>	print directly to the printer
<b>DO_COMPLETE_FIRST</b>	print complete jobs first
<b>KEEPPRINTEDJOBS</b>	keep documents after they have printed
<b>QUEUED</b>	print spooled jobs first

**/[NO]COLLATE**

This qualifier determines whether collating of multiple copies is to be performed.

**/[NO]COLOR**

This qualifier determines whether the printer supports color.

**/COMMENT=string**

This qualifier allows a comment to be entered and associated with the printer.

**/DATA\_TYPE=string**

This qualifier can be specified to indicate the data type of submitted print jobs.

**/DRIVER\_NAME=new-driver-name**

This qualifier can be specified to indicate a new driver that is associated with this printer.

**/[NO]DUPLEX=(HORIZONTAL | VERTICAL)**

This qualifier determines whether the printer supports simplex or duplex printing. If duplex is specified then horizontal or vertical double-side printing can be further specified.

**/FORM\_NAME=string (Windows NT Only)**

This qualifier can be used to associate a known form name with this printer.

**/LOCATION=string**

This qualifier allows you to associate a location comment with the printer.

**/[NO]LOG**

This qualifier causes a message to be displayed indicating successful operation. By default, messages are displayed only when an error occurs.

**/ORIENTATION=(PORTRAIT | LANDSCAPE)**

This qualifier is used to set the printer's orientation. Portrait mode causes the printer to print across the width portion of the page. You may specify Landscape to print across the length portion of the page.

**/PAPER\_SIZE=(LETTER | LEGAL | A4 | CSHEET | DSHEET | ESHEET | LETTERSMALL | TABLOID | LEDGER | STATEMENT | EXECUTIVE | A3 | A4SMALL | A5 | B4 | B5 | FOLIO | QUARTO | 10X14 | 11X17 | NOTE | ENV\_9 | ENV\_10 | ENV\_11 | ENV\_12 | ENV\_14 | ENV\_DL | ENV\_C3 | ENV\_C4 | ENV\_C5 | ENV\_C6 | ENV\_C65 | ENV\_B4 | ENV\_B5 | ENV\_B6 | ENV\_ITALY | ENV\_MONARCH | ENV\_PERSONAL | FANFOLD\_US | FANFOLD\_STD\_GERMAN | FANFOLD\_LGL\_GERMAN)**

This qualifier sets the size of the paper currently loaded.

**/PARAMETERS=string**

For Advanced Users Only: This qualifiers allows you to specify various printer specific parameters.

**/PAUSE**

This qualifier will set the printer state to pause.

**/PORT\_NAME=new-port-assignment**

This qualifier is used to change the printer's current port assignment.

**/PRINTER\_NAME=new-printer-name**

This qualifier is used to change the printer's name and establish a new printer name.

**/PRIORITY=number**

This qualifier sets the default print job priority for this printer.

**/PROCESSOR=string**

This qualifier sets the print processor to be associated with this printer.

**/PURGE**

This qualifier will delete all enqueued print jobs.

**/QUALITY=(HIGH | MEDIUM | LOW | DRAFT | nnn)**

This qualifier determines the quality of the printed output. The *nnn* parameter indicates a numeric DPI setting (for example, 300 for 300 dpi).

**/RESUME**

This qualifier will resume a paused printer.

**/REVOKE**

This qualifier when coupled with /ACCOUNT causes each named account to have its permissions revoked (deleted)

for the named printer.

**/SEPARATOR=file-spec**

This qualifier allows you to specify a file which is to be used as a separator page.

**/[NO]SHARE=share-name**

This qualifier, when specified, indicates that you want the added printer to be a shared resource. The printer's *share-name* will become available to other machines of the workgroup or domain. To stop printer sharing, specify /NOSHARE.

**/[NO]START\_TIME=hh:mm**

This qualifier indicates the time at which this printer will start printing. The time entered must be in 24-hour format. Specifying either /NOSTART or /NUNTIL\_TIME qualifier will effectively remove printer time restrictions.

**/TAKE\_OWNERSHIP**

This qualifier when specified indicates that the user issuing the command is to now take ownership of the printer. This qualifier requires either a valid permission allowing the operation or a right indicating the operation can be performed.

**/[NO]UNTIL\_TIME=hh:mm**

This qualifier indicates the latest time at which this printer will print a job. The time entered must be in 24-hour format. Specifying either /NOSTART or /NUNTIL\_TIME qualifier will effectively remove printer time restrictions.

**Examples:**

```
$ MANAGE SET PRINTER "\\ORION\TEST PRINTER 2" /COMMENT="Test Printer 2 Comment"  
/START_TIME=23:59 /UNTIL_TIME=5:00 /LOG
```

XLNT-I-SETPRNSUC, Settings for \\ORION\TEST PRINTER 2 successfully modified.

This command allows the user to set the execution status of a specific job or a set of jobs as specified by the appropriate qualifiers.

```
$ MANAGE SET PRINTER "\\ORION\TEST PRINTER 2" /PAUSE /LOG
```

XLNT-I-SETPRNSUC, Settings for \\ORION\TEST PRINTER 2 successfully modified.

This command allows the user to change the execution status of a specific printer as well as the characteristics associated with that printer.

```
$ MANAGE SET PRINTER "XLNT PRINTER" -/ACCOUNT=(EVERYONE) /REVOKE /LOG
```

XLNT-I-SETPRNSUC, Settings for XLNT PRINTER successfully modified.

This command changes the security permissions, for the known printer queue "xlnt printer", to revoke access to the printer for the account "EVERYONE."

## 11.12 MANAGE SHOW PRINTER

This command displays characteristics and permissions of a specific printer or all printers.

### Format:

**MANAGE SHOW PRINTER [printer-name]**

### Parameters:

**printer-name**

This optional parameter specifies the name of an existing printer. You may specify a known printer or a UNC style printer name.

### Qualifiers:

**/ALL**

This qualifier indicates that all printers in the domain should be displayed.

**/CONNECTIONS (Windows NT Only)**

This qualifier indicates all printers to which the user has made a previous connection should be displayed.

**/DEFAULT**

This qualifier displays information about the current default printer.

**/DOMAIN=domain-name**

This qualifier, when specified, displays network printers and printservers for the domain name specified. A list of valid domain names can be retrieved using the /PROVIDER qualifier.

**/FULL**

This qualifier displays complete information about the selected printers and/or jobs. On Windows NT systems, permissions associated with the printer are displayed as well. By default, brief information is displayed.

**/JOBS**

This qualifier displays current jobs on the selected printer.

**/LOCAL**

This qualifier displays all locally installed printers.

**/NETWORK (Windows NT Only)**

This qualifier displays network printers in the current domain.

**/ON=machine-name**

This optional qualifier allows you to direct the execution of this command to another system. By default, commands are executed on the local system they are issued.

**/OUTPUT=file-spec**

This optional qualifier when specified directs the command output to the specified file.

**/PAGE**

This qualifier provides the interactive user with the ability to read the information provided a page at a time.

**/PROVIDER[=string]**

This qualifier, specified by itself, will display all print providers. If a qualifier value is specified then domains that use this particular provider will be displayed.

## **/REMOTE (Windows NT Only)**

This qualifier displays network printers and print servers in the current domain.

## **/SHARED**

This qualifier displays printers that have the *shared* attribute. This qualifier cannot be used alone and must be combined with another qualifier.

### **Examples:**

#### **\$ MANAGE SHOW PRINTER**

Printer Name	Jobs	Status
Test Printer 3	0	
Test Printer 1	0	

This command allows the user to view printers locally or remotely. By default the MANAGE SHOW PRINTER command displays a brief listing of the locally installed printer queues.

#### **\$ MANAGE SHOW PRINTER /FULL /DEFAULT**

```
Server Name : \\NTSRV1
Printer Name : \\NTSRV1\Digital PrintServer 17 v48.3
Port Name : SULLU.ADVSYSCON.COM:LPS17_ARIES
Share Name : Digital PrintServer 17 v48.3
Driver Name : Digital PrintServer 17 v48.3
Comment :
Location :
Job Count : 0
Data Type : RAW
Average PPM : 0
Separator File :
Print Processor : winprint
Parameters :
Default Priority : 1
Attributes : Shared
Paper Orientation : Portrait
Print Quality : 300 DPI
Form Name : Letter
Paper Size : Letter, 8 1/2- by 11-inches
Start Time : No Time Restriction
Until Time : No Time Restriction
Status :
Permissions :
  Owner : Administrators
  Name           Access Type
  ----           -
  CREATOR OWNER  Manage Documents
  Everyone        Print
  Administrators  Full Control
  Power Users     Full Control
```

Here we can use this command to view the complete details of a specific printer queue and its permissions.

#### **\$ MANAGE SHOW PRINTER /ON=ORION /SHARED**

Printer Name	Documents	Status
\\ORION\Test Printer 2	10	Paused
\\ORION\Canon Bubble-Jet BJC-4000	9	Paused

You can use the SHOW command to view SHARED printers on a remote machine.

#### **\$ MANAGE SHOW PRINTER /PROVIDER**

```
Windows NT Local Print Provider
NetWare or Compatible Network
Windows NT Remote Printers
```

The above command allows you to view the Print Providers that are available.

# 12 Security Commands

The SECURITY commands provide the Windows NT administrator the ability to control and maintain Windows NT User, Group and Policy level security.

## 12.1 SECURITY CREATE

The SECURITY CREATE command allows a Windows NT Administrator to create users and/or groups for a Workstation or Domain within the Windows NT security system. Please note that the qualifiers which follow may contain a designation such as (User) or (Group). That designation means that the qualifier is only valid when processing that entity. If no such designation is made, the qualifier is available for use without restriction.

### Format:

**SECURITY CREATE/Qualifiers entity-type entity**

### Parameters:

**entity-type**

represents one of the following: GROUP or USER.

**entity**

this parameter specifies either a Groupname (when *entity-type* is GROUP) or a Username (when *entity-type* is USER). The entity must be unique; in other words, it cannot already exist in the target machine's security database.

### Description:

This command is used to create new user and/or group accounts on a Windows NT system. The user will normally be an Administrator or have administrative rights. This command provides support for all of the security characteristics that are publicly available, however, the /SCRIPT\_PATH qualifier provides support for login scripts. You are encouraged to read the section on "XLNT and Login Scripting".

### Qualifiers:

#### **/[NO]ACTIVE**

(User) This qualifier indicates whether the user account should be considered active (enabled) or inactive (disabled) for security purposes. By default, the user account is considered active.

#### **/[NO]CHANGE\_PASSWORD**

(User) This qualifier indicates whether the user can change the password for the account. By default the user can change the password for his account.

#### **/CURRENT\_PASSWORD[=string]**

(User) This qualifier provides you with the ability to specify a current password with the command line instead of being prompted for the password, if required. ASCII recommends that you be prompted for the password since your response will not be echoed. This functionality is typically used during password changes by a non-privileged user.

#### **/COMMENT=string**

This optional qualifier provides a descriptive comment about the user.

#### **/COUNTRY\_CODE=nnn**

(User) Uses the operating system country code to determine language dependent help files and other language dependent facilities. By default a country code of zero (0) means the native country associated with the system.

**/DOMAIN[=domain]**

This qualifier when specified indicates that security information is to be targeted to the logon domain. If the /DOMAIN qualifier is not specified then security information is targeted to the workstation. If the domain value is not specified then the current domain is used.

**/[NO]EXPIRES=date-time**

(User) This qualifier allows the administrator to set the expiration date (if any) for this account. If /EXPIRES is specified, the date-time is specified as a date in absolute time format (i.e. DD-MMM-YYYY). This date is used as the expiration date for the account. /NOEXPIRES means that no expiration date is associated with the account (the default).

**/GLOBAL[=(group-name,...)]**

(User) This qualifier is used to add the user to one or more global groups. The group names specified must be global groups that already exist.

(Group) This qualifier is used to designate a group as global. By default, GLOBAL is assumed. LOCAL and GLOBAL qualifiers are mutually exclusive.

**/HOME\_DIRECTORY=file-spec**

(User) This qualifier designates the home directory for the user account. The value specified must be a valid and existing file-specification used as a path.

**/HOME\_DRIVE=drive-letter**

(User) This qualifier allows the user to designate a drive letter to be associated with the HOME\_DIRECTORY when a UNC style path is specified. By default, Windows use drive letter "Z".

**/LOCAL[=(group-name,...)]**

(User) This qualifier is used to add the user to one or more local groups. The group names specified must be local groups that already exist.

(Group) This qualifier is used to designate a group as local. By default, GLOBAL is assumed. LOCAL and GLOBAL qualifiers are mutually exclusive.

**/[NO]LOG**

This qualifier causes a message to be displayed indicating successful operation. By default, messages are displayed only when an error occurs.

**/MEMBER=(user-name,...)**

(Group) This qualifier allows you to add users to a group. The value specified is one or more usernames.

**/NAME=string**

(User) This qualifier allows the administrator to associate a full and complete name with the account rather than just the short username.

**/ON=machine-name**

This optional qualifier allows you to direct the execution of this command to another Windows NT system. By default, unless the DOMAIN qualifier is specified, commands are executed on the local system they are issued.

**/PASSWORD[=string]**

(User) This qualifier designates the password for this user account. If the PASSWORD qualifier is specified without any value, XLNT will prompt for the password without echo. NOPASSWORD means that the account will be created without a password.

**/[NO]PASSWORD\_EXPIRES**

(User) The qualifier PASSWORD\_EXPIRES is used to set a user's password to expire at next logon. If the user's password is already set to expire at next logon, then this qualifier is clear that setting. Specifying

NOPASSWORD\_EXPIRES means that the password is to never expire. This setting will cause WinNT to ignore any other password expiration that may be effect. By default, if this qualifier is omitted, then the current WinNT System Policy governing password expiration lifetime is in effect.

**/PROFILE\_PATH=file-spec**

(User) This qualifier sets the path for the user's account profile.

**/RAS=([NO]DIAL, NONE | CALLER | ADMIN, phone-string)**

(User) This qualifier controls the user's ability to access the NT machine via RAS facilities. The DIAL keyword indicates whether the user can dial into the system or not. The NONE, CALLER or ADMIN indicate whether callback processing is enabled and if ADMIN is specified, the "phone-string" argument is used to indicate the number to callback.

**[NO]REQUIRED PASSWORD**

(User) This qualifier determines whether a password is required for the user account. By default, passwords are required.

**/RIGHTS=(right-name,...)**

This qualifier may be specified to include specific user rights. One or more user rights may be added separated by a comma. Valid user rights are as follows (please note that you may enter only that portion which is unique and unambiguous rather than the entire string):

SeBackupPrivilege, SeTcbPrivilege, SeMachineAccountPrivilege, SeChangeNotifyPrivilege, SeSystemtimePrivilege, SeCreatePagefilePrivilege, SeCreateTokenPrivilege, SeCreatePermanentPrivilege, SeDebugPrivilege, SeRemoteShutdownPrivilege, SeAuditPrivilege, SeIncreaseQuotaPrivilege, SeIncreaseBasePriorityPrivilege, SeLoadDriverPrivilege, SeLockMemoryPrivilege, SeSecurityPrivilege, SeSystemEnvironmentPrivilege, SeProfileSingleProcessPrivilege, SeSystemProfilePrivilege, SeAssignPrimaryTokenPrivilege, SeRestorePrivilege, SeShutdownPrivilege, SeInteractiveLogonRight, SeNetworkLogonRight, SeServiceLogonRight, SeBatchLogonRight

**/SCRIPT\_PATH=file-spec**

(User) This qualifier sets the logon script path. If specified, the file-specification must be an operating system valid specification that will be used to retrieve a file for logon script processing. If omitted, no logon script processing is performed.

**[NO]TIMES=(ALL | time-spec,...)**

(User) This qualifier represents the logon day/hours. TIMES is expressed as day[-day][,time[-time]] limited to 1-hour increments. Days can be spelled out or abbreviated. Hours must be 24-hour notation (0-23). ALL means a user can always log on, and /NOTIMES means a user can never log on. Day (or day range) must always be specified before times. When multiple times are specified, they apply to the day (or day range) previously specified until a new day (or day range) is encountered. For example, /TIME=(MON-FRI,9-17,SAT,6-7,10-12) means that the user may logon Monday through Friday from 9am until 5pm and on Saturday from 6am to 7am and then from 10am until 12noon. All other non-specified times, the user may not login. If omitted, /TIMES=ALL is the default.

**/WORKSTATION=( ALL | machine-name,...)**

(User) This qualifier allows the administrator to determine specific machines within the domain that this user may logon. As many as eight (8) machines may be specified. By default, user's may logon to any machine in the domain.

**Example:**

```
$ SECURITY CREATE USER TESTUSER /DOMAIN /NAME="A TEST USER"
```

This example will create a user on the domain with a username of TESTUSER, a Fullname of "A TEST USER" and a password will be prompted by the SECURITY CREATE command.

## 12.2 SECURITY DELETE

The SECURITY DELETE command allows a Windows NT Administrator to remove a user and/or group from the Windows NT security system. Please note that the qualifiers which follow may contain a designation such as (User) or (Group). That designation means that the qualifier is only valid when processing that entity. If no such designation is made, the qualifier is available for use without restriction.

### Format:

**SECURITY DELETE/Qualifiers entity-type entity**

### Parameters:

#### **entity-type**

represents one of the following: GROUP or USER.

#### **entity**

this parameter specifies either a Groupname (when *entity-type* is GROUP) or a Username (when *entity-type* is USER). The entity must already exist in the target machine's security database.

### Description:

This command allows an Administrator to delete a User and/or Group account. Please note that once an account is deleted, you cannot simply recreate the account to restore the previous SID.

### Qualifiers:

#### **/[NO]CONFIRM**

This qualifier when specified indicates that the user is to be prompted for confirmation prior to performing the DELETE operation. By default, no confirmation is requested. Please note that if you will be issuing this command interactively and desire the confirmation protection as a default, simply follow the example noted in the [COPY Command](#) and it's use of F\$MODE.

#### **/DOMAIN[=domain]**

This qualifier when specified indicates that security information is to be targeted to the logon domain. If the /DOMAIN qualifier is not specified then security information is targeted to the workstation. If the domain value is not specified then the current domain is used.

#### **/GLOBAL**

(Group) This qualifier is used to designate a group as global. By default, the group is considered global.

#### **/LOCAL**

(Group) This qualifier is used to designate a group as local. By default, the group is considered global.

#### **/[NO]LOG**

This qualifier causes a message to be displayed indicating successful operation. By default, messages are displayed only when an error occurs.

#### **/ON=machine-name**

This optional qualifier allows you to direct the execution of this command to another Windows NT system. By default, unless the DOMAIN qualifier is specified, commands are executed on the local system they are issued.

### Example:

```
$ SECURITY DELETE USER TESTUSER /DOMAIN /LOG
```

User TESTUSER successfully deleted

This example will delete the user TESTUSER and all information associated with it. If the user TESTUSER was created again with TESTUSER it will not retain the original information.

```
$ SECURITY DELETE GROUP MYUSERS /DOMAIN /LOCAL /LOG  
Group MYUSERS successfully deleted
```

This example will delete the local group USERS from the domain.

## 12.3 SECURITY MODIFY

The SECURITY MODIFY command allows a Windows NT Administrator to modify users and/or groups in the Windows NT security system. In addition, users as members of a group may also be modified. Please note that the qualifiers which follow may contain a designation such as (User) or (Group). That designation means that the qualifier is only valid when processing that entity. If no such designation is made, the qualifier is available for use without restriction.

### Format:

**SECURITY MODIFY/Qualifiers entity-type entity**

### Parameters:

#### **entity-type**

represents one of the following: GROUP or USER.

#### **entity**

this parameter specifies either a Groupname (when *entity-type* is GROUP) or a Username (when *entity-type* is USER). The entity must already exist in the target machine's security database.

### Description:

This command is used to modify an existing User and/or Group account.

### Qualifiers:

#### **/[NO]ACTIVE**

(User) This qualifier indicates whether the user account should be considered active (enabled) or inactive (disabled) for security purposes.

#### **/[NO]CHANGE\_PASSWORD**

(User) This qualifier indicates whether the user can change the password for the account.

#### **/CURRENT\_PASSWORD[=string]**

(User) This qualifier specifies the current password for the user and is mandatory when attempting to change a user account's password. If the PASSWORD qualifier is omitted and this qualifier is specified, then the PASSWORD qualifier is assumed.

#### **/COMMENT=string**

This optional qualifier provides a descriptive comment about the user.

#### **/COUNTRY\_CODE=nnn**

(User) Uses the operating system country code to determine language dependent help files and other language dependent facilities. By default a country code of zero (0) means the native country associated with the system.

#### **/DOMAIN[=domain]**

This qualifier when specified indicates that security information is to be targeted to the logon domain. If the /DOMAIN qualifier is not specified then security information is targeted to the workstation. If the domain value is not specified then the current domain is used.

#### **/[NO]EXPIRES=date-time**

(User) This qualifier allows the administrator to set the expiration date (if any) for this account. If /EXPIRES is specified, the date-time is specified as a date in absolute time format (i.e. DD-MMM-YYYY). This date is used as the expiration date for the account. /NOEXPIRES means that no expiration date is associated with the account.

**/[NO]GLOBAL=(group-name,...)**

(User) This qualifier is used to add or remove the user to/from one or more global groups. The group names specified must be global groups that already exist.

**/HOME\_DIRECTORY=file-spec**

(User) This qualifier designates the home directory for the user account. The value specified must be a valid and existing file-specification used as a path.

**/HOME\_DRIVE=drive-letter**

(User) This qualifier allows the user to designate a drive letter to be associated with the HOME\_DIRECTORY when a UNC style path is specified.

**/[NO]LOCAL=(group-name,...)**

(User) This qualifier is used to add or remove the user to/from one or more local groups.

**/[NO]LOG**

This qualifier causes a message to be displayed indicating successful operation. By default, messages are displayed only when an error occurs.

**/[NO]MEMBER=(user-name,...)**

(Group) This qualifier allows you to add or remove users to/from a group. The value specified is one or more usernames.

**/NAME=string**

(User) This qualifier allows the administrator to associate a full and complete name with the account rather than just the short username.

**/ON=machine-name**

This optional qualifier allows you to direct the execution of this command to another Windows NT system. By default, unless the DOMAIN qualifier is specified, commands are executed on the local system they are issued.

**/[NO]PASSWORD[=string]**

(User) This qualifier changes the password for this user account. If the PASSWORD qualifier is specified without any value, XLNT will prompt for the password without echo. NOPASSWORD means that the account will be modified to not have a password.

**/[NO]PASSWORD\_EXPIRES**

(User) The qualifier PASSWORD\_EXPIRES is used to set a user's password to expire at next logon. If the user's password is already set to expire at next logon, then this qualifier is clear that setting. Specifying NOPASSWORD\_EXPIRES means that the password is to never expire. This setting will cause WinNT to ignore any other password expiration that may be effect.

**/PROFILE\_PATH=file-spec**

(User) This qualifier sets the path for the user's account profile.

**/[NO]REQUIRED\_PASSWORD**

(User) This qualifier determines whether a password is required for the user account.

**/RAS=([NO]DIAL, NONE | CALLER | ADMIN, phone-string)**

(User) This qualifier controls the user's ability to access the NT machine via RAS facilities. The DIAL keyword indicates whether the user can dial into the system or not. The NONE, CALLER or ADMIN indicate whether callback processing is enabled and if ADMIN is specified, the "phone-string" argument is used to indicate the number to callback.

**/[NO]RIGHTS=(right-name,...)**

This qualifier may be specified to include or exclude one or more user's rights. For a complete list of user rights, please read the RIGHTS qualifier under the [SECURITY CREATE](#) command.

**/SCRIPT\_PATH=file-spec**

(User) This qualifier sets the logon script path. If specified, the file-specification must be an operating system valid specification that will be used to retrieve a file for logon script processing.

**/[NO]TIMES=(ALL | time-spec,...)**

(User) This qualifier represents the logon day/hours. TIMES is expressed as day[-day][,time[-time]] limited to 1-hour increments. Days can be spelled out or abbreviated. Hours must be 24-hour notation (0-23). ALL means a user can always log on, and /NOTIMES means a user can never log on. Day (or day range) must always be specified before times. When multiple times are specified, they apply to the day (or day range) previously specified until a new day (or day range) is encountered. For example, /TIME=(MON-FRI,9-17,SAT,6-7,10-12) means that the user may logon Monday through Friday from 9am until 5pm and on Saturday from 6am to 7am and then from 10am until 12noon. All other non-specified times, the user may not login. If omitted, /TIMES=ALL is the default.

**/UNLOCK**

(User) This qualifier allows the administrator to unlock a locked account.

**/USERNAME=string**

(User) This qualifier allow the administrator to rename an existing user account.

**/WORKSTATION=( ALL | machine-name,...)**

(User) This qualifier allows the administrator to determine specific machines within the domain that this user may logon. As many as eight (8) machines may be specified. The keyword ALL indicates all workstations may be used.

**Examples:**

```
$ SECURITY MODIFY USER TESTUSER /NOACTIVE /DOMAIN
```

This command will disable the account of the user TESTUSER on the current domain. To enable the user account again enter the command:

```
$ SECURITY MODIFY USER TESTUSER /ACTIVE /DOMAIN
$ SECURITY MODIFY USER TESTUSER /EXPIRES="01-JAN-1997" /HOME_DIRECTORY=C:\TESTUSER -
_$ /DOMAIN
```

This example will modify the user account of TESTUSER on the current domain and set the account to expire on January 1, 1997 at 12:00am, and also set his home directory to C:\TESTUSER.

## 12.4 SECURITY SHOW

The SECURITY SHOW command allows a Windows NT Administrator to view user, group and policy level security.

### Format:

**SECURITY SHOW/Qualifiers entity-type [entity]**

### Parameters:

#### **entity-type**

represents one of the following: GROUP, POLICY or USER.

#### **entity**

this optional parameter specifies either a Groupname (when *entity-type* is GROUP) or a Username (when *entity-type* is USER). When *entity-type* is POLICY, this parameter is ignored. If *entity* is omitted, the all Users or Groups are displayed.

### Description:

The SECURITY SHOW command can be used to obtain a brief or full display of User and Group accounts on a system. You can also obtain a display of the security policy in effect.

### Qualifiers:

#### **/DOMAIN[=domain]**

This qualifier when specified indicates that security information is to be retrieved from the logon domain. If the /DOMAIN qualifier is not specified then security information is retrieved from the workstation. If the domain value is not specified then the current domain is used.

#### **/[NO]FORMAT**

This qualifier, only valid for the SHOW command, allows you to determine whether the report should be generated in a format for simpler parsing (/NOFORMAT) or whether the report should be formatted for human reading (/FORMAT). FORMAT is the default.

#### **/FULL**

This qualifier indicates that a complete listing of the Group and/or User's security information, including user rights, are to be displayed. By default, rights information is not displayed.

#### **/[NO]GLOBAL**

This qualifier may be specified to omit the display of global group(s) that user(s) may be a member of. By default, GLOBAL is assumed which means that all global group(s) the user is a member of are displayed.

#### **/[NO]LOCAL**

This qualifier may be specified to omit the display of local group(s) that user(s) may be a member of. By default, LOCAL is assumed which means that all local group(s) the user is a member of are displayed.

#### **/ON=machine-name**

This optional qualifier allows you to direct the execution of this command to another Windows NT system. By default, unless the DOMAIN qualifier is specified, commands are executed on the local system they are issued.

#### **/OUTPUT[=file-specification]**

This optional qualifier allows you to direct the output of this command to a file. By default, output will appear on the console window the command was issued from (or more correctly where \$STDOUT is defined). Any valid Windows NT specification may be entered.

## **/[NO]RIGHTS**

This qualifier may be specified to include or omit the display of a group or user's rights. By default, rights are not included for display.

### **Examples:**

```
$ SECURITY SHOW USER TESTUSER /DOMAIN /FULL
```

```
User Account from MY-DOMAIN
```

```
User Name = TESTUSER  
Full Name = Domain Test User  
Comment =  
Country Code = 0  
Account Active = Yes  
Account Locked = No  
Account Expires = Never  
Password Last Set = 7/17/97 10:23:02 AM  
Password Expires = N/A  
Password Changed = 7/17/97 10:23:02 AM  
Password Required = Yes  
User may change password = Yes  
Workstations Allowed = All  
Home Directory =  
Profile Path =  
Logon Script =  
Last Logon = Never  
Logon Error Count = 0  
Logon Success Count = 0  
Logon Hours Allowed = All  
Primary Global Group = Domain Users  
Local Groups = Users  
Global Groups = Domain Users
```

```
TESTUSER has the following rights  
-None
```

```
The following Local groups have rights:
```

```
Users has the following rights  
-None
```

```
The following Global groups have rights:
```

```
Domain Users has the following rights  
-SeNetworkLogonRight  
-SeBatchLogonRight
```

This example displayed all information for user TESTUSER on the domain. The /FULL qualifier prints out all of the local and global groups that the user is a member of and all granted rights to those groups.

# 13 Service Commands

This section summarizes the syntax of the XLNT Service related commands. These commands are available only under Windows NT.

## 13.1 CONFIGURE SERVICE Command

The **CONFIGURE SERVICE** command modifies an already installed NT service on the designated computer.

**Format:**

**CONFIGURE SERVICE** *service-name*

**Parameters:**

**service-name**

a character string of up to 256 characters that names the existing service to be changed.

**Description:**

The **CONFIGURE SERVICE** command is used by Administrators to modify attributes and characteristics of an existing service. Typically the more advanced qualifiers would be used for "home-grown" services while qualifiers such as **/START\_TYPE** could be used for any service. This command provides complete access to the Windows NT Service attributes.

**Qualifiers:**

**/ACCESS=(string[,...])**

Specifies the access to the service. Any or all of the following access types can be specified:

**ALL** - All access rights are granted.

**CHANGE\_CONFIG** - allows the service to change its configuration.

**ENUM\_DEPENDENTS** - allows the service to enumerate all the services dependent on it.

**INTERROGATE** - allows the service to be interrogated as to its current status.

**PAUSE\_CONTINUE** - allows the service to be paused and continued.

**QUERY\_CONFIG** - allows the service to query its configuration.

**QUERY\_STATUS** - allows the service to query its status.

**START** - allows the service to be started.

**STOP** - allows the service to be stopped.

**STANDARD\_RIGHTS** - allows the service to be deleted and to read and modify its security descriptors.

**GENERIC\_READ** - combines the following accesses: **STANDARD\_RIGHTS\_READ**, **QUERY\_CONFIG**, **QUERY\_STATUS** and **ENUM\_DEPENDENTS**.

**GENERIC\_WRITE** - combines the following accesses: **STANDARD\_RIGHTS\_WRITE** and **CHANGE\_CONFIG**.

**GENERIC\_EXECUTE** - combines the following accesses: **STANDARD\_RIGHTS\_EXECUTE**, **START**, **STOP**, **PAUSE\_CONTINUE**, **INTERROGATE**.

**/DEPENDENCIES = (string [...])**

This qualifier allows you to specify one or more services or load ordering groups that must be started prior to starting this service.

**/DISPLAY\_NAME = name**

A character string of up to 256 characters that is to be used by user interface programs to identify the service. If omitted, the *service name* is used.

**/ERROR\_CONTROL = NORMAL | IGNORE | SEVERE | CRITICAL**

Specifies the severity of the error if this service fails to start during startup, and determines the action taken by the startup program if failure occurs. One of the following values can be specified:

**NORMAL** - startup program logs the error and display a message but continues the startup operation. This is the default.

**IGNORE** - startup program logs the error but continues the startup operation.

**SEVERE** - startup program logs the error. If the last-known-good configuration is being started, the startup operation continues. Otherwise, the system is restarted with the last-known-good configuration.

**CRITICAL** - startup program logs the error, if possible. If the last-known-good configuration is being started, the startup operation fails. Otherwise, the system is restarted with the last-known-good configuration.

**/LOAD\_ORDER\_GROUP = string**

A character string that allows services (typically drivers) to be grouped together for the further purposes of sequencing loads within the group using the /TAG\_ID qualifier.

**/ON = computer-name**

A character string that names the target computer. If omitted, the service will be modified on the local computer.

**/PASSWORD = string**

A string containing the password to the account name specified by the /SERVICE\_START\_NAME qualifier.

**/PATH = file-specification**

This qualifier specifies the path name of the executable image that contains the service. **Note: The path cannot be specified as a UNC specification.**

**/SERVICE\_START\_NAME = string**

If the service type is OWN\_PROCESS, this string specifies the account name in which the service process will be logged on when it runs, in the form "DomainName\Username". If the account belongs to the built-in domain, ".\Username" can be specified. If omitted, the service will be logged on as the "LocalSystem" account.

**/START\_TYPE = AUTO | BOOT | DISABLED | MANUAL | SYSTEM**

Specifies when to start the service. One of the following values can be specified:

**AUTO** - specifies that the service will start automatically during system startup.

**BOOT** - specifies a device driver started by the operating system.

**DISABLED** - specifies a service that cannot be started.

**MANUAL** - specifies that the service must be manually started via the XLNT START SERVICE command or the SERVICES function of the Control Panel group. This is the default.

**SYSTEM** - specifies a device driver started by the IolnitSystem function.

**/TAG\_ID = number**

When used with /LOAD\_ORDER\_GROUP, this qualifier allows the specific sequencing of services in the order they are to be loaded. This qualifier requires a numeric integer value that indicates the order the service is loaded. For example, 1 indicates that this service is to be loaded first within the group, 2 would be the second service to be

loaded, etc.

**/TYPE = string**

Specifies the type of service to be installed. One of the following may be specified:

**FILE\_SYSTEM\_DRIVER** - specifies a file system driver service

**KERNEL\_DRIVER** - specifies a kernel-mode driver service

**OWN\_PROCESS** - specifies a Win32 service that runs in its own process.

**SHARE\_PROCESS** - specifies a Win32 service that shares a process with other services.

The **INTERACTIVE** modifier may be specified with either of the above service types to identify a Win32 service that interacts with the desktop.

**Example:**

**\$ CONFIGURE SERVICE XLNTNET/START\_TYPE=AUTO**

This command modifies the service XLNTNET. The initial startup state is set to "Automatic" (rather than Manual which is the default). Automatic means that the service will be automatically started when Windows NT is booted.

## 13.2 INSTALL SERVICE Command

The **INSTALL SERVICE** command installs an NT service on the designated computer.

### Format:

**INSTALL SERVICE service-name**

### Parameters:

**service-name**

a character string of up to 256 characters that names the service to be installed.

### Description:

The **INSTALL SERVICE** command allows an Administrator or application developer to install a service on a Windows NT system (local or remote) and specify all required service attributes.

### Qualifiers:

**/ACCESS=(string[,...])**

Specifies the access to the service. Any or all of the following access types can be specified:

**ALL** - All access rights are granted. This is the default.

**CHANGE\_CONFIG** - allows the service to change its configuration.

**ENUM\_DEPENDENTS** - allows the service to enumerate all the services dependent on it.

**INTERROGATE** - allows the service to be interrogated as to its current status.

**PAUSE\_CONTINUE** - allows the service to be paused and continued.

**QUERY\_CONFIG** - allows the service to query its configuration.

**QUERY\_STATUS** - allows the service to query its status.

**START** - allows the service to be started.

**STOP** - allows the service to be stopped.

**STANDARD\_RIGHTS** - allows the service to be deleted and to read and modify its security descriptors.

**GENERIC\_READ** - combines the following accesses: **STANDARD\_RIGHTS\_READ**, **QUERY\_CONFIG**, **QUERY\_STATUS**, and **ENUM\_DEPENDENTS**.

**GENERIC\_WRITE** - combines the following accesses: **STANDARD\_RIGHTS\_WRITE** and **CHANGE\_CONFIG**.

**GENERIC\_EXECUTE** - combines the following accesses: **STANDARD\_RIGHTS\_EXECUTE**, **START**, **STOP**, **PAUSE\_CONTINUE**, **INTERROGATE**.

**/DEPENDENCIES = (string [,...])**

This qualifier allows you to specify one or more services or load ordering groups that must be started prior to starting this service.

**/DISPLAY\_NAME = name**

A character string of up to 256 characters that is to be used by user interface programs to identify the service. If omitted, the *service name* is used.

**/ERROR\_CONTROL = NORMAL | IGNORE | SEVERE | CRITICAL**

Specifies the severity of the error if this service fails to start during startup, and determines the action taken by the

startup program if failure occurs. One of the following values can be specified:

**NORMAL** - startup program logs the error and display a message but continues the startup operation. This is the default.

**IGNORE** - startup program logs the error but continues the startup operation.

**SEVERE** - startup program logs the error. If the last-known-good configuration is being started, the startup operation continues. Otherwise, the system is restarted with the last-known-good configuration.

**CRITICAL** - startup program logs the error, if possible. If the last-known-good configuration is being started, the startup operation fails. Otherwise, the system is restarted with the last-known-good configuration.

#### **/LOAD\_ORDER\_GROUP = string**

A character string that allows services (typically drivers) to be grouped together for the further purposes of sequencing loads within the group using the /TAG\_ID qualifier.

#### **/ON = computer-name**

A character string that names the target computer. If omitted, the service will be installed on the local computer.

#### **/PASSWORD = string**

A string containing the password to the account name specified by the /SERVICE\_START\_NAME qualifier. This qualifier is ignored if the /SERVICE\_START\_NAME qualifier is omitted.

#### **/PATH = file-specification**

This qualifier specifies the path name of the executable image that contains the service. **Note: The path cannot be specified as a UNC specification.** If this qualifier is omitted, the default path will be the current device/directory and the image name will be the same as the service name with the “.EXE” suffix appended.

#### **/SERVICE\_START\_NAME = string**

If the service type is OWN\_PROCESS, this string specifies the account name in which the service process will be logged on when it runs, in the form “DomainName\Username”. If the account belongs to the built-in domain, “.\Username” can be specified. If omitted, the service will be logged on as the “LocalSystem” account.

#### **/START\_TYPE = AUTO | BOOT | DISABLED | MANUAL | SYSTEM**

Specifies when to start the service. One of the following values can be specified:

**AUTO** - specifies that the service will start automatically during system startup.

**BOOT** - specifies a device driver started by the operating system.

**DISABLED** - specifies a service that cannot be started.

**MANUAL** - specifies that the service must be manually started via the XLNT START SERVICE command or the SERVICES function of the Control Panel group. This is the default.

**SYSTEM** - specifies a device driver started by the lolnitSystem function.

#### **/TAG\_ID = number**

When used with /LOAD\_ORDER\_GROUP, this qualifier allows the specific sequencing of services in the order they are to be loaded. This qualifier requires a numeric integer value that indicates the order the service is loaded. For example, 1 indicates that this service is to be loaded first within the group, 2 would be the second service to be loaded, etc.

#### **/TYPE = string**

Specifies the type of service to be installed. One of the following may be specified:

**FILE\_SYSTEM\_DRIVER** - specifies a file system driver service

**KERNEL\_DRIVER** - specifies a kernel-mode driver service

**OWN\_PROCESS** - specifies a Win32 service that runs in its own process.

**SHARE\_PROCESS** - specifies a Win32 service that shares a process with other services.

The **INTERACTIVE** modifier may be specified with either of the above service types to identify a Win32 service that interacts with the desktop.

**Example:**

**\$ INSTALL SERVICE XLNTNET/START\_TYPE=AUTO**

This command installs the service XLNTNET. The initial startup state is set to "Automatic" (rather than Manual which is the default). Automatic means that the service will be automatically started when Windows NT is booted.

## 13.3 PAUSE SERVICE Command

The PAUSE SERVICE command pauses the specified service on the designated computer.

**Format:**

**PAUSE SERVICE service-name**

**Parameters:**

**service-name**

A character string of up to 256 characters that names the service to be paused.

**Qualifiers:**

**/ON = computer-name**

A character string that names the target computer. If omitted, the service will be paused on the local computer.

**Example:**

**\$ PAUSE SERVICE XLNTNET/ON=PLUTO**

This command will pause the service XLNTNET on machine PLUTO. The service must have been started for this command to succeed.

## 13.4 REMOVE SERVICE Command

The REMOVE SERVICE command deletes the specified service from the designated computer. This command is the opposite of INSTALL SERVICE.

**Format:**

**REMOVE SERVICE service-name**

**Parameters:**

**service-name**

A character string of up to 256 characters that names the service to be removed.

**Qualifiers:**

**/ON = computer-name**

A character string that names the target computer. If omitted, the service will be removed from the local computer.

**Example:**

**\$ REMOVE SERVICE XLNTNET**

This command removes the existing service XLNTNET.

## 13.5 RESET SERVICE Command

The RESET SERVICE command will stop and then restart the specified Windows service.

**Format:**

**RESET SERVICE service-name**

**Parameter:**

**service-name**

A character string of up to 256 characters that names the service to be reset.

**Qualifier:**

**/ON=computer-name**

A character string that names the target computer. If omitted, the service will be reset on the local computer.

## 13.6 RESUME SERVICE Command

The RESUME SERVICE command resumes the specified service on the designated computer. This command is the opposite of PAUSE SERVICE.

**Format:**

**RESUME SERVICE service-name**

**Parameters:**

**service-name**

A character string of up to 256 characters that names the service to be resumed.

**Qualifiers:**

**/ON = computer-name**

A character string that names the target computer. If omitted, the service will be resumed on the local computer.

**Example:**

**\$ RESUME SERVICE XLNTNET**

This command resumes the paused service XLNTNET.

## 13.7 SHOW SERVICE Command

The SHOW SERVICE command displays the current status of a specified service or all services on the designated computer.

### Format:

**SHOW SERVICE [service-name]**

### Parameters:

#### **service-name**

This optional parameter when specified causes a specific service to be displayed. When omitted, all services are displayed.

### Description:

The SHOW SERVICE command is used to obtain a brief or full display of a selected service or all services on the local or specified remote machine. By default, a brief display of all installed services with their service names, description and active status is presented. To display a specific service, simply enter the service name using the optional parameter *service-name*. To obtain a full display of a specific service use the /FULL qualifier. This qualifier will also display the service's list of permissions.

### Qualifiers:

#### **/ACTIVE**

This qualifier causes active services only to be displayed.

#### **/FULL**

This qualifier causes a complete display of all attributes and characteristics associated with a service (including security).

#### **/ON = computer-name**

A character string that names the target computer. If omitted, the services on the local computer will be displayed.

#### **/OUTPUT = filename**

Controls where the output of the command is sent. By default, the display is written to the console window.

### Example:

#### **\$ SHOW SERVICE**

Status of Services on TEST 29-Aug-1996 13:04:49

Service	Description	State
Alerter	Alerter	Stopped
Browser	Computer Browser	Active
ClipSrv	ClipBook Server	Stopped
DHCP	DHCP Client	Stopped
EventLog	EventLog	Active
FTPSVC	FTP Server	Active
LanmanServer	Server	Active
LanmanWorkstation	Workstation	Active
LmHosts	TCP/IP NetBIOS Helper	Active
LPDSVC	TCP/IP Print Server	Stopped
Messenger	Messenger	Active
NetDDE	Network DDE	Stopped
NetDDEdsdm	Network DDE DSDM	Stopped
NetLogon	Net Logon	Active
NtLmSsp	NT LM Security Support Provider	Stopped
OLE	OLE	Active
RasMan	Remote Access Connection Manager	Active

RemoteAccess	Remote Access Server	Stopped
Repl icator	Di rectory Repl icator	Stopped
RPCLOCATOR	Remote Procedure Call (RPC) Locator	Stopped
RPCSS	Remote Procedure Call (RPC) Service	Stopped
Schedule	Schedule	Active
SimpleTcp	Simple TCP/IP Services	Active
SMTP	SMTP Server	Stopped
SNMP	SNMP Service	Active
Spooler	Spooler	Active
UPS	UPS	Stopped
XLntNet	XLnt Network Service	Active

This example shows a complete listing of all the services on this machine.

### \$ SHOW SERVICE BQMSQMGR/FULL

Status of Services on TEST 14-Apr-1998 15: 39: 42

Service: BqmsQmgr					Status: Stopped
Description: BQMS Queue Manager	Type:	Own Process	Path: C:\ASCI\NEWXLNT\BQMSQMGR.EXE		
Startup: Automatic	Load Order Group:		Error Control: Normal	Tag:	
Start Name: Local System	Accepts:		Depends On: RPCSS	RPCLOCATOR	
Security Information:			Everyone		
Read (FQEIUR)		Power Users	Read (FQEGSZIUR)		
Administrators	Full Control (FCQEGSZIURDPO)				
NT AUTHORITY\SYSTEM	Read (FQEGSZIUR)				

This example shows a complete listing of the service BQMSQMGR. Note the dependencies and permissions list. For more information on service permissions, see the SET PERMISSIONS command.

## 13.8 START SERVICE Command

The START SERVICE command starts the specified service on the designated computer.

### Format:

**START SERVICE service-name**

### Parameters:

**service-name**

A character string of up to 256 characters that names the service to be started.

### Description:

This command is used to start a service that had either been stopped or had never been started (possibly as a result of its START\_TYPE being MANUAL or DISABLE).

### Qualifiers:

**/ON = computer-name**

A character string that names the target computer. If omitted, the service will be started on the local computer.

**/ARGUMENTS = (string [...])**

One or more argument strings that are to be passed to the service when it is started. Each argument should be separated from the other by a comma.

### Example:

#### \$ START SERVICE XLNTNET

This command starts the XLNTNET service manually.

## 13.9 STOP SERVICE Command

The `STOP SERVICE` command stops the specified service on the designated computer.

**Format:**

`STOP SERVICE service-name`

**Parameters:**

`service-name`

A character string of up to 256 characters that names the service to be stopped.

**Qualifiers:**

`/ON = computer-name`

A character string that names the target computer. If omitted, the service will be stopped on the local computer.

**Example:**

`$ STOP SERVICE XLNTNET`

This command stops the XLNTNET service.

# 14 SHARE Commands

The series of SHARE commands below allow the Administrator to create and maintain shares for possible use by remote clients. The SHARE commands which follow are.

## 14.1 SHARE ADD

The SHARE ADD command will share a local resource to the network.

**Format:**

**SHARE ADD resource**

**Parameter:**

**resource**

Specifies the name of the resource to be shared.

**Qualifiers:**

**/COMMENT=string**

Specifies an optional string that contains a comment about the shared resource.

**/[NO]LOG**

Specifies whether a message is displayed if the command executes successfully.

**/MAX\_CONNECTIONS=number**

Specifies the maximum number of shared connections that the resource can accommodate.

**/ON=servername**

Specifies the name of the server computer on which the operation will execute. If omitted, the local computer will be used.

**/PATH=pathstring**

Specifies a string that contains the local path to the shared resource. For disks, this is the path being shared. For print queues, this is the name of the print queue being shared.

**/TYPE=string**

Specifies a value indicating the type of resource being shared. It can be one of the following set:

DISK	Disk Drive
PRINTQ	Print Queue
COMM	Communications Device
IPC	Interprocess Communication (IPC)

**Example:**

```
$ SHARE ADD CDRIVE /PATH=C:\ASCII /TYPE=DISK /LOG
XLNT-I-ADDSHARE, share added successfully
```

This example will create a share of the C:\ASCII directory on the local computer and call it CDRIVE. It can now be seen by anyone on the network. The /TYPE=DISK tells us that we are sharing a disk drive. You can set permissions on the share with [SET PERMISSIONS](#).

## 14.2 SHARE DELETE

The SHARE DELETE command will delete a share name from the list of shared resources and will disconnect all connections to the shared resource.

**Format:**

**SHARE DELETE resource**

**Parameter:**

**resource**

Specifies the name of the resource to be deleted.

**Qualifiers:**

**/[NO]LOG**

Specifies whether a message is displayed if the command executes successfully.

**/ON=servername**

Specifies the name of the server computer on which the operation will execute. If omitted, the local computer will be used.

**Example:**

```
$ SHARE DELETE CDRIVE /LOG
XLNT-I-DELSHARE, share deleted successfully
```

This example will delete a share on the local machine with the name of CDRIVE.

## 14.3 SHARE SET

The SHARE SET command will allow the user to modify the parameters of a shared resource.

**Format:**

**SHARE SET resource**

**Parameter:**

**resource**

Specifies the name of the shared resource to be modified.

**Qualifiers:**

**/COMMENT=string**

Specifies an optional string that contains a comment about the shared resource.

**/[NO]LOG**

Specifies whether a message is displayed if the command executes successfully.

**/MAX\_CONNECTIONS=number**

Specifies the maximum number of shared connections that the resource can accommodate.

**/ON=servername**

Specifies the name of the server computer on which the operation will execute. If omitted, the local computer will be used.

**/TYPE=string**

Specifies a value indicating the type of resource being shared. It can be one of the following set:

DISK	Disk Drive
PRINTQ	Print Queue
COMM	Communications Device
IPC	Interprocess Communication (IPC)

**Example:**

```
$ SHARE SET CDRIVE /COMMENT="Sharing of my root C: DRIVE" /MAX_CONNECTIONS=10 /LOG
XLNT-I-SETSHARE, share modified successfully
```

This example will set various characteristics for a share named CDRIVE. A comment of "Sharing of my root C: DRIVE" as well as setting a maximum network connection of ten (10) will be set for the share.

# 14.4 SHARE SHOW

The SHARE SHOW command will display information about the shared resources on a server. The information will include the name of each shared resource; the type of resource being shared; the current number of connections to the resource; the resource path; and any comment made about the resource. If a non-administrative user enters this command a restricted view of non-administrative shares are displayed.

**Format:**

**SHARE SHOW [resource]**

**Parameter:**

**resource**

Specifies the name of the resource to be displayed. If omitted, information about all shared resources will be displayed.

**Qualifiers:**

**/ON=servername**

Specifies the name of the server computer on which the operation will execute. If omitted, the local computer will be used.

**/OUTPUT=filename**

Specifies where the output of the command is sent.

**Example:**

**\$ SHARE SHOW**

Shared Resources on TEST 08-Jul -1997 16:15:25

Share	Resource	Use	Comments
F	F: \	000	
ADMIN\$	C: \WINNT35	000	Remote Admin
IPC\$		001	Remote IPC
C\$	C: \	001	Default share
D\$	D: \	000	Default share
CSHARE	C: \	000	

# 15 ADS Commands

The series of commands that follow support the manipulation of user account or group Active Directory objects.

## 15.1 ADS CREATE

The ADS CREATE command creates a new Active Directory object.

### Format:

**ADS CREATE object name**

### Parameters:

#### **object**

This parameter determines the type of Active Directory object to be created. Its value may be specified as USER to create a new Active Directory User object; or as GROUP to create a new Active Directory Group object.

#### **name**

This parameter specifies the name of the object to be created.

### Qualifiers:

#### **/ACCOUNT\_NAME=account-name**

This qualifier contains a string specifying the new user's or group's SAM Account Name.

#### **/[NO]ACTIVE**

This optional qualifier specifies whether the new user object should be enabled or disabled. Default is to enable the object.

#### **/CONTAINER=container-name**

This qualifier specifies the name of the Active Directory object which will contain the new object.

#### **/FIRST\_NAME=first-name**

This optional qualifier specifies a new user's first name string.

#### **/INITIALS=initials**

This optional qualifier specifies a new user's initials.

#### **/LAST\_NAME=last-name**

This optional qualifier specifies a new user's last name string.

#### **/PASSWORD=password**

This optional qualifier specifies a login password.

#### **/DISPLAY\_NAME=display-name**

This optional qualifier specifies the user's display name.

#### **/DESCRIPTION=description**

This optional qualifier specifies a user's or group's description string.

**/OFFICE=office**

This optional qualifier specifies the new user's office string.

**/TELEPHONE=telephone\_number**

This optional qualifier specifies the new user's telephone number.

**/EMAIL=email-address**

This optional qualifier specifies the new user's email address.

**/WEB\_PAGE=url**

This optional qualifier specifies the new user's web page url.

**/STREET=street-address**

This optional qualifier specifies the new user's street address.

**/PO\_BOX=post-office-box**

This optional qualifier specifies the new user's post office box.

**/CITY=city**

This optional qualifier specifies the new user's city string.

**/STATE=state**

This optional qualifier specifies the new user's state or province string.

**/ZIP\_CODE=zip-code**

This optional qualifier specifies the new user's zip or postal code.

**/COUNTRY=country**

This optional qualifier specifies the new user's country. The value must be specified as the ISO 3166 Standard two-character country code.

**/LOGON\_NAME=logon-name**

This optional qualifier specifies the new user's logon name string.

**/EXPIRES=expiration-date**

This option qualifier specifies the date and time at which the user's account will expire. It can be specified as an absolute or combination time as per the Date and Time Format rules.

**/PROFILE\_PATH=profile-path**

This optional qualifier specifies a path to the user's profile. The value can be a local absolute path or a UNC path.

**/LOGON\_SCRIPT=logon-script**

This optional qualifier specifies the path to the user's logon script.

**/LOCAL\_PATH=home-directory**

This optional qualifier specifies the user's home directory path.

**/HOME\_PHONE=home-phone**

This optional qualifier specifies the user's home telephone number.

**/PAGER=pager-number**

This optional qualifier specifies the user's pager number.

**/MOBILE\_PHONE=mobile-phone**

This optional qualifier specifies the user's mobile phone number.

**/FAX\_NUMBER=fax-number**

This optional qualifier specifies the user's fax number.

**/IP\_PHONE=ip-phone**

This optional qualifier specifies the user's IP phone number.

**/TITLE=title**

This optional qualifier specifies the user's job title.

**/DEPARTMENT=department**

This optional qualifier specifies the user's department name.

**/COMPANY=company-name**

This optional qualifier specifies the user's company name.

**/SIMPLE\_DISPLAY\_NAME=display-name**

This optional qualifier specifies the name to be displayed in the address book for the specified user.

**/MEMBER=members**

This optional qualifier specifies one or more distinguished names of those objects that are members of the specified group.

**/MEMBER\_OF=members**

This optional qualifier specifies the distinguished names of one or more groups to which this object belongs.

**/SECURITY**

This optional qualifier specifies that the new group is security enabled, i.e., it is a Security group. It cannot be specified if the /DISTRIBUTION qualifier is also specified. If neither qualifier is specified, /SECURITY is the default.

**/DISTRIBUTION**

This optional qualifier specifies that the new group will be a Distribution group only. It cannot be specified if the /SECURITY qualifier is also specified.

**/LOCAL**

This optional qualifier specifies that the new group will be a local group. It cannot be specified if the /GLOBAL and/or /UNIVERSAL qualifier(s) are also specified. If none of these qualifiers are specified, /LOCAL is the default.

**/GLOBAL**

This optional qualifier specifies that the new group will be a global group. It cannot be specified if the /LOCAL and/or /UNIVERSAL qualifier(s) are also specified.

**/UNIVERSAL**

This optional qualifier specifies that the new group will be a universal group. It cannot be specified if the /LOCAL and/or /GLOBAL qualifier(s) are also specified.

## 15.2 ADS DELETE

The ADS DELETE command deletes the specified Active Directory object.

### Format:

**ADS DELETE object name**

### Parameters:

#### **object**

This parameter determines the type of Active Directory object to be deleted. Its value may be specified as **USER** to delete an existing user object; or as **GROUP** to delete an existing group object.

#### **name**

This parameter specifies the name of the object to be deleted.

### Qualifier:

**/CONTAINER=container-name**

This qualifier specifies the name of the Active Directory object which contains the object to be deleted.

## 15.3 ADS MODIFY

The ADS MODIFY command modifies the specified Active Directory object.

### Format:

**ADS MODIFY object name**

### Parameters:

#### **object**

This parameter determines the type of Active Directory object to be modified. Its value may be specified as **USER** to modify an existing user object; or as **GROUP** to modify an existing group object.

#### **name**

This parameter specifies the name of the object to be modified.

### Qualifiers:

#### **/ACCOUNT\_NAME=account-name**

This qualifier contains a string specifying the user's or group's SAM Account Name.

#### **/[NO]ACTIVE**

This optional qualifier specifies whether the user object should be enabled or disabled.

#### **/CONTAINER=container-name**

This qualifier specifies the name of the Active Directory object which will contain the object.

#### **/FIRST\_NAME=first-name**

This optional qualifier specifies the user's first name string.

#### **/INITIALS=initials**

This optional qualifier specifies the user's initials.

#### **/LAST\_NAME=last-name**

This optional qualifier specifies the user's last name string.

#### **/PASSWORD=password**

This optional qualifier specifies a login password.

#### **/DISPLAY\_NAME=display-name**

This optional qualifier specifies the user's display name.

#### **/DESCRIPTION=description**

This optional qualifier specifies the user's or group's description string.

#### **/OFFICE=office**

This optional qualifier specifies the user's office string.

#### **/TELEPHONE=telephone\_number**

This optional qualifier specifies the user's telephone number.

#### **/EMAIL=email-address**

This optional qualifier specifies the user's email address.

**/WEB\_PAGE=url**

This optional qualifier specifies the user's web page url.

**/STREET=street-address**

This optional qualifier specifies the user's street address.

**/PO\_BOX=post-office-box**

This optional qualifier specifies the user's post office box.

**/CITY=city**

This optional qualifier specifies the user's city string.

**/STATE=state**

This optional qualifier specifies the user's state or province string.

**/ZIP\_CODE=zip-code**

This optional qualifier specifies the user's zip or postal code.

**/COUNTRY=country**

This optional qualifier specifies the user's country. The value must be specified as the ISO 3166 Standard two-character country code.

**/LOGON\_NAME=logon-name**

This optional qualifier specifies the user's logon name string.

**/EXPIRES=expiration-date**

This option qualifier specifies the date and time at which the user's account will expire. It can be specified as an absolute or combination time as per the Date and Time Format rules.

**/PROFILE\_PATH=profile-path**

This optional qualifier specifies a path to the user's profile. The value can be a local absolute path or a UNC path.

**/LOGON\_SCRIPT=logon-script**

This optional qualifier specifies the path to the user's logon script.

**/LOCAL\_PATH=home-directory**

This optional qualifier specifies the user's home directory path.

**/HOME\_PHONE=home-phone**

This optional qualifier specifies the user's home telephone number.

**/PAGER=pager-number**

This optional qualifier specifies the user's pager number.

**/MOBILE\_PHONE=mobile-phone**

This optional qualifier specifies the user's mobile phone number.

**/FAX\_NUMBER=fax-number**

This optional qualifier specifies the user's fax number.

**/IP\_PHONE=ip-phone**

This optional qualifier specifies the user's IP phone number.

**/TITLE=title**

This optional qualifier specifies the user's job title.

**/DEPARTMENT=department**

This optional qualifier specifies the user's department name.

**/COMPANY=company-name**

This optional qualifier specifies the user's company name.

**/SIMPLE\_DISPLAY\_NAME=display-name**

This optional qualifier specifies the name to be displayed in the address book for the specified user.

**/MEMBER=members**

This optional qualifier specifies one or more distinguished names of those objects that are members of the specified group.

**/MEMBER\_OF=members**

This optional qualifier specifies the distinguished names of one or more groups to which this object belongs.

**/SECURITY**

This optional qualifier specifies that the new group is security enabled, i.e., it is a Security group. It cannot be specified if the /DISTRIBUTION qualifier is also specified. If neither qualifier is specified, /SECURITY is the default.

**/DISTRIBUTION**

This optional qualifier specifies that the new group will be a Distribution group only. It cannot be specified if the /SECURITY qualifier is also specified.

**/LOCAL**

This optional qualifier specifies that the new group will be a local group. It cannot be specified if the /GLOBAL and/or /UNIVERSAL qualifier(s) are also specified. If none of these qualifiers are specified, /LOCAL is the default.

**/GLOBAL**

This optional qualifier specifies that the new group will be a global group. It cannot be specified if the /LOCAL and/or /UNIVERSAL qualifier(s) are also specified.

**/UNIVERSAL**

This optional qualifier specifies that the new group will be a universal group. It cannot be specified if the /LOCAL and/or /GLOBAL qualifier(s) are also specified.

## 15.4 ADS SHOW

The ADS SHOW command displays information about various objects contained in the Active Directory.

### Format:

**ADS SHOW object name**

### Parameters:

#### **object**

This parameter determines the type of Active Directory object to be displayed. Its value may be specified as **USER** to display information about one or more Active Directory User objects; as **GROUP** to display information about one or more Active Directory Group objects; or as **OBJECT** if the optional **/FILTER** qualifier is present. If this parameter is omitted, and the **/FILTER** qualifier is not specified, *all* active directory objects will be displayed.

#### **name**

This optional parameter specifies the name of the object to be displayed. If omitted, information on all objects of the specified type will be displayed.

### Qualifiers:

#### **/CONTAINER=container-name**

This optional qualifier specifies the name of the Active Directory object which contains the target object.

#### **/FULL**

This optional qualifier will display completed information about the specified object(s). By default, only brief data is displayed.

#### **/FILTER=filter-string**

This optional qualifier specifies a search-filter string in LDAP format (e.g., (objectClass=computer) ).

#### **/OUTPUT=file-spec**

This optional qualifier directs the command output to the specified file.

#### **/PROPERTY=property-name**

This optional qualifier displays the specified object's property value only.

# 16 Language Statements

The following sections describe the language statements of XLNT.

## 16.1 = (Assignment Statement)

The assignment statement defines a symbolic name for values of varying data; or a new value is stored into an existing symbol.

### Format:

symbol-name =[=] expression

symbol-name[bit-position,size] =[=] replacement-expression

array-symbol = ( [index:] value,... )

### Parameters:

#### symbol-name

Specifies an alphanumeric string name for the symbol being assigned (or defined if implicitly used). The name can contain any alphanumeric character, the underscore (`_`), and the dollar sign (`$`). The name *must* begin with an alphabetic character (A-Z, a-z). A single equal sign (=) places the symbol in the local symbol table for the current command level. Two equal signs (==), place the symbol in the global symbol table.

#### expression

Provides the value for the symbol being defined. It can consist of a character string, an integer, an unsigned longword, an object, a symbol name, a lexical function, the returned value of a declared function or a combination of these entities. The expression is evaluated and the results assigned to the symbol. All literal character strings must be enclosed in quotation marks. If the expression contains a symbol, the expression is evaluated using the symbol's value.

### **[bit-position,size]**

Indicates that a binary overlay is to be inserted into a 32-bit integer value of the specified symbol. The current value of the symbol is used and then the number of bits (*size*) is replaced by the result of the replacement-expression. The bit position is the location relative to bit 0 at which the overlay is to occur.

If the symbol being overlaid is an integer, then the bit-position and size must be constrained to 32.

The brackets are required notation and no spaces are allowed between the symbol name and the left bracket. The bit-position and size must be expressed as integers.

### **array-symbol**

Specifies a symbol-name as defined above which must also be an array (numeric or associative). If the array is numeric indexed, it must also be one-dimensional. The array-symbol must not specify an element reference for its use in this context as an array value initializer.

### **( [index:] value,... )**

Indicates an initial value to be assigned to each or any specific element of the array. The array initializer form the assignment statement is intended to allow multiple values to be stored within an array in a single statement. The *index* parameter is optional for numeric indexed arrays. If omitted, the *value* parameter is stored, beginning with element 0 and continuing in a sequential ascending manner until all *value* parameters are exhausted in the list. The list is delimited with parenthesis "()". If the *index* parameter is specified, then the index is used as an element reference into the array and the value is stored in that array element. For a numeric indexed array the index must be numeric. For an associative indexed array the index must be a string, and the index is mandatory. The value stored must agree with the datatype of the array. Multiple values may be specified, comma delineated. If the index parameter is specified, a colon ":" must be specified between the index and the value.

### **Description:**

The assignment statement is the simplest method of both assigning a value to a symbol as well as creating the symbol itself. If the symbol doesn't already exist, XLNT will create the symbol and assign a datatype based on usage. To create a local symbol a single equal sign (=) is used. To create a global symbol a double equal sign (==) is used.

### **Examples:**

**\$ LS == "DIRECTORY"**

This simple assignment statement assigns the value "DIRECTORY" to the string symbol LS. Further the LS symbol becomes global through the use of the double equal sign (==) notation.

**\$ BELL[0,32]=7**

This example shows a simple numeric overlay. The value 7 is inserted into the 32-bit integer symbol BELL.

**\$ WEEKDAYS = ("SUN","MON","TUE","WED","THU","FRI","SAT")**

This example shows an array initializer statement. The one-dimensional string array WEEKDAYS will be initialized, starting at element index 0 with a value of "SUN" and the next value in the list will be stored at each succeeding element index.

**\$ WEEKDAYS = ("Sun":"Sunday","Mon":"Monday")**

This example shows how an associative array might be initialized. The key "Sun" will have the value "Sunday", the key "Mon" will have the value "Monday" and so on.

## 16.2 := (String Assignment)

The string assignment statement defines a symbolic name for a character string value.

### Format:

symbol-name :=[=] string

symbol-name[offset,size] :=[=] replacement-string

### Parameters:

#### symbol-name

Specifies an alphanumeric string name for the string symbol being assigned (or defined, if implicitly used). The name can contain any alphanumeric character, the underscore (`_`), and the dollar sign (`$`). The name *must* begin with an alphabetic character (A-Z, a-z). A single equal sign (`:=`) places the symbol in the local symbol table for the current command level. Two equal signs (`:=:`), place the symbol in the global symbol table.

#### string

Provides the string for the symbol being defined. The principal difference between the explicit string assignment statement and the standard assignment statement is that the string assignment permits the implicit definition of null strings. For example, if you wish to assign a symbol to the value of another symbol which has not been defined (as in the case of passing null parameters to command procedures), the standard assignment statement would return an error. The explicit string assignment would complete successfully, assigning a null value to the new symbol.

#### [offset,size]

Indicates that a portion of a string symbol value is to be overlaid with a replacement string. The size portion indicates the length of the replacement string and the offset portion indicates the relative position (first byte in string is considered offset 0) within the original string symbol value at which the replacement begins. The brackets are required notation, and no spaces are allowed between the symbol and the left bracket. Offset and size are integer values.

#### replacement-string

Specifies the string that is used to overwrite the string you are modifying. If the replacement-string is less than the size argument, the replacement-string is space filled from the right. If the replacement-string is longer than the size argument, the replacement-string is truncated from the right. The replacement-string is viewed by XLNT as a string literal. Symbols and/or lexicals can be used but explicit symbol substitution must be requested.

### Description:

The string assignment statement is similar to the assignment statement except that the symbol and value must be a string. The string assignment simplifies the string value in the expression in that a set of quotation marks are not required since the value must be a string. The other major purpose of the string assignment statement is that a sub-string replacement can be performed as described above.

### Examples:

```
$ TIME := SHOW TIME
```

This example shows the symbol TIME equated to the string symbol SHOW TIME.

```
$ FILE := TEST_FILE  
$ FILE[0,4] := MAST  
$ SHOW SYMBOL FILE  
FILE = "MAST_FILE"
```

This example illustrates a simple string replacement. First, the string symbol FILE is equated with "TEST\_FILE" (note that no quotation marks were required in the assignment because the right side is expected to be a string). Next the string replacement is performed, overlaying MAST across the first 4 bytes of the string symbol FILE's value (TEST\_FILE). The SHOW SYMBOL command confirms the overlay.

## 16.3 @ (Execute Procedure) Command

The @ command executes a command procedure.

### Format:

**@file-spec/qualifier [parameter[,...]]**

### Parameters:

#### file-spec

Specifies the file for the preceding command or the command procedure to be executed. Wildcard specifications are not permitted. If the file type is omitted, ".XCP" is assumed (for XLNT Command Procedure, e.g. @*mycommands* defaults to @*mycommands.xcp*) or .EXE for Run-Time usage. ASCII recommends that you omit the file type as this is required when using encoded scripts under the XLNT Run-Time License Edition.

#### parameter[,...]

Specifies from one to eight optional parameters to pass to the command procedure. The symbols (P1, P2,...P8) are assigned character string values in the order of entry. To omit a specific parameter when you need to pass other parameters after it, just specify two double quotes in a row, i.e. "".

### Description:

The @ command is used to execute an XLNT command procedure. The @ command may be invoked interactively or within a command procedure. Each time a new command procedure is invoked the command level is incremented by one. Conversely, each time a command procedure exits, the command level is decremented by one. The interactive command shell is considered level zero (0). The /OUTPUT qualifier is used to create an output file containing all output data that would have been displayed on the console window. When a command procedure is invoked up to eight (8) parameters may be specified as input arguments to the command procedure. By default when a command procedure is invoked using the @ command, the file type is assumed to be .XCP (except for the Run-Time License Edition which expects encoded scripts with a type of .EXE).

### Qualifier:

#### /OUTPUT=file\_spec

This qualifier allows the \$STDOUT handle to be temporarily assigned to the file specification shown. All procedure and program output to \$STDOUT will therefore be directed to the file specified. This qualifier, if used, must immediately come after the command procedure otherwise the qualifier will be interpreted as a parameter of the procedure. By default, no special procedure \$STDOUT redirection is performed.

### Examples:

#### \$ @TEST

This simple example executes the command procedure TEST.XCP in the current directory. No parameters have been passed.

#### \$ @TEST machine

Similar to the first example except that a single parameter *machine* is being passed. Please note that *machine* will be converted automatically to uppercase.

#### \$ @TEST "machine"

The same as the one above except that *machine* will not be converted to uppercase.

#### \$ @TEST/OUTPUT=TEST.LOG "machine"

The same as the one above except all procedure output will now be sent to TEST.LOG.

## 16.4 ! (Comment)

The ! command allows a user to insert comments into a command procedure. A comment can appear anywhere on an XLNT command line. All characters to the right of the comment will be ignored.

**Format:**

!

**Example:**

```
$ ! WRITE $STDOUT "hello"
```

This example shows how a command line is commented. The WRITE command will not be analyzed or executed due to the presence of the comment symbol.

## 16.5 ABORT Command

The ABORT statement combines the WRITE \$STDOUT statement and the EXIT command into one simple statement used for failing a command procedure. Note: ABORT only works when invoked within a command procedure. ABORT is ignored at the interactive command level.

### Format:

**ABORT string [status-code]**

### Parameter:

#### **string**

A string that is to be displayed to \$STDOUT.

#### **status-code**

An optional integer value that will represent the exit status of the current command procedure level. By default, the status-code is set to ERROR.

### Description:

The ABORT command is used to both exit a command procedure with an error status as well as display a message concerning the command procedure termination. Use of the ABORT command would be similar to issue a WRITE \$STDOUT as well as an EXIT error code. ABORT should be used to indicate an abnormal termination.

### Example:

```
$ ABORT "You failed to enter data"
```

This example displays a message indicating the reason for the procedure's failure. Since the ABORT statement is used, the procedure will exit with failure.

## 16.6 CALL Command

The CALL command transfers control to a labeled subroutine within a command procedure. The CALL command is similar to the @ (execute procedure) command in that a new command level is created.

### Format:

**CALL label [parameter[...]]**

### Parameters:

#### label

Specifies a label of 1 to 128 alphanumeric characters that appears as the first item on a command line. The label is a transfer point in which the first XLNT statement contains the SUBROUTINE statement.

#### parameter

Specifies from zero to eight optional parameters to pass to the command procedure. A null parameter ("" ) can also be passed. The parameters assign character string values to the symbols named P1, P2, and so on in the order of entry to a maximum of eight.

### Description:

The CALL statement provides the script writer with a method of executing a command procedure without requiring a separate file. This method allows the author to provide a single command procedure file yet still retain all the modularity that's desired in constructing a well thought out script. Also, since the label can be programatically derived the CALL statement can dispatch to any routine dynamically versus a static invocation. When a subroutine is invoked a new command level is created.

### Example:

```
$ CALL PROCESS_FILE "C:\XLNT\*. *"
```

```
.  
.  
.
```

```
$PROCESS_FILE: SUBROUTINE  
$ DIR 'P1  
$ ENDSUBROUTINE
```

This example shows how a command procedure can be modularized to have internal functions. The subroutine PROCESS\_FILE is invoked using the CALL command. A single parameter (the quoted string) is passed to the subroutine using the built-in symbol P1. The PROCESS\_FILE subroutine is denoted through the use of the SUBROUTINE and ENDSUBROUTINE statements. A new execution level is created through the use of the CALL command, so any local symbols in PROCESS\_FILE are not viewable by the calling procedure.

```
$ ROUTINE="DATE_CHECK, ACCT_CHECK, SERVICE_CHECK"
```

```
$ CALL '$ELEMENT(",",ROUTINE, J)
```

```
.  
.  
.
```

```
$DATE_CHECK: SUBROUTINE
```

```
$ ENDSUBROUTINE  
$ACCT_CHECK: SUBROUTINE
```

```
$ ENDSUBROUTINE
```

This example illustrates how a CALL can invoke one of a number of different routines based on program logic without requiring a separate CALL statement for each routine to be called. While the example doesn't show any input parameters

passed they could have been added and the number of parameters could be different (even varying datatypes) for each routine called.

**See also:**

**ENDSUBROUTINE SUBROUTINE**

## 16.7 CLS Command

This command clears the console window.

**Format:**

CLS

## 16.8 DEBUGBREAK Command

This command causes a debug breakpoint to occur when an XLNT Debugging session is active. **(Professional Edition Only)**

**Format:**

**DEBUGBREAK**

**Description:**

This command causes a breakpoint when the XLNT Debugger is present. If the Debugger is not present then execution continues.

## 16.9 DECK Command

The DECK command allows in-script input to a program up to the next XLNT command.

**Format:**

**DECK**

**Description:**

DECK provides a qualifier for ignoring the normal dollar sign (\$) command prompt as the end of in-script input and instead allows you to match on a pattern of your choosing. By default, DECK will terminate in-stream input when encountering a \$ EOD statement. Note: \$DECK is meant to be invoked within a command procedure and is ignored at the interactive command level.

**Qualifiers:**

**/DOLLARS[=string]**

This qualifier sets the EOF indicator to the desired quoted string if an input stream contains records that begin with \$ EOD. Please note that white-space on XLNT commands are ignored so \$EOD and \$ EOD are syntactically the same.

**Example:**

```
$ DECK  
$ WRITE $STDOUT "Test procedure"  
$ EOD
```

This example shows how the \$WRITE command is treated as in-stream data within the command procedure. Normally a line that began with a dollar sign would be as a command and signal the end of data. In this case using the \$ DECK command allows all data regardless of whether it begins with a dollar sign or not, to be treated as data. The \$EOD signals the end of the data stream.

**See also:**

[EOD](#)

## 16.10 DECLARE FUNCTION Command

The DECLARE FUNCTION command allows a user to declare a function for later execution. Function usage in XLNT is very similar to lexicals. This means that a function can return a value and can be used in the same contexts as XLNT built-in lexicals. Declared functions may be available as exported routines within a loaded DLL (however the DLL does not have to be loaded when this DECLARE FUNCTION command is invoked).

### Format:

**DECLARE FUNCTION datatype name library arguments,.... [options]**

### Parameters:

#### datatype

In addition to the list of [Supported Datatypes](#), the VOID datatype may be used to indicate that no value is passed back from the function.

#### name

A unique function name within a *library*. This function name must not conflict with another symbol name. The function name may be quoted if case sensitive.

#### library

The filename portion of the DLL (no path or UNC specification allowed).

#### arguments,...

(Optional) If specified, this list contains the datatype of each argument that may be specified with this function. See the [Supported Datatypes](#) list for valid datatypes.

#### options,...

(Optional) If specified, this list contains all options which further modify the handling or declaration of this function. Currently, the options available are GLOBAL, CDECL and STDCALL. By default, declared functions are local to the command procedure level they are invoked on. In addition, by default the calling mechanism used by XLNT when invoking user functions is CDECL. STDCALL is used when calling WIN32 API routines and must be specified.

### Description:

This statement provides the ability to declare and then later invoke any DLL exported API routine directly from an XLNT procedure. This means that if you either need to invoke a Win32 API or any API routine, you don't have to write small C or Basic programs to assist. To use the DECLARE FUNCTION statement you will need to know several things about the routine. They are: Routine Name, DLL that exports the routine, the calling standard used, and any arguments and their datatypes the routine expects, and whether the routine passes back a value and, if so, what the datatype is. If you're a C programmer think of the DECLARE FUNCTION statement as a C function prototype. For Win32 or other Microsoft provided API's, both on-line and Microsoft provided documentation indicate quite clearly all the items required. Most of the On-Line help for routines provide a "QuickInfo" section. That section will indicate the DLL or Import Library that exports the routine. All Win32 APIs use the STDCALL calling standard.

### Example:

```
$ DECLARE FUNCTION dword "InitiateSystemShutdownA" advapi32 -string,string,dword,dword,dword STDCALL
$ STATUS = F$LOADLIBRARY("c:\winnt\system32\advapi32.dll")
$ RESULT = InitiateSystemShutdownA(p1,p2,p3,p4,p5)
```

This example declares the Win32 API function "InitiateSystemShutdownA". Please note that many functions have both ASCII (A) or Unicode (W) variants. In this case, this example declares the ASCII version of this API routine. The function itself returns a DWORD status and consists of five (5) arguments (3 string and 2 dword). The routine is loaded in ADVAPI32 (note that both the path and the type are omitted) and uses the standard calling sequence (STDCALL). The lexical F\$LOADLIBRARY loads the DLL (the path is hardcoded so the example isn't confusing but a better method would be something like this: f\$loadlibrary (" ' ' f\$getvar("systemroot") \system32 \advapi32.dll") so the DLL could be loaded no matter where the system was located or named). The actual invocation is a straightforward assignment statement.

**See also:**

[F\\$LOADLIBRARY DECLARE SYMBOL](#)

## 16.11 DECLARE SYMBOL Command

This command allows a user to declare a symbol (or variable) formally and associate a datatype and possible initial value with the symbol.

### Format:

**DECLARE SYMBOL datatype name[=initial-value] [options,...]**

### Parameter:

#### datatype

This parameter indicates the datatype of the symbol being declared. See the list of [Supported Datatypes](#). A structure name may also be specified. See both the [Declaring Structures](#) discussion as well as the reference on the [STRUCTURE Command](#).

#### name[=initial-value]

This parameter contains the unique name of the symbol. This name must not conflict with any other symbol or function. To initialize the declared symbol, simply specify an equal sign (=) followed by the value to be used based on datatype. For example, a numeric datatype would be followed by an unquoted initial value that corresponded to the legal numeric range appropriate to the datatype chosen (radix operators may be used). A string datatype would contain a quoted initial string value.

#### options

(Optional) The following options may be specified; FIXED=numeric, for fixed-length strings, INITIALIZE and GLOBAL for globally declared symbols (by default, string symbols are dynamic and declared symbols are local). INITIALIZE will zero and space fill based on datatype. To declare an array use the following keywords; OCCURS=nnnn| VARIABLE, ASSOCIATIVE. The OCCURS keyword allows for a fixed maximum number of elements as specified by the number "nnnn" or a variable number of elements (no specified maximum) specified by the value VARIABLE. To create a multi-dimensional array specify each maximum separated by a colon (for example, 10:100, creates a two-dimensional array of 1000 elements). XLNT currently supports a maximum of three dimensions for numeric indexed arrays. The keyword ASSOCIATIVE indicates that the array is an associative rather than numeric indexed array (by default, numeric indexed arrays are created). Associative arrays are one-dimensional and contain a variable number of elements.

### Description:

This statement provides the script writer with the ability to declare symbols with datatypes instead of or in addition to using XLNT's Typeless facility. Note that when a symbol is declared the memory and resources are taken immediately. Once a symbol has been declared, its datatype cannot change unless the symbol is deleted. When a symbol is deleted, the symbol's declaration is also removed. If multiple options are specified, they should be comma delineated.

### Examples:

#### **\$ DECLARE SYMBOL STRING TEST**

This command explicitly creates a string symbol named TEST.

#### **\$ DECLARE SYMBOL INTEGER COUNT=1 GLOBAL**

This command explicitly creates the global integer symbol COUNT. The symbol is initialized to 1.

#### **\$ DECLARE SYMBOL RECORD INPUT INITIALIZE**

This command creates the symbol INPUT and is associated with the structure named RECORD. The symbol INPUT is then initialized based on the datatypes of the various field-names within RECORD.

#### **\$ DECLARE SYMBOL INTEGER RECORDSIZE OCCURS=40**

This command creates an integer array named RECORDSIZE which can have a maximum of forty (40) elements. The

array created is numerically indexed.

**\$ DECLARE SYMBOL STRING REC\_ARRAY ASSOCIATIVE**

This command creates an associative array of a variable number of elements named REC\_ARRAY.

**See also:**

[STRUCTURE](#)

## 16.12 DOS Command

The DOS command will execute a DOS command line.

**Format:**

**DOS [command]**

**Parameter:**

**command**

Specifies a DOS command line. XLNT will perform some processing on the command line, e.g. symbol substitution, removal of comments, etc. The command line will be passed to the DOS command interpreter for syntax checking and execution. If the parameter is omitted, XLNT will place the session into the context of the DOS command interpreter and will remain there until the user enters the EXIT command.

Hint: If transparent foreign command recognition is enabled with SET PREFERENCES there is little need for this command except: (1) When the DOS command conflicts with an XLNT command, or (2) An internal DOS command needs to be invoked.

***Note that the DOS interpreter is invoked as a subprocess of the XLNT process. Any actions that change the state of the subprocess, such as changing the directory, will not be reflected when control is returned to the XLNT process.***

**Example:**

```
$ NAME = "Screen Name"  
$ DOS TITLE 'NAME'
```

This command lets you change the title of the XLNT Console Window. Note that even though the TITLE command will be processed through the DOS command interpreter (actually CMD.EXE), XLNT will still perform symbol substitution.

***Note: Due to differences in the implementation of MS-DOS between Windows, XLNT is forced to work differently. For Windows NT, DOS can be supported using the same console window.***

## 16.13 ENDFOR Statement

The ENDFOR statement ends a FOR loop block. You cannot issue this statement interactively.

**Format:**

**ENDFOR**

**Example:**

```
$ FOR (COUNT=1, COUNT .LE. 2, COUNT=COUNT+1)  
$ WRITE $STDOUT "Loop count is "count"  
$ ENDFOR
```

Loop count is 1

Loop count is 2

In this example the simple FOR loop block is terminated with the ENDFOR statement.

**See also:**

[FOR](#)

## 16.14 ENDFOREACH Statement

The ENDFOREACH statement ends a FOREACH loop block. You cannot issue this statement interactively.

Format:

```
ENDFOREACH
```

Example:

```
$ foreach value in Array2  
$   write $stdout "Next value is "value"  
$ endforeach
```

In this example the simple FOREACH loop block is terminated with the ENDFOREACH statement.

**See also:**

[FOREACH](#)

## 16.15 ENDSTRUCTURE Statement

The ENDSTRUCTURE statement ends a STRUCTURE definition.

**Format:**

**ENDSTRUCTURE**

**Example:**

```
$ STRUCTURE TEST
  STRING NAME           30
  STRING ADDRESS        30
  STRING CITY           10
  STRING STATE          2
  STRING ZIP_CODE       5
ENDSTRUCTURE
```

This example shows how the ENDSTRUCTURE statement ends the formal structure definition. While the example doesn't show a dollar sign in front of each line in the structure definition, a dollar could have been placed as the first character with no ill effect.

**See also:**

[STRUCTURE](#)

## 16.16 ENDSUBROUTINE Command

The ENDSUBROUTINE command defines the end of a subroutine within a command procedure. You cannot issue this statement interactively.

**Format:**

**ENDSUBROUTINE**

**Example:**

```
$ CALL PROCESS_FILE "C:\XLNT\*.*)"
.  
.  
.  
$PROCESS_FILE: SUBROUTINE  
$ DIR 'P1  
$ ENDSUBROUTINE
```

This example shows how a command procedure can be modularized to have internal functions. The subroutine PROCESS\_FILE is invoked using the CALL command. A single parameter (the quoted string) is passed to the subroutine using the built-in symbol P1. The PROCESS\_FILE subroutine is denoted through the use of the SUBROUTINE and ENDSUBROUTINE statements.

**See also:**

[CALL SUBROUTINE](#)

## 16.17 ENDUNTIL Statement

The ENDUNTIL statement ends an UNTIL loop block. You cannot issue this statement interactively.

**Format:**

**ENDUNTIL**

**Example:**

```
$ DONE = 1
$ UNTIL DONE .GT. 5
$   DONE = DONE + 1
$ ENDUNTIL
```

This example shows a simple UNTIL loop with the ENDUNTIL terminating the loop block.

**See also:**

[UNTIL](#)

## 16.18 ENDWHILE Statement

The ENDWHILE statement ends a WHILE loop block. You cannot issue this statement interactively.

**Format:**

**ENDWHILE**

**Example:**

```
$ DONE = 1
$ WHILE DONE .LT. 5
$   DONE = DONE + 1
$ ENDWHILE
```

This example shows how the ENDWHILE ends the WHILE loop block.

**See also:**

[WHILE](#)

## 16.19 EOD Statement

The \$ EOD command is used to delimit in-script input. The EOD command must be used to end in-script input begun by a DECK command.

**Format:**

```
$ EOD
```

**Example:**

```
$ DECK  
$ WRITE $STDOUT "Test procedure"  
$ EOD
```

This example shows how the \$EOD statement delimits the \$DECK statement. Normally, in-script data is terminated when a \$ (dollar sign) is encountered in the command procedure. With \$DECK and \$EOD, in-script data can contain any characters. \$DECK/\$EOD is particularly useful when a command procedure needs to generate XLNT statements (since XLNT statements typically begin with a dollar sign (\$)).

**See also:**

[DECK](#)

## 16.20 EXIT Command

The EXIT command terminates processing of a command procedure or subroutine and returns control to the calling command level.

**Format:**

**EXIT [status-code]**

**Parameter:**

**status-code**

Returns the numeric status code of the exiting procedure or subroutine. This value is placed into the global symbol \$STATUS.

**Examples:**

**\$ EXIT**

This command simply exits the current command procedure and assumes a success status.

**\$ EXIT 4**

This command exits the command procedure and also explicitly passes back an exit status of 4. It is up to the caller to determine the significance of the status returned.

## 16.21 FOR Command

The FOR command provides the ability to loop within a block of XLNT command/statements and to test an expression during each iteration.

### Format:

FOR ([*init-command*], [*leave-expression*], [*iterate-command*])

### Parameters:

#### **init-command**

Any valid XLNT command. Normally an assignment statement is used to initialize a symbol that will be used to control the loop. The *init-command* is executed first and only once.

#### **leave-expression**

Defines the test to be performed. The expression can consist of one or more numeric constants, string literals, symbolic names or lexical functions separated by logical, arithmetic or string operators. The expression is evaluated prior to entering the body of the loop.

#### **iterate-command**

Any valid XLNT command. Normally an assignment statement which modifies the symbol tested in the *leave-expression* is used. The *iterate-command* is executed at the conclusion (or bottom) of each loop. The *leave-expression* would then be tested to determine whether the loop should continue.

### Description:

The FOR statement is used to iterate a loop block. A loop block is denoted between the FOR statement and an ENDFOR statement. The *init-command*, if present, is executed first. The *leave-expression*, if present is evaluated. The loop block is iterated if the expression evaluates to true. If the expression is false then control is passed to the statement following the related ENDFOR. . The *iterate-command*, if present, is executed at the bottom of the loop. Please note that the parameters of the FOR statement must be contained within parenthesis. Any or all parameters may be omitted through the use of comma's or the closing parenthesis (for example, FOR (,,)). A FOR statement with all parameters omitted will produce an endless loop. The LEAVE statement may be used to terminate the loop.

The FOR statement cannot be executed at command level 0 (interactive).

### Example:

```
$ for (proj_count=0,,proj_count=proj_count+1)
$   proj_file=f$search ("...\*.mak",context)
$   proj_filename=f$parse(proj_file,,"NAME",)
$!
$   if proj_file .eqs. "" then leave
$   .
$   .
$   .
$ endfor
$ write $stdout "Project files: "proj_count"
$ endif
```

This example illustrates a use of the FOR statement to count iterations. *proj\_count* is set to zero and for each iteration the symbol is incremented. Note the use of LEAVE to terminate the FOR loop since the FOR statement does not provide a *leave-expression*.

### See also:

ENDFOR UNTIL WHILE

## 16.22 FOREACH Command

The FOREACH command iterates through a numerically-indexed array or ActiveX collection.

Format:

```
FOREACH iteration-variable IN collection
```

Parameters:

iteration-variable

The symbol that represents an individual element of the array or object collection. Its datatype will be set to that of the collection's elements.

collection

The symbol that represents the XLNT array or ActiveX collection object.

Description:

The FOREACH command retrieves an iteration variable for each element in the specified collection. If a variable is successfully retrieved, all the statements within the FOREACH loop block will be executed. A loop block consists of all the statements contained between a FOREACH and a matching ENDFOREACH statement. If all the elements in the collection have been iterated, or the collection is empty, control passes to the first statement after the ENDFOREACH command.

The FOREACH statement cannot be executed at command level 0 (interactive).

Example:

```
$ declare symbol integer IntArray occurs = variable
```

```
.  
. .  
. . .
```

```
$ TotValue = 0
```

```
$ foreach IntValue in IntArray
```

```
$   write $stdout "Next value is "IntValue"
```

```
$   TotValue = TotValue + IntValue
```

```
$ endforeach
```

```
$ write $stdout "Total Value is "TotValue"
```

This example first declares an integer array named IntArray. Sometime later in the procedure, presumably after IntArray has been populated, an integer symbol named TotValue is set to zero. At that point, a FOREACH loop is executed,

extracting each consecutive value from IntArray and adding it to TotValue. When all values in the array have been retrieved, the total value is displayed.

## 16.23 GOSUB Command

The GOSUB command transfers control to a labeled routine within a command procedure *without* creating a new procedure level. When the routine returns, control is returned to the statement following the GOSUB.

**Format:**

**GOSUB label**

**Parameter:**

**label**

Specifies a label of 1 to 128 alphanumeric characters that appears as the first item on a command line.

**Description:**

GOSUB is used to transfer control to a labeled routine within a command procedure without creating a new command level. GOSUB cannot be invoked interactively (at command level 0). Since GOSUB transfers control to a label you should think of GOSUB as a call to a "local" subroutine. The GOSUB statement can be nested up to 16 levels per command level. The major benefit that GOSUB offers versus CALL is that all local symbols are available to GOSUB since a new command level is not created. A labeled routine that is the target of a GOSUB must issue a RETURN to return to the next sequential statement after the GOSUB.

**Example:**

```
$ GOSUB GET_INPUT
$ WRITE $STDOUT "Your name is "ANS"
$ EXIT
$GET_INPUT:
$ INQUIRE ANS "Please enter your name"
$ RETURN
```

This code sequence describes the use of the GOSUB statement. While similar to the CALL command, several differences become obvious. First, no parameter passing is allowed since a new execution level is not created. Therefore the local routine GET\_INPUT has complete visibility to any local symbols created by the caller and vice-versa. The GOSUB statement is useful when you want to create discrete code but have no requirement for parameter passing.

## 16.24 GOTO Command

The GOTO command transfers control to a labeled statement in a command procedure.

**Format:**

**GOTO label**

**Parameter:**

**label**

Specifies a label of 1 to 128 alphanumeric characters that appears as the first item on a command line.

**Description:**

With all of the looping and block structured syntax available in XLNT, a GOTO statement really isn't needed. That said the GOTO statement is used to transfer control to any label within the current command procedure level. You cannot issue the GOTO statement interactively; it must be present within a command procedure. The label target can appear anywhere within the current command procedure. If a GOTO is issued within a loop block (FOR, WHILE, UNTIL) and the label appears within the block then a normal transfer occurs. However, if the label is outside the loop block, then the loop is implicitly terminated and the transfer occurs. Script writers should understand this side-effect and either avoid using GOTO within loop blocks or code their scripts with this effect in mind. If duplicate labels appear in the procedure prior to the GOTO statement then control will transfer to the last duplicate label encountered. If duplicate labels appear after the GOTO statement then control will transfer to the first label encountered. The GOTO statement is useful in one important context within XLNT and that's the ON statement. A typical ON statement will contain a GOTO to transfer control to the error handler.

**Example:**

```
$ GOTO LABEL
```

```
.
```

```
$LABEL:
```

```
$ A="WERE_DONE"
```

```
.
```

```
$WERE_DONE:
```

```
.
```

```
$ GOTO 'A
```

These examples show the simple GOTO facility. The parameter of the GOTO must be a label (no restrictions are present on forward references). In the second example, a string symbol is used to represent the label WERE\_DONE. In this case, an explicit symbol substitution is made.

## 16.25 HELP Command

**XLNT**<sup>®</sup> is a software product, developed by Advanced Systems Concepts, Inc. XLNT provides a powerful, but easy to use, scripting language to facilitate command-line and batch processing interfaces for the users of the Microsoft Windows 2000/2003/XP operating systems.

**HELP** starts the WinHelp version of the XLNT Help facility. Additional information on WinHelp is available by clicking on the Help button (which appears on top with the Menu bar). XLNT Windows Help is executed asynchronously meaning that you don't have to leave help in order to continue using XLNT. You can even terminate XLNT and continue using the On-Line Help.

**Format:**

**HELP [WINDOW]**

**Example:**

**\$ HELP**

## 16.26 IF Command

Tests the value of an expression and, depending on the syntax specified, executes the following:

### Format 1:

One command following the THEN keyword if the expression is true, for example.

```
$ IF expression THEN command
```

### Format 2:

Multiple commands following the \$THEN command if the expression is true. One or more commands following the \$ELSE command if the expression is false, for example.

```
$ IF expression
```

```
$ THEN [command]
```

```
command
```

```
.
```

```
.
```

```
.
```

```
[$ELSE] [command]
```

```
command
```

```
.
```

```
.
```

```
.
```

```
$ ENDIF
```

### Parameters:

#### expression

Defines the test to be performed. The expression can consist of one or more numeric constants, string literals, symbolic names, or lexical functions separated by logical, arithmetic, or string operators.

#### command

Specifies the XLNT command or commands to be executed.

### Description:

The IF statement provides decision making capability within the XLNT scripting language. IF provides a mechanism for issuing an expression and then testing its value for the purposes of determining whether an expression tests as true or false.

### Examples:

```
$ QUERY = "Y"  
$ IF QUERY THEN WRITE $STDOUT "TRUE"
```

```
$ IF QUERY  
$ THEN  
$     WRITE $STDOUT "TRUE"  
$ ELSE  
$     WRITE $STDOUT "FALSE"
```

## **\$ ENDIF**

These commands depict simple IF statements. In the first IF statement, a single XLNT command is executed if the expression is true (which it is since QUERY is logically true). The second IF statement is more conventional in that multiple XLNT commands may be executed if true (THEN) or false (ELSE). The ENDIF statement ends a THEN or ELSE block. The indentation of the commands under the THEN and ELSE blocks are for readability and are not syntactically required. You may also nest IF statements.

The \$STATUS symbol is processed in a different manner by the IF statement than other symbols. \$STATUS represents the exit status of the last invoked program or statement (other than lexicals or user-defined functions). Programs which use the Win32 method of providing both a status and severity indication can be checked with \$STATUS to determine whether the program performed its intended operation. While \$STATUS is a 32-bit integer value for Win32 systems and is usually non-zero, an XLNT IF statement will automatically determine the severity of the exit status code and provide a TRUE indication if the status is successful or a FALSE indication if the status shows failure.

## 16.27 INQUIRE Command

The INQUIRE command reads a value from the console window and assigns it to a symbol.

### Format:

**INQUIRE** *symbol-name* [*prompt-string*]

### Parameters:

#### **symbol-name**

Specifies a symbol consisting of from 1 to 128 alphanumeric characters.

#### **prompt-string**

Specifies the prompt to be displayed when the INQUIRE command is executed.

### Description:

The INQUIRE statement is used to prompt and request a response. The response is always treated as a string. Data is converted, by default, to uppercase (this can be altered via the /NOUPPERCASE qualifier).

### Qualifiers:

#### **/[NO]ECHO**

By default, the user is able to see keystrokes entered. When /NOECHO is specified, the user's keystrokes are not echoed back. This qualifier is useful when passwords are prompted.

#### **/ERROR=value**

Specifies the error label should the INQUIRE command timeout.

#### **/GLOBAL**

Specifies that the symbol name is that of a global symbol. The default is a local symbol.

#### **/[NO]PUNCTUATION=value**

Indicates the punctuation added to the prompt-string. The string “: “ will be added if the qualifier and/or value is omitted. No punctuation is added if the qualifier is negated.

#### **/TIMEOUT=value**

Specifies the time, in seconds, at which the pending INQUIRE command will expire. The default is infinite.

#### **/[NO]UPPER**

By default, data entered is converted to uppercase. To avoid converting case, specify /NOUPPER.

### Example:

```
$ INQUIRE CHECK "Enter Y[ES] to continue"  
$ IF .NOT. CHECK THEN EXIT
```

The INQUIRE command displays the following prompting message at the terminal:

```
Enter Y[ES] to continue:
```

The INQUIRE command prompts for a value, which is assigned to the symbol CHECK. The IF command tests the value assigned to the symbol CHECK. If the value assigned to CHECK is true (that is, an odd numeric value, a character string that begins with a T, t, Y, or y, or an odd numeric character string), the procedure continues executing.

### See also:

[READ](#)

## 16.28 ITERATE Command

The ITERATE command passes control to the next iteration of the FOR, WHILE and UNTIL block in which it appears, bypassing any remaining commands within the loop block.

**Format:**

ITERATE

**Example:**

```
$ FOR (I=1, I .LT. 10, I=I+1)
$ IF I .EQ. 2 THEN ITERATE
$ WRITE $STDOUT "Loop count is ' ' I "
$ ENDFOR
```

This example shows a simple FOR loop which counts from 1 to 9. Note the IF statement. When the count is 2, the WRITE statement isn't processed due to the ITERATE command. All further processing for that specific iteration is omitted. If the intention was to terminate the loop block entirely, the LEAVE command would be used.

**See also:**

FOR UNTIL WHILE

## 16.29 KILL Command

The KILL command will terminate an active Windows process on a local or remote machine.

**Format:**

**KILL process-id**

**Parameter:**

**process-id**

The numeric hexadecimal process-id that uniquely identifies every Windows process.

**Qualifiers:**

**/ON=computer-name**

A character string that names the target computer. If omitted, the process will be terminated on the local computer.

**/USERNAME=username**

For a remote kill operation, your account must have administrative privileges on the remote system. If it does not, then a username/password must be supplied for a suitably privileged account on that system.

**/PASSWORD=password**

Password string for remote administrator.

**Example:**

```

XLNT Session for User: JAlfrunT
Licensed to: XLnt User (1 license)
$ SHOW SYSTEM
Windows NT 5.0 on ATHENA 3/20/2006 2:42:50 PM Uptime: 4 22:54:51.999
Pid Process Handles Threads Prio Work Set PageFlts Cpu
00000000 Idle 0 1 0 16384 1 4 17:46:54.640
00000008 System 204 46 8 221184 26502 0 00:01:20.890
00000090 SMSS 33 6 11 393216 629 0 00:00:00.656
000000A8 CSRSS 673 13 13 2838528 95375 0 00:08:04.234
000000A4 WINLOGON 460 18 13 11718656 936493 0 00:00:46.937
000000D8 SERVICES 635 37 9 8474624 25277 0 00:00:12.046
000000E4 LSASS 389 19 9 3305472 150956 0 00:03:08.187
0000019C svchost 408 11 8 4939776 6223 0 00:00:01.703
000001B8 spoolsv 187 12 8 7180288 9002 0 00:00:05.453
000001F8 msdtc 213 23 8 6053888 1705 0 00:00:00.281
00000274 DWRCS 111 8 8 4653056 6549 0 00:00:04.562
00000280 svchost 444 27 8 8097792 4046 0 00:00:00.656
00000318 FrameworkServic 260 11 8 6983680 46985 0 00:00:02.125
00000318 explorer 494 18 8 2306048 414362 0 00:04:49.734
00000334 McShield 198 19 13 22900736 953443 0 00:04:09.796
0000034C UsTaskMgr 130 11 8 311296 48669 0 00:00:00.140
00000364 naPrdMgr 107 4 8 1134592 3529 0 00:00:00.046
00000390 ndm 118 5 8 3653632 1226 0 00:00:00.640
000003B4 omtsreco 60 4 8 6590464 1617 0 00:00:00.093
000003FC shstat 62 6 8 602112 95427 0 00:00:00.265
000003E0 UpdaterUI 97 4 8 380928 1314796 0 00:00:00.531
0000041C atiptaxx 83 2 8 3866624 1038 0 00:00:00.203
00000444 Desk95 67 1 8 2961408 720 0 01:25:38.281
00000484 AcroTray 18 1 8 1400832 342 0 00:00:00.015
000002D0 regsvc 77 3 8 3022848 754 0 00:00:00.046
000003DC mstask 131 7 8 5439488 1518 0 00:00:00.109
000004C8 userdump 47 4 8 1183744 290 0 00:00:00.015
000004FC WinMgmt 309 10 8 2502656 89526 0 00:00:44.843
00000300 svchost 414 7 8 13750272 89811 0 00:00:17.937
0000050C heremote 161 8 8 8679424 4761 0 00:00:19.359
0000053C mqsvc 204 22 8 6008832 7739 0 00:00:00.374
000004DC wuauclt 164 3 8 5443584 1339 0 00:00:00.171
000006D8 LOCATOR 56 3 8 3854336 1237 0 00:00:00.109
00000410 abatadmin 315 6 8 28913664 62512 0 00:03:25.499
00000944 OUILOOK 567 16 8 8728576 176253 0 00:40:51.124
0000031C CMD 21 1 8 49152 648 0 00:00:00.031
00000750 scardsvr 55 5 8 1409024 396 0 00:00:00.031
00000920 XLntCli 109 1 8 77824 3517 0 00:00:00.234
000008E0 mstsc 196 12 8 1167360 82199 0 00:00:49.593
000009D0 devenu 419 13 8 46669824 113731 0 00:00:41.515
00000954 AbatEAgent 331 21 8 15642624 9926 0 00:00:06.109
00000754 AbatJss 572 23 8 17383424 13810 0 00:00:19.859
00000790 devenu 299 10 8 5701632 31761 0 00:00:03.812
00000980 XLntCli 109 1 8 77824 20508 0 00:00:00.906
000006E4 WINWORD 272 6 8 13352960 40827 0 00:00:15.109
000007D4 autorun 44 2 8 3702784 1215 0 00:00:00.078
00000824 _autorun 127 4 8 8601600 2369 0 00:00:00.265
00000834 msisexec 103 4 8 7266304 8262 0 00:00:12.562
00000830 XLntCli 44 1 8 69632 705 0 00:00:00.015
0000075C XLntCli 50 2 8 2609152 772 0 00:00:00.031
00000900 XLntCli 50 2 8 2605056 771 0 00:00:00.031
00000738 XLntCli 50 2 8 2605056 771 0 00:00:00.031
00000904 XLntCli 101 2 8 3047424 1193 0 00:00:00.062
00000000 _Total 10848 508 0 330551296 4914033 4 20:23:31.999
$

```

\$ KILL 85

This command kills process 85 (XLntCli).

## 16.30 LEAVE Command

The LEAVE command is used to terminate the current loop block.

**Format:**

**LEAVE**

**Example:**

```
$ FOR (COUNT=1,,COUNT=COUNT+1)
$ READ DATA_FILE INPUT_RECORD
$ IF INPUT|TAG .EQ. 2 THEN LEAVE
$ WRITE $STDOUT "Loop count is ' ' COUNT' "
$ ENDFOR
```

This example shows an interesting use of the FOR loop while demonstrating the LEAVE command. Note that the FOR statement contains both an *init\_statement* as well as an *iterate\_statement* but no *expression* is present. This means that the loop is an endless loop. The procedure reads data from a file into a symbol with an implied structure containing a field TAG. According to the procedure logic when TAG is a 2, the loop should end. That is the purpose of the LEAVE statement. To cause a premature (and programmed) ending of an active loop block. Note that if this loop block was nested within another loop block, only the inner loop block would be ended.

**See also:**

FOR UNTIL WHILE

## 16.31 LOGOUT Command

The LOGOUT command terminates an XLNT session.

**Format:**

**LOGOUT**

## 16.32 ON Command

The ON command performs a specified action when a command or program executed within a command procedure encounters an error condition.

### Format:

**ON condition THEN command**

### Parameters:

#### **condition**

Specifies the severity level of the error. One of the following keywords can be specified: WARNING, ERROR.

#### **command**

Specifies the XLNT command to be executed. The special command CONTINUE may also be specified. This indicates that processing should continue after the command which caused the error condition.

### Description:

The ON statement provides a method for establishing an error handler rather than requiring that the \$STATUS symbol be checked for error on each operation. When an ON condition is triggered, the command portion of the ON statement is executed. The ON statement then reverts to the default condition of ON WARNING THEN CONTINUE. An ON statement must therefore be issued again after an error condition is triggered. This action prevents an ON error condition from inadvertently looping. The ON statement also supports the CONTROL\_C condition. This condition is triggered when a CONTROL/C or CONTROL/Break is pressed. ON WARNING is triggered when a warning or an error occurs. ON ERROR is triggered only when an error condition is raised.

To disable ON error handling, use the SET NOON command.

To enable DOS/CMD program's to also use ON error handling, use the SET DOSERROR command.

### Example:

```
$ ON WARNING THEN GOTO ERROR_ROUTINE  
$ INQUIRE/TIMEOUT=10 ANS "Password, please"
```

The ON statement indicates that should an error condition be raised that does not already have a labeled condition associated with it, control should be transferred to the label ERROR\_ROUTINE. So if the INQUIRE times out, control will be transferred to the specified label. Note that if the /ERROR qualifier on the INQUIRE command had been used, control would transfer to that label instead.

## 16.33 RECALL Command

The RECALL command displays up to *nnn* previously entered commands on the screen for subsequent execution, where *nnn* is a user-settable value (default is 100).

### Format:

**RECALL [command-specifier]**

### Parameters:

#### **command-specifier**

Specifies the number or the first several characters of the command to be recalled. The specified characters should be unique. If they are not, the command displays the most recently entered command that matches the characters. The number of the command can also be entered, where 1 is the most recent command and *nnn* is the least recent command. The RECALL command itself is never saved or assigned a number.

### Description:

Whenever commands are interactively entered for XLNT, XLNT saves these commands in a special area called a "RECALL buffer". When you need to re-execute or edit a previously entered command you can recall the command in a variety of ways. Typically commands are recalled using the arrow keys. By default, a total of 20 commands are stored for recall purposes. To recall a specific command or to obtain a list of previously entered commands, the RECALL command is used. RECALL is expected to be used interactively and cannot appear in a command procedure. Entering RECALL/ALL displays a list of all entered commands from the latest to the earliest entered. RECALL followed by a number indicates that a specific command as numbered by the /ALL qualifier is to be recalled. RECALL followed by an alphanumeric entry indicates that a string search of the recall buffer from the latest to the earliest is to occur. The first match, regardless of ambiguity, presents that command to the command buffer. For security purposes you may want to enter RECALL/ERASE to clear the recall buffer when system passwords have been entered.

### Qualifiers:

#### **/ALL**

Specifies that all the command in the recall buffer, along with their assigned numbers, be displayed.

#### **/ERASE**

Erases the contents of the recall buffer.

### Example:

#### **\$ RECALL/ALL**

```
1 SHO SERV
2 SHO SYS/ON=HERA
3 SHO MEM
4 SHO SYS
5 DIR
```

This example recalls all previously entered commands.

## 16.34 RETURN Command

The RETURN command terminates a GOSUB routine and returns control to the command following the GOSUB command.

**Format:**

**RETURN [status-code]**

**Parameter:**

**status-code**

Defines an integer value or expression that gives the exit status of the subroutine.

**Description:**

The RETURN command is used to delimit a labeled routine that was the target of a GOSUB. In other words, RETURN is to GOSUB what ENDSUBROUTINE is to SUBROUTINE. When a RETURN is encountered for a previously GOSUB routine, control transfer back to the next sequential statement following the GOSUB. You cannot issue GOSUB or RETURN interactively.

**Example:**

```
$ GOSUB TIME
$ EXIT
$TIME:
$ SHOW TIME
$ RETURN
```

This example shows how the GOSUB statement transfer control to a labeled routine named TIME. That routine executes until a RETURN statement is encountered. Control then transfers back to the statement following the GOSUB which, in this case, is EXIT.

**See also:**

[GOSUB](#)

## 16.35 RUN Command

The RUN command executes a user-specified program or file as a subprocess of the XLNT process. By default, XLNT will wait for the program to complete execution before prompting for the next command. Also by default, the target program will share the same console window as XLNT. Qualifiers are available to alter either of these default conditions.

### Format:

**RUN file-name**

### Parameter:

**file-name**

Specifies the name of the image to be executed. Wildcards may not be used. The file type defaults to .EXE.

### Qualifiers:

**/ARGS=(arg[,...])**

Specifies arguments to be passed to the target program. Each argument must be separated by a comma. XLNT will perform explicit symbol substitution on an argument before it is passed to the program.

**/DETACHED**

Specifies that the image is to be executed as a detached process, in its own console window.

**/DIRECTORY=directory\_name**

Specifies the default directory for a detached process.

**/ERROR=file\_name**

Specifies the name of a file that will be assigned to the standard error handle for a detached process.

**/INPUT=file\_name**

Specifies the name of a file that will be assigned to the standard input handle for a detached process.

**/NEW\_CONSOLE**

Specifies that the new process will receive its own console window. The default is to share the XLNT console window.

**/OUTPUT=file\_name**

Specifies the name of a file that will be assigned to the standard output handle for a detached process.

**/PRIORITY=[HIGH | IDLE | REAL | NORMAL]**

This qualifier specifies the scheduling priority that will be assigned to the subprocess. The default is normal scheduling priority.

**/[NO]WAIT**

Specifies whether the XLNT process should wait for the user program to complete before prompting for the next command. The default is to wait.

### Description:

The RUN command is used to execute files. When the file is an executable image (type of EXE), the program is run as a subprocess unless the /DETACH qualifier is used. When the file is not an executable image, XLNT will search for a known file association and then run the associated program passing the file specified as a RUN argument. For example, RUN TEST.DOC would actually start Microsoft Word and pass TEST.DOC as a file to be opened. By default, XLNT will wait until the executed program exits. Specifying /NOWAIT causes XLNT to run in parallel with the executed program (desirable when the program is a Windows GUI program). You can redirect the standard file

handles for a running program through the use of the /INPUT, /OUTPUT and /ERROR qualifiers.

**Example:**

**\$ RUN REGEDT32**

This command executes the Windows NT Registry Editor.

**\$ RUN/OUT=DEVICE.LIS/ARG=("@DIR.XCP")/NOWAIT XLNTCLI**

This command actually runs the XLNT command processor itself and submits a command procedure (DIR.XCP) for processing. The "at" sign is very important since any valid XLNT command can be executed through this mechanism. The output of the XLNT session is redirected to DEVICE.LIS. The NOWAIT qualifier indicates that XLNT should not wait for this command to complete.

**See also:**

[SPAWN](#)

## 16.36 SPAWN Command

The SPAWN command executes the specified command within its own subprocess that runs independently of the XLNT process.

### Format:

**SPAWN [command]**

### Parameter:

**command**

Specifies any valid XLNT command. If the parameter is supplied, a new process is created that executes this command and returns control to the spawning process. If the parameter is omitted, a new process is created that prompts for a command. This new process will continue to prompt for and execute commands until the user enters an XLNT *LOGOUT* command.

### Qualifiers:

**/DETACHED**

Specifies that the command is to be executed as a detached process, in its own console window.

**/DIRECTORY=directory\_name**

Specifies the default directory for a detached process.

**/ERROR=file\_name**

Specifies the name of a file that will be assigned to the standard error handle for a detached process.

**/INPUT=file\_name**

Specifies the name of a file that will be assigned to the standard input handle for a detached process.

**/NEW\_CONSOLE**

Specifies that the command will receive its own console window. The default is to share the XLNT console window.

**/OUTPUT=file\_name**

Specifies the name of a file that will be assigned to the standard output handle for a detached process.

**/PRIORITY=[HIGH | IDLE | REAL | NORMAL]**

This qualifier specifies the scheduling priority that will be assigned to the subprocess. The default is normal scheduling priority.

**/[NO]WAIT**

Specifies whether the XLNT process should wait for the user program to complete before prompting for the next command. The default is to wait.

### Example:

**\$ SPAWN/NOWAIT @TEST**

This command spawns a process asynchronously and doesn't wait for it to complete. This command is very useful when performing I/O intensive operations or launching Windows oriented tasks.

### See also:

**RUN**

## 16.37 STRUCTURE Command

The STRUCTURE statement defines the beginning of a structure definition. The definition is terminated with an ENDSTRUCTURE statement. A structure consists of one or more fields which become related as a result of being specified within a single structure. Structures are referenced through the use of the DECLARE SYMBOL command.

### Format:

**STRUCTURE** structure-name [option]

datatype field-name [string-length]

·  
·  
·

**ENDSTRUCTURE**

### Parameters:

#### structure-name

A unique name consisting of characters that would be valid for a symbol name. However, the length of a structure name is limited to 31 characters.

#### option

(Optional) At present the only option that may be specified is GLOBAL. By default, structure definitions are local (within the current command level this statement is executed at).

#### datatype

For each field-name within the structure, this parameter represents a valid datatype as listing in the section named [Supported Datatypes](#) **or** another structure name.

#### field-name

A unique field-name (using the rules governing construction of a valid symbol name) within this structure.

#### [string-length]

This parameter, required if the datatype is a STRING, indicates the length of this string field.

### Example:

#### \$ STRUCTURE TEST

```
STRING NAME          30
STRING ADDRESS       30
STRING CITY           10
STRING STATE          2
STRING ZIP_CODE       5
```

ENDSTRUCTURE

#### \$ DECLARE SYMBOL TEST TEST\_SYMBOL

This example shows how a simple structure named TEST is defined. TEST consists of five string fields. The DECLARE SYMBOL command actually associates a symbol (named TEST\_SYMBOL) with the structure and memory is allocated (note how the structure name is placed in the *datatype* position of the DECLARE SYMBOL command). To access any of the fields within the TEST structure for TEST\_SYMBOL, the vertical bar (|) is used. For example, TEST\_SYMBOL|NAME access the NAME field within the TEST\_SYMBOL.

#### \$ STRUCTURE LOCATION

```
STRING ADDRESS          30
STRING ADDRESS2        30
```

ENDSTRUCTURE

```
$ STRUCTURE MIXED  
  STRING NAME          30  
  INTEGER AGE  
  LOCATION MYLOCATION  
  BYTE   FLAG  
ENDSTRUCTURE
```

#### **\$ DECLARE SYMBOL MIXED TEST\_SYMBOL**

This example shows a nested structure and also shows how varying datatypes can be used.

**See also:**

```
DECLARE SYMBOL ENDSTRUCTURE
```

## 16.38 SUBROUTINE Command

The SUBROUTINE command defines the beginning of a subroutine in a command procedure. The SUBROUTINE command must be the first executable statement in the subroutine.

**Format:**

**SUBROUTINE**

**Example:**

```
$ CALL PROCESS_FILE "C:\XLNT\*.*)"
.
.
.
$PROCESS_FILE: SUBROUTINE
$ DIR 'P1
$ ENDSUBROUTINE
```

This example shows how a command procedure can be modularized to have internal functions. The subroutine PROCESS\_FILE is invoked using the CALL command. A single parameter (the quoted string) is passed to the subroutine using the built-in symbol P1. The PROCESS\_FILE subroutine is denoted through the use of the SUBROUTINE and ENDSUBROUTINE statements.

**See also:**

[CALL](#)

## 16.39 UNTIL Command

The UNTIL command evaluates a specified expression and if false executes the body of the loop. If the expression evaluates to a true condition, control is transferred to the first command after the ENDUNTIL statement.

**Format:**

**UNTIL expression**

**Parameters:**

**expression**

Any valid XLNT expression. The expression will be evaluated at the start of each iteration. If the expression is false, the loop block will be entered. If the expression is true control is passed to the statement after the related ENDUNTIL.

**Example:**

```
$ DONE = 1
$ UNTIL DONE .GT. 5
$   DONE = DONE + 1
$ ENDUNTIL
```

This example shows a simple UNTIL loop. The symbol DONE is initialized to 1 and the symbol is incremented in the loop until the expression becomes true.

**See also:**

[ENDUNTIL FOR WHILE](#)

## 16.40 WAIT Command

The WAIT command causes the issuing process to wait for a specified amount of time.

**Format:**

**WAIT delta-time**

**Parameter:**

**delta-time**

Specifies a delta time (i.e., an offset from the current time to a time in the future) in the following format:

*hour:minute:second*

where:

*hour* is an integer in the range 0 to 23;

*minute* is an integer in the range 0 to 59;

*second* is an integer in the range 0 to 59.

**Description:**

WAIT is used to place a command procedure or interactive session into a temporary wait state until the waiting time specified is over. WAIT is typically used with a command procedure to wait for a specified time rather than loop continuously. Interactively you can terminate a WAIT command by entering Control/C.

**Example:**

**\$ WAIT 00:00:05**

This statement causes XLNT to wait five (5) seconds before continuing.

## 16.41 WHILE Command

The WHILE command evaluates a specified expression and if true executes the body of the loop. If the expression evaluates to a false condition, control is transferred to the first command after the ENDWHILE statement.

**Format:**

**WHILE expression**

**Parameters:**

**expression**

Any valid XLNT expression. The expression will be evaluated at the start of each iteration. If the expression is true, the loop block will be entered. If the expression is false, control is passed to the statement after the related ENDWHILE.

**Example:**

```
$ DONE = 1
$ WHILE DONE .LT. 5
$   DONE = DONE + 1
$ ENDWHILE
```

This example shows a simple WHILE loop. This example is actually the opposite the UNTIL loop's example. The test for the WHILE is for DONE to be less than 5. For the UNTIL the test was greater than 5.

**See also:**

[ENDWHILE FOR UNTIL](#)

## 16.42 XC Command

(Professional Edition Only)

The XC command is used to compile XLNT command procedure scripts into an executable image. The XC command is only available as part of the XLNT Professional Edition. Note: You must have an XLNT product edition installed to run the executable.

### Format:

**XC input-procedure-file output-executable-spec**

### Parameters:

#### **input-procedure-file**

A valid Windows file specification indicating the location of the XLNT command procedure.

#### **output-executable-spec**

Any valid Windows file specification indicating the location and/or complete name of the output file image. By default, XC will create a file using the filename portion of the input script file specification and use a type of "EXE" for the output-executable-spec.

### Qualifier:

#### **/LOG**

#### **/NOLOG (default)**

This qualifier causes a message to be displayed indicating successful operation. By default, messages are displayed only when an error occurs.

### Example:

```
$ XC TEST.XCP
```

This command encodes the procedure TEST.XCP into TEST.EXE. The encoded file can be used with any XLNT Product Edition, however, it must be used with the XLNT Run-Time License Edition.

# 17 Set / Show Commands

This section summarizes the Set and Show commands that are provided by XLNT.

These commands allow you to set and display various items of data.

## 17.1 SET DEFAULT

The SET DEFAULT command establishes the current device and/or directory. The new default is applied to all subsequent file specifications that do not explicitly include a device or directory name. SET DEFAULT accepts any valid directory specification as its parameter.

**Format:**

**SET DEFAULT [UNC] or [device-name:][\][directory][\][...]**

**Parameters:**

**UNC** (Universal Naming Convention)

\\machine\share\directory\sub-directory\

**-or-**

**device-name:**

Specifies the name of the device you want to establish as the default.

**directory**

Specifies the name of the directory you want to establish as the default. Specifying two periods (..) is shorthand for *up one directory level*.

**Qualifier:**

**/[NO]LOG**

Displays the directory name after it is set. The default is NOLOG.

**Examples:**

**\$ SET DEFAULT D:\ASCI**

Sets the default directory to ASCI on device D:

**\$ SET DEFAULT ..**

Set the default directory to the directory path above this current directory.

**\$ SET DEFAULT TEST**

Sets the default directory to a sub-directory of this current directory called TEST.

**\$ SET DEFAULT \\TEST\C\$\**

Sets the default directory, using UNC specification, to node TEST and the shared C\$ drive on that machine.

## 17.2 SET DOSERROR Command

The SET DOSERROR command allows XLNT to determine whether a DOS program's exit status is not successful for the purposes of ON statement handling.

**Format:**

**SET [NO]DOSERROR [integer]**

**Parameter:**

**integer**

This optional parameter is compared to a DOS program's exit status value. If the exit status is greater than or equal to this integer parameter, then the program is considered to have failed for the purposes of ON error handling. If this parameter is omitted, an integer value of one (1) is used. Most DOS program's indicate success by exiting with a zero and a positive non-zero value indicates failure.

**Examples:**

**\$ SET NODOSERROR**

This command disables ON error handling for DOS programs only. This is the default action when a command procedure is invoked.

**\$ SET DOSERROR 4**

This command enables ON error handling for DOS programs. If a DOS program exits with a status of 4 or greater, the program is considered to have failed.

## 17.3 SET FILE Command

The SET FILE command sets or clears various file attributes.

### Format:

**SET FILE/qualifier(s) file-name[,...]**

### Parameters:

**file-name**

Specifies one or more files whose attributes are to be modified.

### Qualifiers:

**/ACCESS**

Selects the date the file was last accessed for use with the /SINCE and /BEFORE qualifiers.

**/[NO]ARCHIVE**

Set or clear the file "Archive" attribute.

**/BEFORE[=time]**

Selects only those files dated prior to the specified time. You can specify an absolute time or a combination time as per the [Date and Time Format](#) rules. TODAY is the default.

**/[NO]COMPRESS**

Set or clear the file "Compress" attribute.

**/[NO]CONFIRM**

Controls whether a request is issued before each SET FILE operation to confirm that the operation should be performed on that directory. Valid responses are YES or NO. The default response is NO.

**/CREATE**

Selects the file creation date for use with the /SINCE and /BEFORE qualifiers.

**/DIRECTORY**

Modifies directories that match the filename pattern. The default is not to change directory attributes.

**/EXCLUDE=(filename[,...])**

Exclude the specified files from the DIRECTORY command.

**/[NO]HIDDEN**

Set or clear the file "Hidden" attribute.

**/[NO]LOG**

Controls whether the SET FILE command writes out successful operation messages.

**/[NO]READ**

Set or clear the file "Read Only" attribute.

**/SINCE[=time]**

Selects only those files dated after the specified time. You can specify an absolute time or a combination time as per the [Date and Time Format](#) rules. TODAY is the default.

**/[NO]SYSTEM**

Set or clear the file "System file" attribute.

### **/WRITE (default)**

Selects the date the file was last written to for use with the /SINCE and /BEFORE qualifiers.

#### **Example:**

```
$ set file/hidden/log c:\asci\Xlnt\Xlrt\*.*
```

```
XLNT-I-FILEATTRMOD, Attributes modified for C:\ASCII\XLNT\XLRT\DelsL1.isu  
XLNT-I-FILEATTRMOD, Attributes modified for C:\ASCII\XLNT\XLRT\XlntEDII.dll  
XLNT-I-FILEATTRMOD, Attributes modified for C:\ASCII\XLNT\XLRT\DelsL2.isu  
XLNT-I-FILEATTRMOD, Attributes modified for C:\ASCII\XLNT\XLRT\README.TXT  
XLNT-I-FILEATTRMOD, Attributes modified for C:\ASCII\XLNT\XLRT\XLRT.HLP  
XLNT-I-FILEATTRMOD, Attributes modified for C:\ASCII\XLNT\XLRT\XLRT.exe
```

This command sets the HIDDEN attribute for the selected files.

## 17.4 SET HOST Command

The SET HOST command allows XLNT users the ability to interactively logon to a remote computer system and execute commands.

### Format:

**SET HOST hostname /qualifiers**

### Parameter:

**hostname**

This parameter represents the computer you want to communicate with. The computer's name is dependent on the protocol you have chosen. For the *pipes* protocol, the computer's machine name would be used. For *TCP/IP* protocol, the computer's node name or IP address would be used. Additionally an optional colon may be specified to separate the nodename/IP address from a non-standard port number.

### Qualifier:

**/CLI=value**

By default, the destination shell you communicate with is XLNT. Acceptable values are XLNT and DOS (or the WinNT CMD).

**/LOG=[filename]**

This qualifier can provide a copy of the SET HOST session. By default, no copy is provided. If a filename is not specified, then a file named SETHOST.LOG in the current directory will be created.

**/PROTOCOL=value**

By default, TCPIP is used as the protocol for SET HOST communications. Acceptable values are PIPES and TCPIP.

### Description:

When SET HOST is issued, a *Username* prompt will appear on the screen. You may optionally enter a domain-name specification in UNC type format as well as the username for the remote system.

For example:

Username: *\\trust-domain\guest*

That should use the domain named *trust-domain* for authentication of the username *guest*. Specifying just the username alone, such as:

Username: *guest*

causes the default domain (if any) to perform authentication.

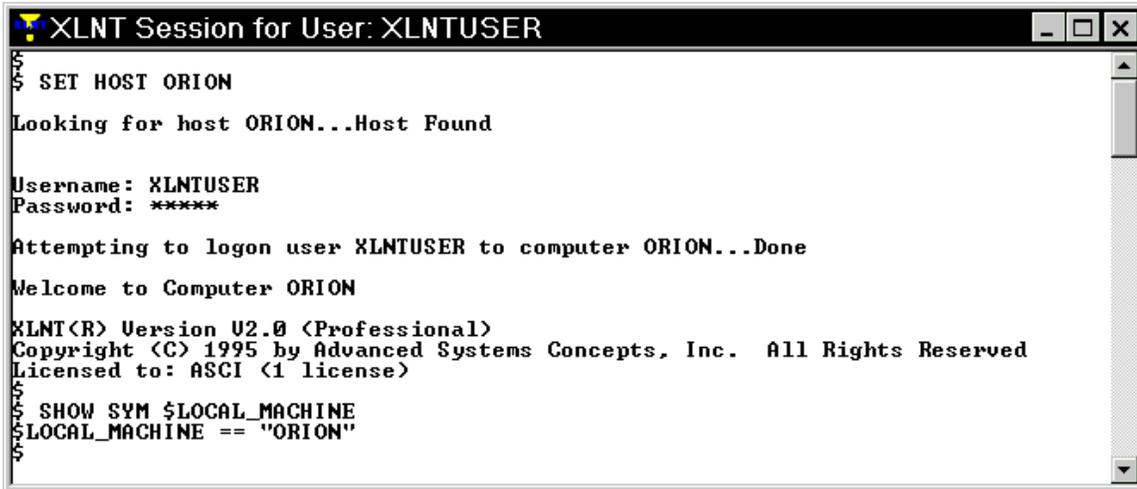
### Example:

**\$ SET HOST SYSTEM**

This command starts a TCP/IP connection to the remote host SYSTEM.

**\$ SET HOST "SYSTEM:1220"**

This command starts a TCP/IP connection to the remote host SYSTEM using the port number 1220 rather than the default port number. Note that the hostname and port specification are enclosed in quotation marks.



```
XLNT Session for User: XLNTUSER
$ SET HOST ORION
Looking for host ORION...Host Found

Username: XLNTUSER
Password: *****

Attempting to logon user XLNTUSER to computer ORION...Done

Welcome to Computer ORION

XLNT(R) Version U2.0 (Professional)
Copyright (C) 1995 by Advanced Systems Concepts, Inc. All Rights Reserved
Licensed to: ASCI (1 license)
$ SHOW SYM $LOCAL_MACHINE
$ LOCAL_MACHINE == "ORION"
$
```

**Note:** Only command-line programs can be executed using SET HOST. If you run a windows program, the session may hang until you manually terminate the windows program.

## 17.5 SET MESSAGE Command

The SET MESSAGE command provides the user with the ability to enable or disable the display of messages associated with XLNT itself. This command does not affect any external programs run with XLNT.

### Format:

**SET MESSAGE /qualifier**

### Qualifier:

**/[NO]MESSAGE**

This qualifier enables or disables the display of XLNT messages. /MESSAGE enables the display of messages (default). /NOMESSAGE disables the display of XLNT messages.

### Description:

The SET MESSAGE command allows the issuer to enable or disable the XLNT message mechanism. This is typically done to avoid additional possibly confusing messages from appearing on the console as part of the execution of a command procedure directed at an end-user. A command procedure can determine whether messages are enabled or disabled through the F\$ENVIRONMENT(MESSAGE) lexical.

### Example:

```
$ a=xx
%XLNT-E-UNKSYM, unknown symbol
$ set message/nomessage
$ a=xx
$ set message/message
$ a=xx
%XLNT-E-UNKSYM, unknown symbol
```

This example begins by demonstrating an illegal assignment statement. Symbol xx is not defined. The XLNT-E-UNKSYM is an error message indicating that the symbol is unknown. Note that once messages are turned off the same operation does not produce a displayed error (please note that the error still occurred and is trap-able and detectable by the procedure). Once messages are enabled, the error is redisplayed. See the [F\\$ENVIRONMENT lexical](#) for more information.

## 17.6 SET ON Command

The SET ON command enables error checking by the command interpreter after execution of each command in the command procedure. Specify SET NOON to disable error checking.

**Format:**

**SET [NO]ON**

**Description:**

Please note that if a SET ON command is issued without a corresponding ON statement, any error will terminate the command procedure.

**Example:**

```
$ SET NOON
$ DELETE *.SAV
$ SET ON
$ COPY *.TXT C:\TEMP\.*
```

The initial SET NOON provides this command procedure extract with a safe way of deleting all .SAV files whether they exist or not. If the files didn't exist a warning condition would be raised. So SET NOON disables error checking. The SET ON restores error checking since a failure of the COPY to perform would be considered serious.

## 17.7 SET PERMISSIONS Command

The SET PERMISSIONS command allows the user to control security access for most objects within Windows NT. This command is available only for Windows NT.

### Format:

**SET PERMISSIONS** object-name[,...]

### Parameters:

#### object-name

#### If the object-name is a File and/or Directory:

Specifies the name of the object on which the permissions will be set, e.g., a file name. If more than one object is being specified, separate the object names with commas (.). Please note that when more than one object is specified the objects must be of the same type. For files and directories, the object name can be specified as:

- a relative path, such as "abc.dat" or "..\abc.dat"
- an absolute path, such as "abc.dat", "c:\dir1\abc.dat", or "g:\remotedir\abc.dat".
- a UNC name, such as "\\machinename\sharename\abc.dat"

Moreover for files and directories, any valid XLNT wildcard specification can be specified.

#### If the object-name is a Share:

Specifies the name of the Share using either the Share name itself (if local) or the remote form \\machine-name\sharename. Note that Administrative shares (i.e. C\$) cannot have permissions set.

#### If the object-name is a Regkey:

Specifies the name of the Registry Key in the form of hive\key1\key2\... (if local) or \\machine-name\hive\key1\key2\... . The hive can be any of the supported hives (and their abbreviations) as listed in the F\$ADDREGISTRY lexical documentation.

#### If the object-name is a Printer:

Specifies the name of the Printer in the form of either printername or \\machine-name\printername.

#### If the object-name is a Service:

Specifies the name of the Service in the form of either Servicename or \\machine-name\servicename.

### Description:

SET PERMISSIONS is a powerful administrative command which allows object ownership and permissions to be granted (allow or deny access) or revoked for any of the five (5) objects listed above. SET PERMISSIONS provides a consistent interface for specifying permissions. The account name (which may optionally include a machine or domain) is listed along with the access requested. Rather than a single command for each permission, SET PERMISSION allows multiple permissions as well as multiple objects (of the same type) in a single command invocation. Moreover, SET PERMISSIONS through its extension of UNC syntax allows remote objects to be manipulated as well. Please note that some of the accounts listed may be internal Windows NT pseudo accounts. Their access should not be changed.

### Qualifiers:

**/ACCOUNT=(name[:pri-perm[:prop-perm]] [,...])**

**Common:** The *name* portion represents the name of an account or group that is being granted or denied access to the object. The name portion may be specified as *machine\name* or *domain\name* or just simply *name*. From a

security point-of-view the *name* represents a trustee. You may also specify a SID using textual SID notation (S-n-n-n-n-n...). *Prop-Perm* only applies to file directory objects and is not applicable to any other objects supported by SET PERMISSIONS.

**Files and Directories:** *Pri-Perm* represents the permissions for the object itself. Valid permissions for files are: READ, CHANGE, ALL, FULL, LIST, ADD, ADDREAD, NOACCESS or NONE. Special access letters RXWDPO may also be specified, however, you cannot specify both special access letters and keywords together for the same trustee. The keywords LIST, ADD, ADDREAD are specific to file directories. *Prop-Perm* represents the default permissions for a file directory object or the permissions that are propagated to a file within the directory. *Prop-Perm* is ignored if the object is not a file directory or if the object is a share. Please note that many of the permissions set in *Pri-Perm* also affect *Prop-Perm*. Valid *Prop-Perm* permissions include the special access letters as well as ALL, FULL and ANS (Access Not Specified). *Pri-Perm* must be specified unless the REVOKE qualifier is also specified or an error will be produced. Multiple trustee/permissions may be specified as a list under the ACCOUNT qualifier. The special access letters represent the following:

- R** - Read Access
- W** - Write Access
- X** - Execute Access
- D** - Delete Access
- P** - Ability to set permissions on object
- O** - Ability to take ownership of object

**Shares:** *Pri-Perm* may be one of the following: FULL (or ALL), READ, CHANGE, NONE (or NOACCESS). Special access letters are not supported for Shares.

**Registry Keys:** *Pri-Perm* may be one of the following: FULL (or ALL), READ, NONE (or NOACCESS). Single letter special access permissions may also be specified as noted:

- Q** - Query Value
- E** - Enumerate Sub Keys
- N** - Change Notification
- S** - Set Value
- C** - Create Sub Keys
- L** - Create Link
- P** - Ability to set permissions (Write DAC)
- O** - Ability to Take Ownership
- D** - Delete
- R** - Read Control

**Services:** *Pri-Perm* may be one of the following: FULL (or ALL), READ, CHANGE, NONE (or NOACCESS). READ is a keyword that contains FQE access and CHANGE is a keyword that represents CGSZIU access. Single letter special access permissions may also be specified as noted:

- F** - Query Configuration
- C** - Change Configuration
- Q** - Query Status
- E** - Enumerate Dependents
- G** - Start Service
- S** - Stop Service
- Z** - Pause/Continue Service
- I** - Interrogate
- U** - User-defined Control

- R** - Read Control
- D** - Delete
- P** - Ability to set permissions
- O** - Ability to Take Ownership

**Printers:** *Pri-Perm* may be one of the following: FULL (or ALL), MANAGE, PRINT, NONE (or NOACCESS). Special access permissions are not allowed.

### **/[NO]LOG**

Specifies whether the names of each object is displayed after its permissions have been set.

### **/OBJECT=(FILE | SHARE | REGKEY | PRINTER | SERVICE)**

Specifies the type of object. Object keywords are FILE, SHARE, REGKEY, PRINTER and SERVICE. If this qualifier is omitted the object is assumed to be a FILE.

### **/REPLACE**

This qualifier indicates that the trustees specified should completely replace all access level permissions. By default, trustees specified with the ACCOUNT qualifier are merged into an existing ACL stream.

### **/REVOKE**

This qualifier removes all access permissions for the account(s), specified by /ACCOUNT, associated with the specified object.

### **/TAKE\_OWNERSHIP**

This qualifier causes the ownership of the specified object(s) to be acquired by the submitter of this command. The user must have the appropriate rights and/or permissions to take ownership of an object. Please note that if you attempt to take ownership of an object that you do not have any access permissions to (in other words, you either have only "Take Ownership" permission or you are exercising your rights as an Administrator), you must specify the REPLACE qualifier to complete the ownership operation. All existing permissions will be replaced with an ACE indicating that you now have full control. If the REPLACE qualifier is not specified, then the take ownership operation will not be performed if you also lack other access rights to the object.

*The following set of qualifiers pertain to FILE objects only.*

### **/ACCESS**

Modifies the time value specified with the /BEFORE or /SINCE qualifier. The /ACCESS qualifier selects files based on their last accessed dates. This qualifier is incompatible with the /CREATED and /WRITE qualifiers.

### **/BEFORE[=time]**

Selects only those files dated prior to the specified time. Time can be specified as absolute time, as a combination of absolute and delta times, or as one of the following keywords: TODAY (default) or YESTERDAY.

### **/[NO]CONFIRM**

Controls whether a request is issued before each SET PERMISSIONS operation to confirm that the operation should be performed on that file. The following responses are valid: YES NO ALL QUIT . A reply of YES allows the operation to continue. A reply of NO will bypass the current file. A reply of ALL will allow the command to continue, but no further prompts will be issued. A reply of QUIT will stop the command .

### **/CREATED**

Modifies the time value specified with the /BEFORE or /SINCE qualifier. The /CREATED qualifier selects files based on their dates of creation. This qualifier is incompatible with the /ACCESS and /WRITE qualifiers.

### **/DIRECTORY**

Specifies that only subdirectories are to have their permissions changed. Specifying DIRECTORY implies SUBDIRECTORY as well. By default, files' permissions are changed.

**/EXCLUDE=(filename[,...])**

Excludes the specified file(s) from the SET PERMISSIONS operation.

**/[NO]EXISTING\_FILES**

Specifies whether permissions being applied to a directory should be applied to the files that currently exist within the directory. The default is to not apply the permissions to the existing files. Note: This qualifier is not intended as a substitute for specifying valid XLNT file specification wildcards.

**/SINCE[=time]**

Selects only those files dated after the specified time. Time can be specified as absolute time, as a combination of absolute and delta times, or as one of the following keywords: TODAY (default) or YESTERDAY.

**/SUBDIRECTORIES**

Specifies whether permissions being set or revoked are to apply to subdirectories. By default, subdirectories are not included. Please read the DIRECTORY qualifier for more information as to how to specifically include or exclude directories.

**/WRITE**

Modifies the time value specified with the /BEFORE or /SINCE qualifier. The /WRITE qualifier selects files according to the dates on which they were last written. This qualifier is incompatible with the /ACCESS and /CREATED qualifiers.

**Examples:****\$ SET PERMISSIONS C:\TEST -  
/ACCOUNT=GUEST:READ /EXIST**

This command changes the permission of the directory C:\TEST for account GUEST to READ access, default file permission is also set to READ. The EXISTING qualifier indicates that all files within the TEST directory are to be changed to reflect the new file permission.

**\$ SET PERMISSIONS C:\TEST /ACCOUNT=GUEST:LIST**

This command adds the LIST permission for account GUEST to the TEST directory. Note that even if /SUBDIR had been specified the LIST permission also denotes that no file permissions are specified.

**\$ SET PERMISSIONS C:\\*.\* -  
/ACCOUNT=(GUEST:LIST,NEWHIRE:FULL, -  
SPECIAL:RW:RWX) /SUBDIR**

This command sets permissions for all files and subdirectories (since /SUBDIR is present) at the root directory level of C:\. Directories will have account GUEST directory permission set to LIST, NEWHIRE will have FULL or ALL access for both directory and file permissions, account SPECIAL will have directory access set to RW and default file permissions set to RWX. Please note that directory oriented permissions are ignored unless a directory or subdirectory is encountered as part of the file searching process.

## 17.8 SET PREFERENCE Command

The SET PREFERENCE command allows the user to control various aspects of the console terminal environment as well as other XLNT Registry items that may be of some use.

### Format:

**SET PREFERENCE /qualifiers**

### Description:

This command is used to set various user and system XLNT preference items. Some options are dynamic and some require the user to leave XLNT and re-enter. To save preferences across XLNT sessions use the /SAVE qualifier.

### Qualifiers:

#### **/[NO]AUTOFOREIGN**

Indicates whether XLNT should process potentially invalid XLNT commands as transparent automatic foreign (CMD) commands. By default, this facility is enabled on a per-user basis.

#### **/COLUMNS=numeric-value**

Specifies the console screen's width. The default value is 80. Minimum is 1, maximum is 132.

#### **/CURRENT**

Indicates that XLNT should examine the current console properties. This qualifier is usually used in conjunction with the /SAVE qualifier.

#### **/EDITOR=file-spec**

Indicates the program to execute when the XLNT EDIT command is entered. By default, the CMD EDIT program is invoked.

#### **/[NO]INSERT**

Indicates whether Overstrike (noinsert) or Insert mode is asserted for the XLNT console. By default, Overstrike (noinsert) is assumed.

#### **/LOGIN\_DIRECTORY="directory-name"**

When specified allows the user to change the initial directory that XLNT sets on initial invocation on a user basis.

#### **/MAXLEVEL=numeric-value**

Indicates the maximum number of nested command procedures that can be executed. The default is 32. This qualifier applies to the system. (For Advanced Users Only).

#### **/MAXRECALL=numeric-value**

Indicates the maximum number of command recall buffers to be allocated. The default is 20. This qualifier applies to the system.

#### **/PROMPT="string"**

When specified, allows the user to change the initial XLNT prompt string (normally the dollar sign (\$)). The qualifier applies to the system.

#### **/SAVE**

Used to save all preferences to the XLNT Registry area.

#### **/[NO]SPLASH\_PAGE**

Controls whether the initial XLNT animation screen is presented.

**/SYLOGIN="file-name"**

When specified indicates the file specification of an XLNT command procedure to be invoked when a user starts XLNT. This qualifier is used to invoke an XLNT procedure for all XLNT users. USERLOGIN, if specified, is executed after SYLOGIN, on a user by user basis.

**Note: You must be a member of the "Administrators" group to change this item.**

**/USERLOGIN="file-name"**

When specified indicates the file specification of an XLNT command procedure to be invoked when a user starts XLNT. This qualifier is used to invoke an XLNT procedure on a user basis. SYLOGIN, if specified, is executed first and then USERLOGIN is executed.

**Example:****\$ SET PUF/INSERT/SAVE**

This command indicates that Insert mode rather than Overstrike should be enabled. The SAVE qualifier indicates that this preference should be saved in the current user's XLNT registry.

## 17.9 SET PROCESS/PRIORITY

The SET PROCESS/PRIORITY command changes the run-time priority of the XLNT shell process or any selected process. You must have the appropriate administrative rights to change a process' priority.

**Format:**

**SET PROCESS/PRIORITY/qualifiers**

**Qualifiers:**

**/IDENTIFICATION=pid**

This optional qualifier indicates the specific process whose priority should be changed. If omitted, the current XLNT shell process is changed.

**/PRIORITY=keyword**

This optional qualifier indicates the new priority class that should be used and associated with the selected process. Valid keywords are: HIGH, IDLE, NORMAL or REALTIME. No default exists for this qualifier.

**Example:**

**\$ SET PROCESS/PRIORITY=HIGH/ID=24**

This command changes the Windows NT priority for the process whose PID is 24(hex) to HIGH.

## 17.10 SET PROMPT

The SET PROMPT command replaces the default XLNT prompt (\$) with the specified string.

### Format:

**SET PROMPT/qualifier ["string"]**

### Parameter:

**string**

Specifies the new prompting string. If this parameter is omitted, XLNT will revert to its default string (\$). If DIRECTORY is specified, this parameter is ignored.

### Qualifier:

**/DIRECTORY**

This qualifier causes the current directory specification to be automatically appended to the current prompt string at the beginning. As the current directory is changed, the prompt will also automatically change. To disable this feature, issue a new SET PROMPT.

**/PREFIX=string**

This qualifier, which is meant to be used with the /DIRECTORY qualifier, allows you to place a string in front of the current directory. If /DIRECTORY is not specified, this qualifier is ignored.

### Examples:

```
$ SET PROMPT " '$LOCAL_MACHINE'> "
```

This useful command lets you set your prompt to the system you are executing XLNT on. Please note that two *single* quotes precede the symbol to indicate explicit symbol substitution.

```
$ SET PROMPT/DIRECTORY/PREFIX=" '$LOCAL_MACHINE'::"
```

```
TEST: : C: \ASCII >
```

This example takes the prompt specification one step further by also allowing an automatic update of the prompt as you move from directory to directory.

## 17.11 SET SYMBOL/SCOPE

The SET SYMBOL command is used to change the lookup of symbols within a command procedure.

### Format:

**SET SYMBOL/SCOPE=keyword(s)**

### Qualifier:

**/SCOPE=([NO]LOCAL,[NO]GLOBAL]**

**NOLOCAL** causes all local symbols defined in outer mode procedure levels to be treated as being undefined by the current procedure level.

**LOCAL** removes any previous symbol translation limit (such as that caused by NOLOCAL).

**NOGLOBAL** causes all global symbols to be inaccessible to the current procedure level.

**GLOBAL** removes any previous symbol translation limit (such as that caused by NOGLOBAL).

### Example:

```
[Procedure Main]
```

```
$ a=1
```

```
$ @b
```

```
$ exit
```

```
[Procedure B]
```

```
$ set symbol/scope=nolocal
```

```
$ show symbol a
```

```
$ exit
```

This example shows two command procedures, MAIN and B. MAIN creates a local symbol named A (with a value of 1), and then invokes procedure B. Proc B issues a symbol scope restriction for local symbols. This now causes the local symbol A (created in the outer procedure MAIN) to become inaccessible or otherwise undefined to procedure B. The SHOW SYM A will therefore fail with an “undefined symbol” error. Please note that SET SYMBOL/SCOPE is only applicable to the current procedure level. Restrictions on symbol scope apply only to the procedure issuing the command.

## 17.12 SET TIME Command

The SET TIME command is used to change the system time for the current machine.

### Format:

**SET TIME absolute-time**

### Parameter:

**absolute-time**

This required parameter is an XLNT absolute time format string that will be used to set the current machine's clock. Absolute time uses the format “dd-mmm-yyyy:hh:mm:ss.mms” (or any portion) or the strings TODAY, YESTERDAY or TOMORROW.

### Description:

This command is used to change the system time for the current machine.

### Example:

```
$ SET TIME 9:35
```

This command sets the current machine's time to 9:35am.

## 17.13 SET VARIABLE Command

The SET VARIABLE command is used to define environment variables.

### Format:

**SET VARIABLE environment-variable text**

### Parameters:

**environment-variable**

The desired environment variable to define.

**text**

The text to be set for the environment variable

### Qualifier:

**/SYSTEM**

This qualifier allows access to the system environment variables that are the basis (or starting point) for each process copy of the environment variables. /SYSTEM may require administrative rights. If omitted, only the process copy is returned. This is the default.

### Description:

The SET VARIABLE command provides a method of setting a CMD environmental variable for Windows. Environmental variables are changed on a process basis only unless the /SYSTEM qualifier is used. Use SHOW VARIABLE to interactively inspect the contents of an environmental variable or F\$GETVARIABLE to programmatically retrieve the contents of an environmental variable.

### Example:

```
$ path=f$getvar("path")
$ show symbol path
PATH = "C:\WINNT35\SYSTEM32; C:\WINNT35; C:\WINDOWS; C:\WINDOWS\SYSTEM;
C:\MSOFFICE; C:\MSOFFICE\WORD; C:\MSDEV\BIN; "
$ set variable path " 'path'; c:\asci\xlnt"
$ show variable path
PATH=C:\WINNT35\SYSTEM32; C:\WINNT35; C:\WINDOWS; C:\WINDOWS\SYSTEM;
C:\MSOFFICE; C:\MSOFFICE\WORD; C:\MSDEV\BIN; C:\ASCII\XLNT
```

This example gets the value of the environment variable PATH and then adds on the additional directory specification.

## 17.14 SET VERIFY

The SET VERIFY command controls whether lines in command procedures are displayed on the console window or are printed in a batch log job.

**Format:**

**SET [NO]VERIFY**

**Description:**

This command is used to set verification (or display) of XLNT commands and statements as the procedure executes. SET VERIFY is typically used in debug mode to help determine scripting errors. Programmatically the F\$VERIFY lexical can be used to read and set command procedure verification as well. SET NOVERIFY disables command verification. By default, interactive console procedures begin with SET NOVERIFY. Batch job procedures assume SET VERIFY for batch job script audit. You can customize these settings by editing the SYLOGIN and/or USRLOGIN.XCP procedures.

**Example:**

**[Contents of file TEST.XCP]**

```
$ SET VERIFY  
$ A=1  
$ B=2  
$ WRITE $STDOUT "TEST"  
$ SET NOVERIFY  
$ B =3
```

```
$ @TEST
```

This procedure sets verification on entry to the procedure. As a result, the procedure's statements and commands are displayed as they are executed. When the SET NOVERIFY is executed, the statements following the command are not echoed or displayed.

## 17.15 SHOW DEFAULT Command

The SHOW DEFAULT command displays the current default device and directory string.

**Format:**

**SHOW DEFAULT**

## 17.16 SHOW DEVICE

The SHOW DEVICE command displays information about all or specific disk devices that are configured into the system.

### Format:

**SHOW DEVICE [device\_name[:]]**

### Parameters:

**device\_name[:]**

Specifies the letter or name of the device to be displayed. If the parameter is omitted, information on all the disk devices in the system is displayed.

### Qualifiers:

**/FULL**

Specifies that a complete list of device information is to be displayed.

**/OUTPUT=filename**

Specifies that the output of the SHOW DEVICE command is to be written to the specified file. If this qualifier is omitted, output is displayed on the console window.

**/PAGE**

Controls whether output from the command is displayed one screen at a time.

### Restriction:

A restriction within WinNT prevents remotely connected devices from being fully displayed (as to the server and share name) when this command is issued from a SET HOST session.

### Example:

#### **\$ SHOW DEVICE**

```
Drive A:, Type is Removable
Drive C:, Type is Fixed
Drive D:, Type is Fixed
Drive E:, Type is CD-ROM
Drive F:, Type is Remote connected to \\TEST\c$
```

This command displays a brief listing of all devices.

#### **\$ SHOW DEVICE/FULL**

```
Drive A:, Type is Removable
```

```
Drive C:, Type is Fixed
Sectors/Cluster :      32      VolumeName      : N/A
Bytes/Sectors   :      512     FileSystemType  : FAT
Free Clusters   :      451     Total Clusters : 44527
Free Disk Space : 7389184     MaxFileLength  : 255
Total Disk Space: 729530368   Serial Number   : 354688770
```

```
File System Characteristics: Case is preserved, Supports UNI CODE
```

```
Drive D:, Type is Fixed
Sectors/Cluster :      64      VolumeName      : N/A
Bytes/Sectors   :      512     FileSystemType  : FAT
Free Clusters   :    18538     Total Clusters : 33006
Free Disk Space : 607453184     MaxFileLength  : 255
Total Disk Space: 1081540608   Serial Number   : 526849531
```

```
File System Characteristics: Case is preserved, Supports UNI CODE
```

```
Drive E:, Type is CD-ROM
Sectors/Cluster :      1      VolumeName      : NTWKSCHK351
Bytes/Sectors   :     2048    FileSystemType  : CDFS
Free Clusters   :          0    Total Clusters  : 299326
Free Disk Space :          0    MaxFileLength   : 37
Total Disk Space: 613019648    Serial Number   : 908330670
```

```
Drive F:, Type is Remote connected to \\TEST\c$
Sectors/Cluster :      1      VolumeName      : N/A
Bytes/Sectors   :     512    FileSystemType  : NTFS
Free Clusters   :     93340    Total Clusters  : 1427265
Free Disk Space : 47790080    MaxFileLength   : 255
Total Disk Space: 730759680    Serial Number   : 2296798860
```

File System Characteristics: Case is preserved, Supports Case-Sensitive filenames, Supports UNICODE, Preserves ACLs, Supports file-based Compression

This command displays very detailed information about each device found on the machine as well as remote devices.

## 17.17 SHOW DLLS

The SHOW DLLS command will display the dynamic link libraries that are currently loaded on the system by process. (Windows 2000/NT Only)

### Format:

**SHOW DLLS**

### Qualifiers:

**/IDENTIFIER=pid**

A number containing the id of the process to be displayed. It may be entered in decimal (default) or in hex by means of the hex radix operator (%x). If omitted, all processes are displayed.

**/FULL**

By default, only the names of each DLL are displayed. If the /FULL qualifier is specified, the DLL's base address, size in bytes, and entry point address are also displayed.

**/OUTPUT=filename**

Controls where the output of the command is sent. By default, the display is written to the console window.

### Example:

**\$ SHOW DLLS/IDENTIFIER=%X73**

Active DLL's on ZEUS 15-Apr-1998 16:22:30

Process Id: 73 (115) C:\MSOFFICE\WINWORD\WINWORD.EXE

C:\WINNT35\System32\ntdll.dll

C:\WINNT35\system32\KERNEL32.dll

C:\WINNT35\system32\GDI32.dll

C:\WINNT35\system32\RPCRT4.dll

C:\WINNT35\system32\COMCTL32.dll

C:\MSOFFICE\WINWORD\wwintl32.dll

C:\WINNT35\system32\USER32.dll

C:\WINNT35\system32\ADVAPI32.dll

C:\WINNT35\system32\SHELL32.dll

C:\WINNT35\System32\WINSPOOL.DRV

## 17.18 SHOW HOST

The SHOW HOST command displays information concerning those remote users who are currently logged into your computer system. This command is only valid for those systems which have the XLNTNET Service installed.

**Format:**

**SHOW HOST/qualifier**

**Qualifier:**

**/OUTPUT=filename**

Directs the output of the SHOW HOST command to the specified file. By default, the output will be displayed on the console window.

**Example:**

**\$ SHOW HOST**

Windows NT 4.0 on NODE 3/13/96 11:22:10 AM Uptime: 22 21:14:56.999

XLNT Local User	Remote User	Remote Machine	Connect Time
Guest	Guest	OTHER	3/13/96 11:22:00 AM
XIntNetS PID: B6			
XIntCli PID: 112			

This example displays all remote users who are accessing your system.

## 17.19 SHOW LOCALTIMEZONE

The SHOW LOCALTIMEZONE command will display information about the current timezone for the local computer.

**Format:**

**SHOW LOCALTIMEZONE**

**Qualifiers:**

**/FULL**

This optional qualifier will display complete information about the local timezone. By default, only the timezone name will be displayed.

**/OUTPUT=file-spec**

This optional qualifier directs the command output to the specified file.

## 17.20 SHOW MACHINE

The SHOW MACHINE command displays the current hardware architecture and platform.

**Format:**

**SHOW MACHINE**

**Example:**

**\$ SHOW MACHINE**

Windows NT 3.51 CPU information for TEST on 8/29/96 4:18:26 PM

Processor Architecture: Intel, 486

Page Size: 4096 No Of Processors: 1

Allocation Granularity: 65536

## 17.21 SHOW MEMORY

The SHOW MEMORY command displays the machine's memory and cache information and statistics.

### Format:

**SHOW MEMORY**

### Qualifiers:

**/CACHE**

Displays detailed cache statistics.

**/FULL**

Displays detailed memory statistics (including CACHE).

**/ON=machine**

Allow the command to be directed at a valid machine within the domain.

### Example:

#### \$ SHOW MEMORY/FULL

Windows NT 4.0 System Memory Resources for T2 on 1/29/98 9:34:55 AM

#### Physical Memory Usage (Bytes)

Available Bytes : 7049216  
Free System Pages : 7536

#### Dynamic Memory Usage (Bytes)

Non-Paged Pool : 1503232  
Paged Pool : 10448896  
Paged Pool Resident : 7409664

#### System Memory Usage (Bytes)

Code Resident : 1822720  
Driver Total : 4415488  
Driver Resident : 876544  
Cache Resident : 1605632

#### Page File Usage (Bytes)

Committed Bytes : 39321600  
Commit Limit : 86712320

#### Caching Memory Usage (Bytes)

Cache Bytes : 11714560  
Cache Bytes Peak : 14807040

Reads	:	549778	Data Flushes	:	105189
Copy	:	549778	Data Flush Pages	:	105189
Copy Reads	:	549778	Fast Read Not Possible	:	0
Asynch. Copy Reads:	:	0	Fast Read Misses	:	0
Asynch. Fast Reads:	:	0	Asynch. MDL Reads	:	0
Lazy Write Flushes:	:	45897	Lazy Write Pages	:	105006

## 17.22 SHOW PERMISSIONS

The SHOW PERMISSIONS command provides a single command with which security information can be obtained for FILES, SHARES, REGISTRY KEYS, SERVICES and PRINTERS. Object specific security information can also be obtained through commands that are available for each object. For example, DIRECTORY/SECURITY for FILE objects.

### Format:

**SHOW PERMISSIONS/qualifier object,...**

### Parameter:

#### **object**

This parameter or list of parameters is viewed as an object by this command of a type specified by the qualifier OBJECT (default is FILE). The syntax of each object is specific to the type of object specified, however, all objects under XLNT support UNC style syntax which means that a remote specification of \\machine\object may be specified.

### Description:

The SHOW PERMISSIONS command is used to display the owner and permissions of any of the following objects: FILE, REGKEY, SHARE, SERVICES and PRINTERS. This command can be specified for both local and remote objects (assuming the issuer has sufficient permissions or rights). Please read the SET PERMISSIONS command for more information on understanding the various permission's codes.

### Qualifiers:

**/OBJECT=(FILE | SHARE | REGKEY | SERVICE | PRINTER)**

This qualifier indicates the type of object specified as a parameter to the command. By default, if this qualifier is omitted the default object type is FILE.

**/OUTPUT=filename**

Directs the output of the SHOW PERMISSION command to the specified file. By default, the output will be displayed on the console window.

### Example:

**\$ SHOW PERM/OBJ=SERVICE ALERTER**

Permissions for Service Objects on ZEUS 15-Apr-1998 16:27:10

Object : ALERTER

Owner : NT AUTHORITY\SYSTEM

Everyone	Read (FQEIUR)
Power Users	Read (FQEGSZIUR)
Administrators	Full Control (FCQEGSZIURDPO)
NT AUTHORITY\SYSTEM	Read (FQEGSZIUR)

**\$ SHOW PERM/OBJ=REGKEY**

**\\HERA\HKLM\SOFTWARE**

Permissions for Registry Key Objects on ZEUS 15-Apr-1998 16:27:29

Object : \\HERA\HKLM\SOFTWARE

Owner : Administrators

Everyone	Read (QENRSCD)
Administrators	Full Control (QENRSCLPD)
NT AUTHORITY\SYSTEM	Full Control (QENRSCLPD)
CREATOR OWNER	Full Control (QENRSCLPD)

**\$ SHOW PERM/OBJ=PRINTER PRINTSERVER17**

Permissions for Printer Objects on ZEUS 15-Apr-1998 16:32:52

Object : PRINTSERVER17

Owner : Administrators

CREATOR OWNER	Manage Documents (Manage)
Everyone	Print (Print)
Administrators	Full Control (Full)
Power Users	Full Control (Full)

The examples above show the permissions of a service named ALERTER as well as remote registry key SOFTWARE on machine HERA in the HK\_LOCAL\_MACHINE hive. The permissions for printer PRINTSERVER17 are probably more intuitively understandable.

## 17.23 SHOW PREFERENCES

The SHOW PREFERENCES command displays information about the current interactive XLNT user in terms of XLNT's specific processing towards that user.

**Format:**

**SHOW PREFERENCES**

**Example:**

**\$ SHOW PREFERENCES**

No User Login Procedure defined

Default Login Directory: C:\ASCII\NEWXLNT

No default Editor defined

Splash Page Output is Disabled

Console Settings : Rows: 100, Columns: 80, Insert Mode, Enhanced Mode Active

SyLogin Procedure: C:\ASCII\XLNT\sylogin.xcp

Automatic Foreign Command Detection is Enabled

This example shows a user's current XLNT preferences.

## 17.24 SHOW PROCESS

The SHOW PROCESS command displays information about the current process.

**Format:**

**SHOW PROCESS [image-name]**

**Parameter:**

**image-name**

This optional parameter allows a specific process to be selected by image name. If image-name is omitted and the /IDENT qualifier is not specified, the current process is used.

**Qualifiers:**

**/FULL**

This qualifier causes all process and thread information to be displayed. By default, only process information is displayed.

**/IDENT=nnn**

This qualifier causes a specific process to be selected for display. The value specified with this qualifier is the process' identification.

**/MEMORY**

This qualifier displays a process' memory utilization.

**/ON=machinename**

This qualifier allows the command to be directed to and executed on another machine in the domain.

**/OUTPUT=filename**

Directs the output of the SHOW PROCESS command to the specified file. By default, the output will be displayed on the console window.

**Examples:**

**\$ SHOW PROCESS SYSTEM**

```
Windows NT 3.51 on TEST 8/29/96 4:13:27 PM Uptime: 8 05:33:01.999
Logged on User Guest is Interactive
```

```
Process Image Name : System          Process Id       : 00000002
Working Set Min    : 543520361       Working Set Max  : 539769419
Priority Class     : NORMAL          Priority Base    : 8
```

This command displays some general information about the process with the image name of "system".

**\$ SHOW PROCESS SYSTEM/FULL**

```
Windows NT 3.51 on TEST 8/29/96 4:13:20 PM Uptime: 8 05:32:53.999
Logged on User Guest is Interactive
```

```
Process Image Name : System          Process Id       : 00000002
Working Set Min    : 543520361       Working Set Max  : 539769419
Priority Class     : NORMAL          Priority Base    : 8
```

```
Handle Count      : 349              Thread Count     : 18
%CPU Time         : 0 00:03:52.023   Page Faults/Sec : 3218
Page File Usage   : 61440            Peak Page File Usage : 282624
Working Set       : 16384            Peak Working Set  : 475136
Virtual Address Space: 1523712      Peak Virtual Address Space: 1589248
```

```
Thread ID: 00000001          %CPU Time: 0 00:00:37.864
State: Ready                Pri ori ty [Base/Cur]:    0/0
Thread ID: 00000003          %CPU Time: 0 00:00:16.033
```

This command (whose output is abbreviated due to length) shows the more detailed aspects of this command.

## 17.25 SHOW SERVER/FILES

The SHOW SERVER/FILES command will display information about resources open on a server. A file can be opened one or more times by one or more applications. Each file opening is uniquely identified.

### Format:

**SHOW SERVER/FILES**

### Qualifiers:

**/ON=servername**

Specifies the name of the server computer on which the operation will execute. If omitted, the local computer will be used.

**/OUTPUT=filename**

Specifies where the output of the command is sent.

**/USER=username**

Specifies the name of the user whose open files are to be displayed. If omitted, files for all users will be displayed.

### Example:

#### **\$ SHOW SERVER/FILES**

Open Files on TEST 08-Jul-1997 09:29:24

ID	Path	User Name	Locks
20	C:\ASCII\XLNT\TMP.TXT	guest	0

## 17.26 SHOW SESSION

The SHOW SESSION command will display information about all current sessions. The information displayed will include the name of the computer that established the session; the name of the user that established the session; the number of files, devices, and pipes opened during the session; the time that the session has been active; the time that the session has been idle; the manner in which the session was established; and the type of client (i.e., operating system) that established the session.

### Format:

#### SHOW SESSION

### Qualifiers:

#### /CLIENT=clientname

Specifies the name of the client computer for which information is to be displayed. If omitted, information for all clients will be displayed.

#### /ON=servername

Specifies the name of the server computer on which the operation will execute. If omitted, the local computer will be used.

#### /OUTPUT=filename

Specifies where the output of the command will be sent.

#### /USER=username

Specifies the name of the user for which information is to be displayed. If omitted, information for all users will be displayed.

### Example:

#### \$ SHOW SESSION

Active Sessions on TEST 08-Jul-1997 09:31:07

Client	Type	User Name	Active	Idle
-----	----	-----	-----	----
BIGMACHINE	Windows NT	1381 guest	00 00:02:30	00 00:02:03

## 17.27 SHOW SYMBOL

The SHOW SYMBOL command displays the value of the specified symbol.

### Format:

**SHOW SYMBOL [symbol-name]**

### Parameters:

**symbol-name**

Specifies the name of the symbol whose value you want to display. This parameter is required unless you specify the /ALL qualifier. You may also specify the \* (wildcard) character in place of the /ALL qualifier. You cannot use SHOW SYMBOL to examine array-symbol elements (for example, ARRAY{5}).

### Qualifiers:

**/ALL**

Displays the values of all symbols in the specified symbol table (/LOCAL or /GLOBAL).

**/FULL**

Displays complete information concerning symbol including datatype. If the symbol represents a structure then the individual structure field names will be shown as well.

**/GLOBAL**

Searches only the global symbol table for the specified symbol name. If you specify both /ALL and /GLOBAL, all symbol values in the global symbol table are displayed.

**/LOCAL**

Searches only the local symbol table for the specified symbol name. If you specify both /ALL and /LOCAL, all symbol values in the local table are displayed.

### Example:

```
$ SHOW SYMBOL COUNT/FULL
```

```
COUNT = 1      Hex = 00000001
Symbol Type: Integer
Symbol Scope: Local, Level: 0
Symbol is declared
```

```
$ STRUCTURE LOCATION
```

```
STRING NAME 30
STRING ADDRESS 30
STRING CITY 20
STRING STATE 2
STRING ZIP_CODE 5
ENDSTRUCTURE
```

```
$ DECLARE SYMBOL LOCATION LOCATION_RECORD
```

```
$ LOCATION_RECORD|NAME="Adv Systems Concepts"
```

```
$ LOCATION_RECORD|ADDRESS="33-41 Newark St"
```

```
$ LOCATION_RECORD|CITY="Hoboken"
```

```
$ LOCATION_RECORD|STATE="NJ"
```

```
$ LOCATION_RECORD|ZIP_CODE="07030"
```

```
$ SHOW SYMBOL/FULL LOCATION_RECORD
```

```
LOCATION_RECORD (Instance of Structure LOCATION)
```

```
LOCATION_RECORD|NAME: Adv Systems Concepts
```

```
LOCATION_RECORD|ADDRESS: 33-41 Newark St
```

```
LOCATION_RECORD|CITY: Hoboken
```

```
LOCATION_RECORD|STATE: NJ
```

```
LOCATION_RECORD|ZIP_CODE: 07030
```

```
Symbol Scope: Local, Level: 0
```

```
Symbol is declared
```

## 17.28 SHOW SYSTEM

The SHOW SYSTEM command displays current system information concerning active processes and their threads.

**Format:**

**SHOW SYSTEM**

**Qualifiers:**

**/ON=machine**

This qualifier allows the command to be directed to a named machine within the current domain.

**/OUTPUT=filename**

Directs the output of the SHOW SYSTEM command to the specified file. By default, the output will be displayed on the console window.

```

XLNT Session for User: JAlfrunti
Licensed to: Xlnt User (1 license)
$ SHOW SYSTEM
Windows NT 5.0 on ATHENA 3/20/2006 2:42:50 PM Uptime: 4 22:54:51.999
Pid Process Handles Threads Prio Work Set PageFlts Cpu
00000000 Idle 0 1 0 16384 1 4 17:46:54.640
00000008 System 204 46 8 221184 26502 0 00:01:20.890
00000090 SMSS 33 6 11 393216 629 0 00:00:00.656
000000A8 CSRSS 673 13 13 2838528 95375 0 00:08:04.234
000000A4 WINLOGON 460 18 13 11718656 936493 0 00:00:46.937
000000D8 SERVICES 635 37 9 8474624 25277 0 00:00:12.046
000000E4 LSASS 389 19 9 3305472 150956 0 00:03:08.187
000019C suchost 408 11 8 4939776 6223 0 00:00:01.705
00001B8 spoolsv 187 12 8 7180288 9002 0 00:00:05.453
00001F8 msdtc 213 23 8 6053888 1705 0 00:00:00.281
0000274 DWRCS 111 8 8 4653056 6549 0 00:00:04.562
0000280 suchost 444 27 8 8097792 4046 0 00:00:00.656
00002A0 FrameworkServic 260 11 8 6983680 46985 0 00:00:02.125
0000318 explorer 494 18 8 2306048 414362 0 00:04:49.734
0000334 Mcshield 198 19 13 22900736 953443 0 00:04:09.796
000034C UsTaskMgr 130 11 8 311296 48669 0 00:00:00.140
0000364 naPrdMgr 107 4 8 1134592 3529 0 00:00:00.046
0000390 mdm 118 5 8 3653632 1226 0 00:00:00.640
00003B4 omtsreco 60 4 8 6590464 1617 0 00:00:00.093
00003FC shstat 62 6 8 602112 95427 0 00:00:00.265
00003E0 UpdaterUI 97 4 8 380928 1314796 0 00:00:00.531
000041C atiptaxx 83 2 8 3866624 1038 0 00:00:00.203
0000444 Desk95 67 1 8 2961408 720 0 01:25:38.281
0000484 AcroTray 18 1 8 1400832 342 0 00:00:00.015
00002D0 regsvc 77 3 8 3022848 754 0 00:00:00.046
00003DC mstask 131 7 8 5439488 1518 0 00:00:00.109
00004C8 userdump 47 4 8 1183744 290 0 00:00:00.015
00004FC WinMgmt 309 10 8 2502656 89526 0 00:00:44.843
0000300 suchost 414 7 8 13750272 89811 0 00:00:17.937
000050C heremote 161 8 8 8679424 4761 0 00:00:19.359
000053C mgsvc 204 22 8 6008832 7739 0 00:00:00.374
00004DC wsauc1t 164 3 8 5443584 1339 0 00:00:00.171
00006D8 LOCATOR 56 3 8 3854336 1237 0 00:00:00.109
0000410 abatadmin 315 6 8 28913664 62512 0 00:03:25.499
0000944 OUTLOOK 567 16 8 8728576 176253 0 00:40:51.124
000031C CMD 21 1 8 49152 648 0 00:00:00.031
0000750 scardsvr 55 5 8 1409024 396 0 00:00:00.031
0000920 XlntCli 109 1 8 77824 3517 0 00:00:00.234
00008E0 mstsc 196 12 8 1167360 82199 0 00:00:49.593
00009D0 devenv 419 13 8 46669824 113731 0 00:00:41.515
0000954 AbatEAgent 331 21 8 15642624 9926 0 00:00:06.109
0000754 AbatJss 572 23 8 17383424 13810 0 00:00:19.859
0000790 devenv 299 10 8 5701632 31761 0 00:00:03.812
0000980 XlntCli 109 1 8 77824 20508 0 00:00:00.906
00006E4 WINWORD 272 6 8 13352960 40827 0 00:00:15.109
00007D4 autorun 44 2 8 3702784 1215 0 00:00:00.078
0000824 _autorun 127 4 8 8601600 2369 0 00:00:00.265
0000834 msixec 103 4 8 7266304 8262 0 00:00:12.562
0000830 XlntCli 44 1 8 69632 705 0 00:00:00.015
000075C XlntCli 50 2 8 2609152 772 0 00:00:00.031
0000900 XlntCli 50 2 8 2605056 771 0 00:00:00.031
0000738 XlntCli 50 2 8 2605056 771 0 00:00:00.031
0000904 XlntCli 101 2 8 3047424 1193 0 00:00:00.062
0000000 _Total 10848 508 0 330551296 4914033 4 20:23:31.999

```

Figure 24. SHOW SYSTEM

## 17.29 SHOW TIME

The SHOW TIME command displays the current date and time.

**Format:**

**SHOW TIME**

**Example:**

**\$ SHOW TIME**

Thu 29-Aug-1996 16:37:51.664

## 17.30 SHOW USER

The SHOW USER command displays information concerning those local and remote users who are currently logged into your computer system.

**Format:**

**SHOW USER/qualifier**

**Qualifier:**

**/OUTPUT=filename**

Directs the output of the SHOW USER command to the specified file. By default, the output will be displayed on the console window.

**Example:**

**\$ SHOW USER**

Windows NT 4.0 on NODE 3/13/96 11:27:58 AM Uptime: 22 21:20:44.999

XLNT Local User	Mode	Current PID
Guest	Interactive	0000009D

## 17.31 SHOW VARIABLE

The SHOW VARIABLE command is used to display environment variables.

**Format:**

**SHOW VARIABLE environment-variable**

**Parameters:**

**environment-variable**

The desired environment variable to display.

**Example:**

**\$ SHOW VARIABLE PATH**

```
PATH = C:\WINNT35\SYSTEM32;C:\WINNT35;C:\WINDOWS;C:\WINDOWS\SYSTEM;  
C:\MSOFFICE;C:\MSOFFICE\WINWORD;C:\MSDEV\BIN;C:\ASCII\XLNT
```

# 18 Built-in (or Lexical) Functions

This section summarizes the built-in (or lexical) functions that are provided by XLNT.

Lexicals are built-in XLNT functions that can be used as operands in expressions. The term “lexical” refers to the function’s inline ability to be substituted anywhere within an expression. Lexicals always return a value. Depending on the lexical the value may be a TRUE or FALSE indication as to whether the lexical worked or some information that has been requested. XLNT provides many different types of lexicals, although some lexicals can be categorized into groups. For example, the Registry lexicals or DLL loading lexicals would be two groups of common lexicals. When reading the argument descriptions of lexicals it is important to note what is being requested. Some arguments require a symbol, some require a string, some allow a choice. When an argument requires a symbol, for example – context usage, you must supply a symbol. When an argument requires a string, you must either supply a quoted string or use explicit symbol substitution. For example,....

## DLL Loading Lexicals

DLL Loading lexicals allow for one or more DLL's to be loaded and their routines to be exported for direct use by XLNT procedures. The F\$LOADLIBRARY lexical provides the ability to load a DLL. The single argument provided is a fully qualified path and filename of the DLL to be loaded. If successful, a handle to the DLL is returned in a user provided symbol. If the symbol is local, then the DLL is implicitly freed when the command procedure level exits. If the symbol is global then the DLL is implicitly freed when the command procedure itself exits. You can determine whether a DLL is already loaded by invoking the F\$CHECKLIBRARY routine. To free a loaded DLL, call the F\$FREELIBRARY routine.

## Registry Lexicals

Registry lexicals require very similar arguments between the differing routines. The “hive” argument can be any Microsoft hive name (for example, HK\_LOCAL\_MACHINE) or abbreviation (HKLM). The following table provides a list of the abbreviations for each hive.

<u>Abbreviation</u>	<u>Full-Name Hive</u>
HKLM	HK_LOCAL_MACHINE
HKCU	HK_CURRENT_USER
HKCR	HK_CLASSES_ROOT
HKUSERS	HK_USERS
HKDD	HK_DYN_DATA
HKCC	HK_CURRENT_CONFIG
HKPD	HK_PERFORMANCE_DATA

To specify a remote machine’s hive you use UNC style syntax and specify \\machine\hive. When specifying a registry-key and value-name, the syntax is equally straightforward. Use key\subkey\subkey\...\value-name. If no value-name is to be specified then an ending slash defines the key. If the value-name is NONAME then you would use this syntax: key\subkey\...\ (two trailing slashes indicate that the value-name is NONAME). As with all NT object access, you may require permissions and/or rights to perform various Registry operations.

# 18.1 F\$ADDREGISTRY Lexical

The F\$ADDREGISTRY lexical adds a registry key (and optionally, a value name and value) to a hive.

**Format:**

**F\$ADDREGISTRY (hive, key, [reg-datatype, value])**

**Return Value:**

A character string containing the value "TRUE" or "FALSE" depending on whether the lexical succeeded or failed in its operation.

**Arguments:**

**hive**

A quoted string containing one Microsoft valid hive specification (for example, HK\_LOCAL\_MACHINE or HKLM). Please see the "Registry Lexicals" discussion for the table of hives and their abbreviations. To specify a remote machine, use UNC format and specify the \\machine\hive format.

**key**

A quoted string defining both the actual key and any associated value name using the syntax: key\sub-key\sub-sub-key...\value-name

If no value name is present then an ending single slash defines the key portion. If a NONAME value-name is to be associated with the key then two slashes are used. If your value name contains embedded slashes you must always include the leading slash on the beginning of the value-name string.

**reg-datatype**

(Optional) A quoted string consisting of one of the following registry datatypes: REG\_DWORD, REG\_SZ, REG\_MULTI\_SZ, REG\_BINARY, REG\_EXPAND\_SZ

**value**

(Optional) If reg-datatype is specified, the actual value associated with the key and value-name to be added. If REG\_DWORD is specified as the datatype, then this value must be specified as a longword. the other datatypes require that this value be expressed as a character string (including any optional radix).

**Description:**

This lexical is used to create registry keys and/or store a value in a registry key/value name.

**Examples:**

```
$ A=F$ADDREGISTRY ("HKEY_CURRENT_USER", -  
  "SOFTWARE\ASC\MYKEY\VALUE-NAME", -  
  "REG_DWORD", %X15)
```

This example adds the key SOFTWARE\ASC\MYKEY with a value name of VALUE-NAME to the hive HKEY\_CURRENT\_USER. The datatype associated with VALUE-NAME is REG\_DWORD and a hexadecimal value of 15 is associated with that value name.

```
$ A=F$ADDREGISTRY ("HKEY_CURRENT_USER", -  
  "SOFTWARE\ASC\MYKEY\\", -  
  "REG_DWORD", %X15)
```

This example is similar to the one above except that a NONAME value-name has been created and associated with the REG\_DWORD value 15.

**See also:**

**F\$CHANGeregistry F\$DELETERegistry F\$LOOKUPRegistry**

## 18.2 F\$CHANGEREISTRY Lexical

The F\$CHANGEREISTRY lexical changes a value for a specific key and value-name within a hive.

### Format:

**F\$CHANGEREISTRY (hive, key, reg-datatype, value)**

### Return Value:

A character string containing the value "TRUE" or "FALSE" depending on whether the lexical succeeded or failed in its operation.

### Arguments:

#### hive

A quoted string containing one Microsoft valid hive specification (for example, HK\_LOCAL\_MACHINE or HKLM). Please see the "Registry Lexicals" discussion for the table of hives and their abbreviations. To specify a remote machine, use UNC format and specify the \\machine\hive format.

#### key

A quoted string defining both the actual key and any associated value name using the syntax: key\sub-key\sub-sub-key...\value-name

If a NONAME value-name is to be associated with the key then two slashes are used. If your value name contains embedded slashes you must always include the leading slash on the beginning of the value-name string.

#### reg-datatype

A quoted string consisting of one of the following registry datatypes: REG\_DWORD, REG\_SZ, REG\_MULTI\_SZ, REG\_BINARY, REG\_EXPAND\_SZ

#### value

The actual value associated with the key and value-name to be added. If REG\_DWORD is specified as the datatype, then this value must be specified as a longword. the other datatypes require that this value be expressed as a character string (including any optional radix).

### Examples:

```
$ A=F$CHANGEREISTRY ("HKEY_CURRENT_USER", -  
                    "SOFTWARE\ASCIMYKEY\VALUE-NAME", -  
                    "REG_DWORD", %X21)
```

This example changes the value of the key SOFTWARE\ASCIMYKEY and value-name VALUE-NAME to hex 21.

### See also:

[F\\$ADDREGISTRY](#) [F\\$DELETEREGISTRY](#) [F\\$LOOKUPREGISTRY](#)

## 18.3 F\$CHECKLIBRARY Lexical

The F\$CHECKLIBRARY lexical is used to determine whether a DLL has been loaded.

**Format:**

**F\$CHECKLIBRARY ( file-spec )**

**Return Value:**

A character string containing the value TRUE or FALSE depending on whether the DLL is loaded.

**Argument:**

**file-spec**

The file-specification (including path) of the DLL to be checked.

**Example:**

**\$ A=F\$CHECKLIBRARY ("C:\MYDLL.DLL")**

This command verifies that the DLL named MYDLL.DLL is loaded. The symbol A will be either TRUE or FALSE depending on whether that DLL is loaded.

**See also:**

[F\\$FREELIBRARY](#) [F\\$LOADLIBRARY](#)

## 18.4 F\$CVSI Lexical

The F\$CVSI lexical converts the specified bits in a character string to a signed number.

**Format:**

**F\$CVSI (start-bit,number-of-bits,string)**

**Return Value:**

The integer equivalent of the extracted bit field, converted as a signed value.

**Arguments:**

**start-bit**

Specifies the offset of the first bit to be extracted. The rightmost bit of a string is considered position 0 for determining offset. Specify the offset as a positive non-zero integer expression.

**number-of-bits**

Specify the length of the bit string to be extracted.

**string**

Specifies the string from which the bits are taken.

**Example:**

\$ A = "12"

\$ X = F\$CVSI (0,4,A)

**See also:**

[F\\$CVUI](#)

## 18.5 F\$CVTIME Lexical

The F\$CVTIME lexical is used to convert between various time formats.

### Format:

**F\$CVTIME ([input-time] [,output-time-format] [,output-field])**

### Return Value:

A character string containing the requested information.

### Arguments:

#### input-time

Specifies a string containing absolute time (*dd-mmm-yyyy:hh:mm:ss.mms*) or the strings TODAY, TOMORROW, or YESTERDAY. This argument must be specified as a character string expression. If the argument is omitted or is specified as a null string (two double quotes together), the current system date and time will be used.

#### output-time-format

Specifies the time format for the information to be returned. It must be specified as one of the following string expressions:

**ABSOLUTE** - the requested information should be returned in absolute time format (*dd-mmm-yyyy hh:mm:ss.mms*). The term *mms* refers to milliseconds.

**COMPARISON** - the requested information should be returned in comparison format (*yyyy-mm-dd hh:mm:ss.mms*).

**JULIAN** - the requested information should be returned in julian date format (*dddyy, yyddd, dddyddd, yyyyddd*).

If this argument is omitted, COMPARISON format is the default.

#### output-field

Specifies a character string expression denoting the portion of the date/time string that is to be returned. When this argument is omitted entirely, the default for ABSOLUTE and COMPARISON is DATETIME, and the default for JULIAN is YEAR. One of the following, non-abbreviated values can be specified:

**DATE** - returns the date portion only or in Julian format (DDDDYYYY)

**MONTH** - returns the month only

**DATETIME** (default) - returns the entire date/time string

**SECOND** - returns the second of the minute only

**DAY** - returns the day of the month only or in Julian format (DDDDYY)

**TIME** - returns the time only

**HOUR** - returns the hour only

**YEAR** - returns the year only or in Julian format (YYDDD)

**MINUTE** - returns the minute of the hour only

**WEEKDAY** - returns the weekday

**YYYYDDD** - YYYYDDD for Julian format only

### Examples:

```
$ TIME = F$TIME()
$ SHOW SYM TIME
TIME = "28-Dec-1995 13:35:22.310"
$ TIME=F$CVTIME(TIME)
TIME = "1995-12-28 13:42:18.268"
```

This example shows how you can convert an absolute time into a comparison time.

**See also:**

[F\\$FORMATDATE](#) [F\\$FORMATIME](#) [F\\$TIME](#)

## 18.6 F\$CVUI Lexical

The F\$CVUI lexical converts the specified bits in a character string to an unsigned number.

**Format:**

**F\$CVUI (start-bit,number-of-bits,string)**

**Return Value:**

The integer equivalent of the extracted bit field, converted as a unsigned value.

**Arguments:**

**start-bit**

Specifies the offset of the first bit to be extracted. The rightmost bit of a string is considered position 0 for determining offset. Specify the offset as a positive non-zero integer expression.

**number-of-bits**

Specify the length of the bit string to be extracted.

**string**

Specifies the string from which the bits are taken.

**Example:**

\$ A = "12"

\$ X = F\$CVUI (0,4,A)

**See also:**

[F\\$CVSI](#)

## 18.7 F\$DELETEREGISTRY Lexical

The F\$DELETEREGISTRY lexical deletes a specific key (and all sub-keys and value-names) in a registry.

### Format:

**F\$DELETEREGISTRY (hive, key)**

### Return Value:

A character string containing the value "TRUE" or "FALSE" depending on whether the lexical succeeded or failed in its operation.

### Arguments:

#### hive

A quoted string containing one Microsoft valid hive specification (for example, HK\_LOCAL\_MACHINE or HKLM). Please see the "Registry Lexicals" discussion for the table of hives and their abbreviations. To specify a remote machine, use UNC format and specify the \\machine\hive format.

#### key

A quoted string defining both the actual key and any associated value name using the syntax: key\sub-key\sub-sub-key...\value-name

If a NONAME value-name is to be associated with the key then two slashes are used. Please note that all sub-keys and value-name associated with the named key are deleted as well. If your value name contains embedded slashes you must always include the leading slash on the beginning of the value-name string.

### Examples:

```
$ A=F$DELETEREGISTRY ("HKEY_CURRENT_USER", -  
"SOFTWARE\ASCIMYKEY")
```

This example deletes the key SOFTWARE\ASCIMYKEY and all sub-keys and value-name.

### See also:

[F\\$ADDREGISTRY](#) [F\\$CHANGEREGISTRY](#) [F\\$LOOKUPREGISTRY](#)

## 18.8 F\$DIRECTORY Lexical

The F\$DIRECTORY lexical returns the current device and directory specification.

**Format:**

**F\$DIRECTORY ( )**

**Return Value:**

A character string containing the current device and directory path. A UNC path may also be returned if the current directory includes a machine specification.

**Example:**

```
$ CURDIR = F$DIRECTORY ( )  
$ SHOW SYM CURDIR  
CURDIR = "C:\ASC\NEWXLNT"
```

This example invokes F\$DIRECTORY and the result is assigned to the symbol CURDIR. That symbol contains the current device/directory path.

## 18.9 F\$EDIT Lexical

The F\$EDIT lexical edits the character string based on the items in the **edit-list** argument.

### Format:

**F\$EDIT(string, edit-list)**

### Return Value:

A character string containing the edited string is returned.

### Arguments:

#### string

Specifies a character string to be edited.

#### edit-list

Specifies a character string containing one or more of the following keywords which specify the types of edits to be made:

**COLLAPSE** - Removes all spaces or tabs.

**COMPRESS** - Replaces multiple spaces/tabs with a single space.

**LOWERCASE** - Changes all uppercase characters to lowercase.

**TRIM** - Removes leading and trailing spaces or tabs.

**UNCOMMENT** - Remove comment symbol (!)

**UPCASE** - Changes all lowercase characters to uppercase.

### Example:

```
$ LINE = " THIS LINE CONTAINS A "" QUOTED "" WORD"
$ SHOW SYMBOL LINE
LINE = " THIS LINE CONTAINS A " QUOTED " WORD"
$ NEW_LINE = F$EDIT(LINE, "COMPRESS, TRIM")
$ SHOW SYMBOL NEW_LINE
NEW_LINE = "THIS LINE CONTAINS A " QUOTED " WORD"
```

This example makes two points. First, the string NEW\_LINE has been compressed and trimmed of excessive white space. Second, the extra white space around the quoted string has not been disturbed.

```
$ LOOP:
$ READ/END_OF_FILE = DONE INPUT_FILE RECORD
$ RECORD = F$EDIT(RECORD, "TRIM, UPCASE")
$ WRITE OUTPUT_FILE RECORD
$ GOTO LOOP
```

This example reads records from a file and both trims of records of excessive white space as well as converting lower case data to upper case.

## 18.10 F\$ELEMENT Lexical

The F\$ELEMENT lexical extracts one element from a string of elements.

### Format:

**F\$ELEMENT(element-number,delimiter,string)**

### Return Value:

A character string containing the specified element.

### Arguments:

#### **element-number**

Specifies the number of the element to be extracted (beginning at element zero). If the **element-number** argument exceeds the number of elements in the string, F\$ELEMENT returns the delimiter.

#### **delimiter**

Specifies a character used to separate the elements in the string.

#### **string**

Specifies a string containing a delimited list of elements.

### Example:

```
$ DAY_LIST = "MON/TUE/WED/THU/FRI/SAT/SUN"
$ INQUIRE DAY "ENTER DAY (MON TUE WED THU FRI SAT SUN)"
$ NUM = 0
$ LOOP:
$   LABEL = F$ELEMENT(NUM,"/",DAY_LIST)
$   IF LABEL .EQS. "/" THEN GOTO END
$   IF DAY .EQS. LABEL THEN GOTO 'LABEL'
$   NUM = NUM +1
$   GOTO LOOP
$
$ MON:
```

This example sets up a loop to test an input value against the day of the week. Note that when the day of the week is found the value itself can be used as a target for a corresponding label.

## 18.11 F\$ENUMDOMAIN Lexical

The F\$ENUMDOMAIN lexical returns a known domain name or provides the ability to check whether a string represents a valid domain name.

### Format:

**F\$ENUMDOMAIN ([domain-name], [context])**

### Return Value:

A character string that represents a known domain name or a null value that indicates the end of the domain name enumeration or the specified search string is not a domain name.

### Arguments:

#### **domain-name**

(Optional) If specified, the character string should represent a domain name that is to be verified by the lexical. If omitted, an enumeration of known domain names is assumed (in which case, the context argument should be specified).

#### **context**

(Optional) If specified, this integer symbol should be initialized to zero to begin a domain name enumeration. Once the enumeration has begun, the symbol should not be updated by your procedure. If the symbol specified does not exist, a new symbol of proper context will be created.

### Example:

```
$loop:
$ domain = f$enumdomain (, context)
$ if domain .eqs. "" then goto endloop
$ write $stdout "Domain name is "domain"
$ goto loop
$endloop:
$ exit
```

This simple example shows how the lexical can be used to enumerate all known domain names. If the returned string is a null-string, the enumeration is completed.

```
$ domain = f$enumdomain ("test-domain")
$ if domain .nes. "" then domain-correct
$ write $stdout "The domain specified is not a domain"
```

This example shows how you might check a possible domain name string and determine whether the domain name specified is a legal and known domain name.

### See also:

[F\\$ENUMMACHINE](#) [F\\$ENUMSHAREPOINT](#)

## 18.12 F\$ENUMGROUP Lexical

The F\$ENUMGROUP lexical returns a group name for a domain or specific machine. This lexical is meant to be used to retrieve a list of existing groups. F\$GROUPINFO provides support for further group information. **(Windows NT Only)**

### Format:

**F\$ENUMGROUP ([domain-name], [machine-name], global-local-flag, context)**

### Return Value:

A character string that represents a group name.

### Arguments:

#### domain-name

(Optional) If specified, a character string that represents a valid and known domain name. If both domain and machine-name are omitted, the default domain this machine is part of is used.

#### machine-name

(Optional) If specified, a character string that represents a valid and known machine name. If both domain and machine-name are omitted, the default domain this machine is part of is used.

#### global-local-flag

This argument indicates whether a local or global group should be enumerated. A string value beginning with the letter "G" indicates global. Any other value indicates local. If the argument is omitted, global is assumed.

#### context

This integer symbol should be initialized to zero to begin a group name enumeration. Once the enumeration has begun, the symbol should not be updated by your procedure. If the symbol specified does not exist, a new symbol of proper context will be created.

### Example:

```
$ for (,)
$   group = f$enumgroup (,,"G",context)
$   if group .eqs. "" then leave
$   sho sym group
$ endfor
$ exit
A = "Domain Admins"
A = "Domain Guests"
A = "Domain Users"
A = "NT-ASCII Global BQMS"
A = "NTBETAADMIN"
```

This example loops through all the global groups in the default login domain.

## 18.13 F\$ENUMMACHINE Lexical

The F\$ENUNMACHINE lexical returns a known machine-name within a specified domain *or* provides the ability to check whether a string represents a valid machine name.

### Format:

**F\$ENUMMACHINE (domain-name, [machine-name], [context], [machine-type])**

### Return Value:

A character string that represents a known machine name or a null value that indicates the end of the machine name enumeration or the specified search string is not a valid machine name.

### Arguments:

#### domain-name

The character string must represent a valid and known domain name.

#### machine-name

(Optional) If specified allows the caller to determine whether the specified character string represents a known machine name within the specified domain. If omitted, the context argument is normally used to specify an enumeration of known machine names within the specified domain.

#### context

(Optional) If specified, this integer symbol should be initialized to zero to begin a machine name enumeration. Once the enumeration has begun, the symbol should not be updated by your procedure. If the symbol specified does not exist, a new symbol of proper context will be created.

#### machine-type

(Optional) If specified this string symbol will be set with the characteristics of the enumerated machine. Multiple characteristics are returned separated by commas. The following list indicates possible machine types: "Workstation", "Server", "SQL Server", "Domain Controller", "Backup Domain Controller", "Time Server", "Apple File Protocol", "Novell Server", "Domain Member", "Printer Server", "Dial-In Server", "Xenix Server", "UNIX Server", "NT", "Windows for Workgroups", "Server MFPN", "NT Server", "Potential Browser", "Backup Browser", "Master Browser", "OSF Server", "VMS Server", "Windows Client", "DFS Root", "Alternate Transport", "Domain Enumerator".

### Example:

```
$loop:
$ machine = f$enummachine ("test-domain", ,context)
$ if machine .eqs. "" then goto endloop
$ write $stdout "Machine name is "machine"
$ goto loop
$endloop:
$ exit
```

This simple example shows how the lexical can be used to enumerate all known machine names within a specified domain. If the returned string is a null-string, the enumeration is completed.

```
$ machine = f$enummachine ("test-domain","test-machine")
$ if machine .nes. "" then machine-correct
$ write $stdout "Themachine specified is not a legal machine"
```

This example shows how you might check a possible machine name string and determine whether the machine name specified is legal.

**See also:**

[F\\$ENUMDOMAIN](#) [F\\$ENUMSHAREPOINT](#)

## 18.14 F\$ENUMSHAREPOINT Lexical

The F\$ENUMSHAREPOINT lexical returns a known share name *or* provides the ability to check whether a string represents a valid share name.

### Format:

**F\$ENUMSHAREPOINT (domain-name, machine-name, [share-name], [context])**

### Return Value:

A character string that represents a known share name or a null value that indicates the end of the share name enumeration or the specified search string is not a valid share name.

### Arguments:

#### domain-name

The character string must represent a valid and known domain name.

#### machine-name

The character string must represent a valid and known machine name within the specified domain.

#### share-name

(Optional) If specified allows the caller to determine whether the specified character string represents a known share name within the specified machine. If omitted, the context argument is normally used to specify an enumeration of known share names within the specified machine.

#### context

(Optional) If specified, this integer symbol should be initialized to zero to begin a share name enumeration. Once the enumeration has begun, the symbol should not be updated by your procedure. If the symbol specified does not exist, a new symbol of proper context will be created.

### Example:

```
$loop:  
$ share = f$enumsharepoint ("test-domain", "machine1",,context)  
$ if share .eqs. "" then goto endloop  
$ write $stdout "Share name is "share"  
$ goto loop  
$endloop:  
$ exit
```

This simple example shows how the lexical can be used to enumerate all known share names within a specified machine. If the returned string is a null-string, the enumeration is completed.

```
$ share = f$enumsharepoint ("test-domain","machine","test-share")  
$ if share .nes. "" then share-correct  
$ write $stdout "The share specified is not a legal share name"
```

This example shows how you might check a possible share name string and determine whether the share name specified is legal.

### See also:

[F\\$ENUMDOMAIN](#) [F\\$ENUMMACHINE](#)

## 18.15 F\$ENUMUSER Lexical

The F\$ENUMUSER lexical returns a user name for a domain or specific machine. This lexical is meant to be used to retrieve a list of existing users. F\$USERINFO provides support for further user account information. **(Windows NT Only)**

### Format:

**F\$ENUMUSER ([domain-name], [machine-name], context, [logon-type])**

### Return Value:

A character string that represents a user name. If "logon-type" is specified, a fully formed machine\user-name will be returned. A null string indicates the end of the enumeration.

### Arguments:

#### **domain-name**

(Optional) If specified, a character string that represents a valid and known domain name. If both domain and machine-name are omitted, the default domain this machine is part of is used.

#### **machine-name**

(Optional) If specified, a character string that represents a valid and known machine name. If both domain and machine-name are omitted, the default domain this machine is part of is used.

#### **context**

This integer symbol should be initialized to zero to begin a user name enumeration. Once the enumeration has begun, the symbol should not be updated by your procedure. If the symbol specified does not exist, a new symbol of proper context will be created.

#### **logon-type**

(Optional) If specified, allows you to enumerate all desktop or network resource logons on a specified machine. Valid keywords are: DESKTOP and RESOURCES. Note: Once an enumeration is performed you cannot switch to a different logon-type without restarting the enumeration.

### Example:

```
$ FOR (context=0,,)
$  USERNAME = F$ENUMUSER(,,CONTEXT)
$  IF USERNAME .EQS. "" THEN LEAVE
$  WRITE $STDOUT  USERNAME
$ ENDFOR
$ EXIT
```

This example enumerates all users in the current domain and displays their usernames.

## 18.16 F\$ENVIRONMENT Lexical

The F\$ENVIRONMENT lexical returns information about the current XLNT command environment.

**Format:**

**F\$ENVIRONMENT** (item)

**Return Value:**

Information that corresponds to the specified item. The return value can be either an integer or a character string, depending on the specified item.

**Arguments:**

**item**

Specifies the type of information to be returned. One of the following keywords (non-abbreviated) can be specified:

**DEFAULT** (string) - current default device and directory name.

**DEPTH** (integer) - current command procedure depth.

**INTERACTIVE** (string) - TRUE if the process is executing interactively.

**MAX\_DEPTH** (integer) - maximum allowable command procedure depth.

**MESSAGE** (string) - current XLNT Message Display

**PROCEDURE** (string) - file name of the current command procedure. If used interactively, a null string ("" ) is returned.

**PROMPT** (string) - current XLNT prompt string.

**SYMBOL\_SCOPE** (string) - current symbol scope

**VERIFY\_PROCEDURE** (string) - TRUE if procedure verification is in effect (SET VERIFY).

**Example:**

```
$ WRITE $STDOUT F$ENV("MAX_DEPTH")
32
```

This command shows the maximum depth that a command procedure can nest.

```
$ A = F$ENV("MESSAGE")
$ SET MESSAGE 'A
```

This command captures the current XLNT Message setting and allows you to reset the message setting via the SET MESSAGE command.

## 18.17 F\$EXTRACT Lexical

The F\$EXTRACT lexical extracts the specified characters from the specified string.

### Format:

**F\$EXTRACT (start, length, string)**

### Return Value:

The extracted string is returned.

### Arguments:

#### start

Specifies the offset of the starting character of the string to be extracted. The offset is the relative position of a character or a substring with respect to the beginning of the string. Offset positions begin with zero. The string always begins with the leftmost character.

#### length

Specifies the number of characters to extract; must be less than or equal to the size of the string. If the length specified exceeds the number of characters from the offset to the end of the string, F\$EXTRACT returns the characters from the offset to the end of the string.

#### string

Specifies the character string to be edited.

### Example:

```
$ NAME = "JOE SMITH"  
$ FIRST = F$EXTRACT(0,3,NAME)  
$ SHOW SYMBOL FIRST  
FIRST = "JOE"
```

This simple example extracts the first three characters from the symbol NAME.

```
$ P1 = "MYFILE.DAT"  
$ FILENAME = F$EXTRACT(0,F$LOCATE(".",P1),P1)
```

This portion of a command procedure shows how to locate a character within a string, and how to extract a substring ending at that location.

The lexical function F\$LOCATE gives the numeric value representing the offset position of a period in the character string value of P1. (The offset position of the period is equal to the length of the substring before the period.)

This F\$LOCATE function is used as an argument in the F\$EXTRACT function to specify the number of characters to extract from the string. If a procedure is invoked with the parameter MYFILE.DAT, these statements result in the symbol FILENAME being given the value MYFILE.

## 18.18 F\$FILE\_ATTRIBUTES Lexical

This lexical provides detailed information concerning the specified file.

### Format:

**F\$FILE\_ATTRIBUTES (file-spec, item-code)**

### Return Value:

A character string is returned the value of which depends on the item code specified.

### Arguments:

#### file-spec

A string containing a valid file specification. The file specification cannot contain any wildcard characters.

#### item-code

One of the following keywords:

- ACL** - Access Control list (String) **(Windows NT Only)**
- ADT** - Access Date/Time (Microsoft Format) (String)
- AADT** - Access Date/Time (Absolute Format) (String)
- ALQ** - Allocation Quantity (Integer)
- ARC** - Archived File (Boolean)
- BLS** - Block Size (Integer)
- CDT** - Creation Date/Time (Microsoft Format) (String)
- ACDT** - Creation Date/Time (Absolute Format) (String)
- CMP** - Compressed File (Boolean)
- DIR** - Directory File (Boolean)
- DVI** - Device (String)
- EOF** - End-Of-File (Integer)
- FST** - File System Type (String)
- HID** - Hidden File (Boolean)
- MBM** - Owner Member Number (Integer)
- MDN** - MS-DOS Filename (String)
- NRM** - Normal File (Boolean)
- ORG** - File Organization (String)
- PLT** - Platform (String)
- RDO** - Read Only File (Boolean)
- SYS** - System File (Boolean)
- TMP** - Temporary File (Boolean)
- WDT** - Write Date/Time (Microsoft Format) (String)
- AWDT** - Write Date/Time (Absolute Format) (String)

### Description:

This lexical provides the user with a item-code facility for determining the various attributes of a specified file. Many item codes provide a TRUE/FALSE indication concerning a file's attributes. Other item codes return information concerning the requested attribute. The file specified must exist. The ACL item code returns a string that contains zero or more permissions associated with the file. The permissions are formatted for acceptance by the SET PERMISSIONS command. Multiple permissions are separated by commas for easy parsing and handling. To retrieve permissions for other types of objects see the F\$PERMISSIONS lexical.

### Example:

```
$ A=F$FILE_ATTRIBUTES ("C:\ASCII\XLNT\TMP.TMP","ALQ")
$ SHOW SYMBOL A
A = 2475   Hex = 000009AB
```

The statement above retrieves the Allocation Quantity of the selected file. That value is then display via the SHOW SYMBOL command.

```
$ ACL = F$FILE_ATTRIBUTES("D:\XXX","ACL")
$ WRITE $STDOUT "Permissions are: "ACL' "
$ SET PERM/REVOKE/ACCOUNT=GUEST D:\XXX
$ SHOW PERM D:\XXX
$ SET PERM/ACCOUNT=("ACL') D:\XXX
```

This example retrieves a file's existing permissions. The file's permissions are then changed via the SET PERM/REVOKE command (and confirmed with the SHOW PERM command). Finally, the permissions are restored with the SET PERM which uses the returned ACL string to reconstruct the original permissions.

## 18.19 F\$FORMAT Lexical

This lexical provides higher-level formatting of numeric and character string data.

### Format:

**F\$FORMAT (formatting-string, [values,...])**

### Return Value:

A character string is returned representing the formatted data.

### Arguments:

#### formatting-string

A character string contains both string data and conversion specifications. A conversion specification is introduced with the percent (%) character and ended with the conversion control character. The formatting string follows the rules of the C language "sprintf" statement (except that "o", "e", "f" and "g" are not supported).

Specifically, the following characters are supported:

- d Expects a decimal integer.
- x Expects a hexadecimal integer.
- c Expects a single character.
- s Expects a string.
- i Expects an integer.

Specifically, the following escape sequences are supported:

- \a Bell (alert)
- \b Backspace
- \f Formfeed
- \n New line
- \r Carriage return
- \t Horizontal tab
- \v Vertical tab
- \' Single quotation mark
- \" Double quotation mark
- \\ Backslash
- \? Literal question mark

#### values,...

Zero or more values which match the conversion specifications (if any) in the formatting string argument.

### Example:

```
$ write $stdout f$format("My login username is %s",$username)
```

```
My login username is Guest
```

```
$ count = 100
```

```
$ write $stdout f$format("\tLoop Count is: %d\n", count)
```

```
Loop Count is: 100
```

## 18.20 F\$FORMATDATE Lexical

This lexical formats a date string from an absolute or combination time date string.

### Format:

**F\$FORMATDATE ([format-string],[date-time])**

### Return Value:

The value returned is a string representing the formatted date.

### Arguments:

#### format-string

(Optional) If omitted the system default format string is used (see the Control Panel/International icon for more information). If specified, this quoted string represents a picture conversion string which indicates how the date string is to be formatted. The following format characters are available:

<b>d</b>	Day of month as digits with no leading zero for single-digit days.
<b>dd</b>	Day of month as digits with leading zero for single-digit days.
<b>ddd</b>	Day of week as a three-letter abbreviation. The abbreviation is based on the locale.
<b>dddd</b>	Day of week as its full name. The full name is based on the locale.
<b>M</b>	Month as digits with no leading zero for single-digit month.
<b>MM</b>	Month as digits with leading zero for single-digit month.
<b>MMM</b>	Month as a three-letter abbreviation. The abbreviation is based on the locale.
<b>MMMM</b>	Month as its full name. The full name is based on the locale.
<b>y</b>	Year as the last two digits, but with no leading zero for years less than 10.
<b>yy</b>	Year as the last two digits, but with leading zero for years less than 10.
<b>yyyy</b>	Year using four digits.
<b>gg</b>	Period/era string.

In addition to the above special format characters, you can also insert any other single quoted characters that you wish to appear with the formatted date string.

#### date-time

(Optional) If omitted or not specified, the current date is used. If specified a valid absolute or combination date-time string must be specified.

### Example:

**\$ DATE=F\$FORMATDATE ("ddd', ' MMM dd yy", "30-AUG-1996")**

This example shows how you can produce the date string:

"Fri, Aug 30 96"

Note the single quoted characters and spaces within the format-string. They allow great flexibility in producing a formatted date string.

**See also:**

[F\\$CVTIME](#) [F\\$FORMATTIME](#) [F\\$TIME](#)

## 18.21 F\$FORMATTIME Lexical

This lexical formats a time string from an absolute or combination time date string.

### Format:

**F\$FORMATTIME** ([format-string],[date-time])

### Return Value:

The value returned is a string representing the formatted time.

### Arguments:

#### format-string

(Optional) If omitted the system default format string is used (see the Control Panel/International icon for more information). If specified, this quoted string represents a picture conversion string which indicates how the time string is to be formatted. The following format characters are available:

<b>h</b>	Hours with no leading zero for single-digit hours; 12 hour clock.
<b>hh</b>	Hours with leading zero for single-digit hours; 12 hour clock.
<b>H</b>	Hours with no leading zero for single-digit hours; 24 hour clock.
<b>HH</b>	Hours with leading zero for single-digit hours; 24 hour clock.
<b>m</b>	Minutes with no leading zero for single-digit minutes.
<b>mm</b>	Minutes with leading zero for single-digit minutes.
<b>s</b>	Seconds with no leading zero for single-digit second.
<b>ss</b>	Seconds with leading zero for single-digit second.
<b>t</b>	One character time marker string, such as A or P.
<b>tt</b>	Two character time marker string, such as AM or PM.

In addition to the above special format characters, you can also insert any other single quoted characters that you wish to appear with the formatted time string.

#### date-time

(Optional) If omitted or not specified, the current date is used. If specified a valid absolute or combination date-time string must be specified.

### Example:

**\$ D=F\$FORMATTIME ("hh':mm':ss tt", "30-AUG-1996:11:29:40")**

This example shows how you can produce the time string:

"11:29:40 AM"

Note the single quoted characters and spaces within the format-string. They allow great flexibility in producing a formatted time string.

### See also:

[F\\$CVTIME](#) [F\\$FORMATDATE](#) [F\\$TIME](#)

## 18.22 F\$FREELIBRARY Lexical

This lexical frees or unloads a loaded DLL.

**Format:**

**F\$FREELIBRARY (handle)**

**Return Value:**

A character string containing the value TRUE or FALSE as to the whether the operation succeeded.

**Argument:**

**handle**

The handle value returned when the library was loaded.

**Example:**

```
$ DLL_HANDLE=F$LOADLIBRARY("C:\MYDLL.DLL")  
$ A=F$FREELIBRARY (DLL_HANDLE)
```

This example loads and then frees the library MYDLL.

**See also:**

[F\\$CHECKLIBRARY](#) [F\\$LOADLIBRARY](#)

## 18.23 F\$GETBQI Lexical

This lexical provides programmatic information concerning Batch Queues and their jobs within the ActiveBatch sub-system.

### Format:

**F\$GETBQI (display-type, search-str, item [,flags]  
[,context] [,machine])**

### Return-Value:

A character string that contains the value of the information concerning the specified queue or job-entry and the item of information desired. A null-string normally means the information is not present or the search has ended.

### Arguments:

#### display-type

DISPLAY_QUEUE	Search information concerns queue
DISPLAY_ENTRY	Search information concerns job entry

#### search-str

depending on the *display-type* argument this string is either a queue-name or job-entry-id. If the *search-str* represents a queue-name then wildcards are allowed. If the *search-str* represents a job-entry-id then an asterisk representing all possible jobs can also be specified.

#### Item

One of the following item code strings:

<b>QUEUE_PRIORITY</b>	<b>Integer</b>
represents the default job queue priority	
<b>QUEUE_OSPRIORITY</b>	<b>Integer</b>
represents the default O/S scheduling priority	
<b>QUEUE_JOB_LIMIT</b>	<b>Integer</b>
represents the number of jobs which can execute concurrently.	
<b>QUEUE_EXQUEUE_NUMBER</b>	<b>Integer</b>
number of execution queues (if queue is generic)	
<b>QUEUE_ENTRY</b>	<b>Integer</b>
represents the job's entry id. This item is meant to be used when a wildcard search of all queued jobs within a queue is being requested. This item code returns each job in sequential fashion as it is actually on queue.	
<b>QUEUE_NAME</b>	<b>String</b>
the queue's name.	
<b>QUEUE_EXQUEUENAME</b>	<b>String</b>
one of more execution queue's by name.	
<b>QUEUE_OWNER</b>	<b>String</b>
the creator/owner of the queue.	
<b>QUEUE_CLI</b>	<b>String</b>
the default command shell to be used when executing a batch job.	
<b>QUEUE_RANGE_START</b>	<b>String</b>

the time range for starting a queue.	
<b>QUEUE_RANGE_STOP</b>	<b>String</b>
the time range for stopping a queue.	
<b>QUEUE_RANGE_OPEN</b>	<b>String</b>
the time range for opening a queue.	
<b>QUEUE_RANGE_CLOSE</b>	<b>String</b>
the time range for closing a queue.	
<b>QUEUE_DESCRIPTION</b>	<b>String</b>
a user description or comment about the queue.	
<b>QUEUE_GENERIC</b>	<b>Boolean</b>
If true, queue is generic.	
<b>QUEUE_EXECUTION</b>	<b>Boolean</b>
If true, queue is execution.	
<b>QUEUE_STARTED</b>	<b>Boolean</b>
If true, queue is started.	
<b>QUEUE_STOPPED</b>	<b>Boolean</b>
If true, queue is stopped.	
<b>QUEUE_PAUSED</b>	<b>Boolean</b>
If true, queue is paused.	
<b>QUEUE_PENDING_STOP</b>	<b>Boolean</b>
If true, queue is pending being stopped.	
<b>QUEUE_PENDING_START</b>	<b>Boolean</b>
If true, queue is pending start.	
<b>QUEUE_CLOSED</b>	<b>Boolean</b>
If true, queue is closed.	
<b>QUEUE_OPEN</b>	<b>Boolean</b>
If true, queue is open.	
<b>QUEUE_RETAIN_ALL</b>	<b>Boolean</b>
If true, queue retains all jobs.	
<b>QUEUE_RETAIN_ERROR</b>	<b>Boolean</b>
If true, queue retains only jobs that complete in error.	
<b>QUEUE_RETAIN_NONE</b>	<b>Boolean</b>
If true, queue retains no jobs.	
<b>ENTRY_PRIORITY</b>	<b>Integer</b>
represents the job's queued priority.	
<b>ENTRY_ID</b>	<b>Integer</b>
represents the job's entry id.	
<b>ENTRY_NUM_PARAMETERS</b>	<b>Integer</b>
represents the number of input parameters as specified with SUBMIT.	
<b>ENTRY_PARAMETERS</b>	<b>String</b>
represents the job's input parameters as specified with SUBMIT.	
<b>ENTRY_COMPLETION_STATUS</b>	<b>Integer</b>
represents the job's completion or exit status.	
<b>ENTRY_ENTRYNAME</b>	<b>String</b>

represents the job's name.

**ENTRY\_QUEUE\_NAME** String

represents the job's current queue.

**ENTRY\_MACHINE** String

represents the machine the entry is queued for submission to.

**ENTRY\_OWNER** String

represents the creator/owner of the job.

**ENTRY\_SERVER\_TO\_NOTIFY** String

represents the machine name of the NOTIFY\_USER.

**ENTRY\_USER\_TO\_NOTIFY** String

represents the username to be notified (residing on the machine noted by NOTIFY\_SERVER).

**ENTRY\_CLI** String

represents the command shell for this batch job.

**ENTRY\_LOGFILE** String

indicates the output log file specification of this batch job.

**ENTRY\_PROCEDURE** String

indicates the batch script file specification.

**ENTRY\_DESCRIPTION** String

indicates the description or comment concerning the batch job.

**ENTRY\_TIME\_DEFERRED** String

represents the starting or deferred time of the batch job.

**ENTRY\_DEFERRED** Boolean

If true, job is pending for deferred execution.

**ENTRY\_PENDING** Boolean

If true, job is pending execution.

**ENTRY\_EXECUTING** Boolean

If true, job is executing.

**ENTRY\_DONE** Boolean

If true, job has completed.

**ENTRY\_ABORTING** Boolean

If true, job is aborting.

**ENTRY\_HELD** Boolean

If true, job is being held.

**ENTRY\_PAUSED** Boolean

If true, job is paused.

**ENTRY\_RETAINED** Boolean

If true, job has been retained.

**ENTRY\_RESTART** Boolean

If true, job is restartable.

**ENTRY\_NOTIFY** Boolean

If true, user notification is enabled.

**ENTRY\_PRINT** Boolean

If enabled, log file will be automatically printed on job completion. (Feature not currently present).

**ENTRY\_KEEP** Boolean



## 18.24 F\$GETDVI Lexical

This lexical allows the caller to obtain detailed device information.

### Format:

**F\$GETDVI (device, item)**

### Return Value:

A character string that contains the value of the information concerning the specified device and the item of information desired. A null-string normally means the information is not present for this device.

### Arguments:

#### device

This argument represents a device name. The device can be a local disk device letter or a server/sharename using UNC syntax (i.e. \\server\share).

#### item

One of the following keywords:

##### **EXISTS (Boolean)**

Does device exist? This is the only item code that will not raise an exception if an invalid or unknown disk device is specified.

##### **NO\_SECTORS (Integer)**

Number of sectors per cluster

##### **SECTORSIZE (Integer)**

Sector size (in bytes)

##### **FREECLUSTERS (Integer)**

Free clusters on device

##### **TOTALCLUSTERS (Integer)**

Total clusters on device

##### **FREEBYTES (Integer)**

The number of bytes available on the disk device

##### **TOTALBYTES (Integer)**

Total storage size, in bytes, for the disk device

##### **SERIALNO (String)**

Serial number of the disk volume

##### **FILELEN (Integer)**

Length of filename portion (between the slashes) supported for this device. NTFS volumes support 255 characters. FAT volumes support 8.3 (filename and extension)

##### **DRIVETYPE (Integer)**

A value indicating the physical type of device. See *GetDriveType* in the Win32 Programming Guide for more details.

##### **VOLUME (String)**

The volume label of the device

##### **FILESYSTEM (String)**

The file system in use for this volume ("NTFS" or "FAT")

##### **REMOTEPath (String)**

If the device is remotely served through a network drive letter, this item will pass back the actual path (using UNC

notation)

**COMPRESSED (Boolean)**

A true/false indication as to whether the volume is compressed.

**COMPRESSIBLE (Boolean)**

A true/false indication as to whether the volume supports compression.

**ACLS (Boolean)**

A true/false indication as to whether the volume supports and enforces ACLs

**UNICODE (Boolean)**

A true/false indication as to whether the volume supports Unicode

**CASE\_SENSITIVE (Boolean)**

A true/false indication as to whether the volume supports case sensitivity

**CASE\_PRESERVED (Boolean)**

A true/false indication as to whether the volume preserves case for file names

**MNT (Boolean)**

A true/false indication of whether a disk is currently loaded in the device

**RMT (Boolean)**

A true/false indication of whether the device is local or remotely served

**ENCRYPTION (Boolean)**

A true/false indication of whether the file system supports encryption.

**SPARSE\_FILES (Boolean)**

A true/false indication of whether the file system supports sparse files.

**OBJECT\_IDS (Boolean)**

A true/false indication of whether the file system supports object identifiers.

**PARSE\_POINTS (Boolean)**A true/false indication of whether the file system supports parse points.

**VOLUME\_QUOTAS (Boolean)**

A true/false indication of whether the file system supports volume quotas.

**REMOTE\_STORAGE (Boolean)**

A true/false indication of whether the file system supports remote storage.

**Example:**

```
$ freeclusters=f$getdvi("c:","FREECLUSTERS")
```

```
$ show symbol freeclusters
```

```
FREECLUSTERS = 438   Hex = 000001B6
```

```
$ declare symbol ulong freebytes
```

```
$ freebytes=f$getdvi ("c:","FREEBYTES")
```

Two examples using F\$GETDVI. The first is straightforward in requesting the number of freeclusters. The second is a little more devious. By default, XLNT allocates an integer datatype when a symbol is used in a typeless fashion. If the value returned is greater than 2 billion bytes (or so), an integer will yield a negative value. That's why the symbol is first declared as ULONG since FREEBYTES is essentially an unsigned longword.

## 18.25 F\$GETDVIEX

This lexical allows the caller to obtain detailed device information. It is the same as the F\$GETDVI lexical, except that FREEBYTES and TOTALBYTES are returned as 64-bit integers.

### Format:

**F\$GETDVIEX (device, item)**

### Return Value:

A character string that contains the value of the information concerning the specified device and the item of information desired. A null-string normally means the information is not present for this device.

### Arguments:

#### device

This argument represents a device name. The device can be a local disk device letter or a server/sharename using UNC syntax (i.e. \\server\share).

#### item

One of the following keywords:

##### EXISTS (Boolean)

Does device exist? This is the only item code that will not raise an exception if an invalid or unknown disk device is specified.

##### NO\_SECTORS (Integer)

Number of sectors per cluster

##### SECTORSIZE (Integer)

Sector size (in bytes)

##### FREECLUSTERS (Integer)

Free clusters on device

##### TOTALCLUSTERS (Integer)

Total clusters on device

##### FREEBYTES (Int64)

The number of bytes available on the disk device

##### TOTALBYTES (Int64)

Total storage size, in bytes, for the disk device

##### SERIALNO (String)

Serial number of the disk volume

##### FILELEN (Integer)

Length of filename portion (between the slashes) supported for this device. NTFS volumes support 255 characters. FAT volumes support 8.3 (filename and extension)

##### DRIVETYPE (Integer)

A value indicating the physical type of device. See *GetDriveType* in the Win32 Programming Guide for more details.

##### VOLUME (String)

The volume label of the device

##### FILESYSTEM (String)

The file system in use for this volume (“NTFS” or “FAT”)

**REMOTEPATH (String)**

If the device is remotely served through a network drive letter, this item will pass back the actual path (using UNC notation)

**COMPRESSED (Boolean)**

A true/false indication as to whether the volume is compressed.

**COMPRESSIBLE (Boolean)**

A true/false indication as to whether the volume supports compression.

**ACLS (Boolean)**

A true/false indication as to whether the volume supports and enforces ACLs

**UNICODE (Boolean)**

A true/false indication as to whether the volume supports Unicode

**CASE\_SENSITIVE (Boolean)**

A true/false indication as to whether the volume supports case sensitivity

**CASE\_PRESERVED (Boolean)**

A true/false indication as to whether the volume preserves case for file names

**MNT (Boolean)**

A true/false indication of whether a disk is currently loaded in the device

**RMT (Boolean)**

A true/false indication of whether the device is local or remotely served

**ENCRYPTION (Boolean)**

A true/false indication of whether the file system supports encryption.

**SPARSE\_FILES (Boolean)**

A true/false indication of whether the file system supports sparse files.

**OBJECT\_IDS (Boolean)**

A true/false indication of whether the file system supports object identifiers.

**PARSE\_POINTS (Boolean)**

A true/false indication of whether the file system supports parse points.

**VOLUME\_QUOTAS (Boolean)**

A true/false indication of whether the file system supports volume quotas.

**REMOTE\_STORAGE (Boolean)**

A true/false indication of whether the file system supports remote storage.

**Example:**

```
$ freeclusters=f$getdviex("c:","FREECLUSTERS")
```

```
$ show symbol freeclusters
```

```
FREECLUSTERS = 438   Hex = 000001B6
```

```
$ declare symbol int64 freebytes
```

```
$ freebytes=f$getdviex ("c:","FREEBYTES")
```

## 18.26 F\$GETHOSTBYADDR Lexical

This lexical returns a TCP/IP host name.

**Format:**

**F\$GETHOSTBYADDR(ip-address)**

**Return Value:**

A character string representing the machine's TCP/IP host.

**Arguments:**

**ip-address**

a character string representing the machine's TCP/IP address.

**Example:**

```
$ host_name=f$gethostbyaddr("209.1.1.100")
$ sho sym host_name
HOST_NAME = "NS.advsyscon.com"
```

This example retrieves the IP hostname of node 209.1.1.100.

## 18.27 F\$GETHOSTBYNAME Lexical

This lexical returns a TCP/IP address.

**Format:**

**F\$GETHOSTBYNAME (host-name)**

**Return Value:**

A character string representing the machine's TCP/IP address. Please note that if this machine has multiple NIC's and IP addresses, the routine cannot determine which IP address will be passed back.

**Arguments:**

**host-name**

a character string representing the machine's TCP/IP host name.

**Example:**

```
$ ip_addr=f$gethostbyname("ns.advsyscon.com")
$ sho sym ip_addr
IP_ADDR = "209.1.1.100"
```

This example retrieves the IP address given a valid host name.

## 18.28 F\$GETJPI Lexical

This lexical returns detailed process information.

### Format:

**F\$GETJPI ([pid],[image-name], item, [machine])**

### Return Value:

A character string representing the information desired.

### Arguments:

#### pid

(Optional) The process id of the desired process. PID or image-name must be specified. If pid and image-name are specified, pid is used.

#### image-name

(Optional) The image-name of the desired process. Please note that the first process encountered is selected in the event multiple processes are running the same image. . PID or image-name must be specified. If pid and image-name are specified, pid is used.

#### item

One of the following quoted keywords:

##### PROCESSID (Integer)

The process' unique identification

##### HANDLES (Integer)

Number of handles used by the process

##### THREADS (Integer)

Number of threads used by the process

##### WORKINGSET (Integer)

The process' current working set (in bytes)

##### PAGEFAULTS (Integer)

Number of page faults incurred by the process

##### PRIORITY (Integer)

The current execution priority level of the process

##### VIRTUALBYTES (Integer)

The amount of virtual memory (in bytes) used by the process

##### PROCESS (String)

The process' image name

##### CPUTIME (Integer)

The amount of cputime used by the process

#### machine

(Optional) A valid machine name representing the above identified process. **Note: At the present time, this argument is ignored.**

### Example:

```
$ handles=f$getjpi($PID,,"handles",)
$ sho sym handles
HANDLES = 58   Hex = 0000003A
```

This example retrieves the number of handles in use by the current process. Note that the built-in symbol \$PID contains the current process id.

## 18.29 F\$GETLASTERROR Lexical

This lexical returns the last error value associated with a DECLARE'd FUNCTION invocation.

**Format:**

**F\$GETLASTERROR()**

**Return Value:**

An integer value to be interpreted as an error or function status code.

**Argument:**

**None.**

**Example:**

```
$ A=SetConsoleTitleA("Sample")  
$ write $stdout 'f$getlasterror()
```

This example shows a declared function (the Win32 API routine SetConsoleTitleA) being invoked. If that routine had failed, the lexical F\$GETLASTERROR would be used to determine the actual error causing the routine failure.

## 18.30 F\$GETSYI Lexical

This lexical returns detailed system information.

**Format:**

**F\$GETSYI (item, [machine])**

**Return Value:**

A character string representing the desired information.

**Arguments:**

**item**

One of the following quoted keywords:

**MAJOR\_VERSION (Integer)**

The major version number of the Operating System

**MINOR\_VERSION (Integer)**

The minor version number of the Operating System

**BUILD\_NUMBER (Integer)**

The O/S build number

**PAGE\_SIZE (Integer)**

The page size used by this machine

**PROCESSOR\_COUNT (Integer)**

The number of CPU processors installed

**GRANULARITY (Integer)**

The minimum amount of memory (in bytes) the system will allocate.

**ARCHITECTURE (String)**

The architecture of the machine. For example, INTEL, ALPHA, MIPS, etc.

**PROCESSOR\_TYPE (String)**

The type of processor model for this machine. For example, 486, 386, Pentium, 21064, R4000.

**PLATFORM (String)**

The operating system for this machine. For example, "Windows NT", "Win32 for Windows 95", etc.

**UP\_TIME (String)**

A character string representing the days and time that the machine and system have been up.

**VERSION (String)**

The complete version number of the Operating System

**MACHINENAME (String)**

The valid machine name for this system

**MACHINE\_TYPE (String)**

One of more strings that represent the type of machine designation. See F\$ENUMMACHINE for more details.

**machine-name**

(Optional) A character string representing the machine name that is lexical is to retrieve the desired information for.

**Note: At the present time, this argument is ignored.**

**Example:**

```
$ uptime=f$getsyi("up_time")
$ sho sym uptime
```

UPTIME = "14 04:13:56.999"

This command retrieves the system "up\_time" (total time the system has been up since the last reboot).

## 18.31 F\$GETVARIABLE Lexical

This lexical returns the value associated with an environmental variable. This lexical is particularly suited for retrieving variables set by web server/browser programs.

**Format:**

**F\$GETVARIABLE** (“variable”)

**Return Value:**

A character string that represents the value of the specified environment variable.

**Argument:**

**variable**

This argument represents any legal environment variable. For example, SYSTEMROOT, COMPUTERNAME, COMSPEC, HOMEDRIVE, PATH etc. may be specified.

**Example:**

```
$ HOME=F$GETVARIABLE("HOMEDRIVE")
$ SHOW SYMBOL HOME
HOME = "d:"
```

This example gets the value of the environment variable HOMEDRIVE and places that value in the symbol HOME.

```
$ DLL_HNDL=F$LOADLIBRARY(" 'F$GETVAR("SYSTEMROOT")' -
  \SYSTEM32\NETAPI32.DLL")
```

This example shows an embedded use of F\$GETVAR (note that the lexical as any other can be abbreviated to uniqueness). The result of SYSTEMROOT (n.b. C:\WINNT) is merged into the rest of the path for the DLL.

## 18.32 F\$GROUPINFO Lexical

This lexical returns specific information concerning the specified group. **(Windows NT Only)**

### Format:

**F\$GROUPINFO (group, [domain], [machine], item-code, [item-code-arg])**

### Return Value:

Depending on the item code specified, a string, boolean string or integer value will be returned.

### Argument:

#### group

A character string that represents the group whose information is being requested.

#### domain

(Optional) A character string that represents a domain name in which the specified user name exists.

#### machine

(Optional) A character string that represents a local machine name in which the specified user name exists.

#### item-code

One of the following item code keywords:

<b>COMMENT</b>	(String) Administrator's comments
<b>EXISTS</b>	(Boolean) Account Exists? This is the only item code that will not cause an error exception if an invalid or unknown machine/user is specified.
<b>GLOBAL</b>	(Boolean) Is group global? TRUE=Global
<b>IFMEMBER</b>	(Boolean) Determine if group contains a user, appearing in <i>item-code-arg</i> .
<b>MEMBERS</b>	(String) One or more users who are members of the named group. Multiple entries are separated by commas.
<b>SID</b>	(String) A textual SID representing the group is returned.

#### Item-code-arg

(Optional) A string argument that is used by the IFMEMBER item code to complete item code determination. When IFMEMBER is specified, then one or more users can be specified. The item code will determine and return TRUE if the group contains at least one user specified.

### Description:

This lexical is used to retrieve various characteristics of the named group account. If both domain-name and machine-name are omitted, the current login domain is used. Use of the EXISTS item-code helps to avoid handling error exceptions when the group specified is unknown. Once the group is confirmed to exist, the other item codes will most likely return data.

### Examples:

```
$ @groupinfo "domain guests" "" "" "guest"
$ a=f$groupinfo ("domain guests", "", "", "EXISTS")
$ sho sym a
A = "TRUE"
$ a=f$groupinfo ("domain guests", "", "", "COMMENT")
$ sho sym a
A = "All domain guests"
$ a=f$groupinfo ("domain guests", "", "", "MEMBERS")
```

```
$ sho sym a
A = "Guest,Pilot,Vendor,Aphrodite"
$ a=f$groupinfo ("domain guests", "", "", "IFMEMBER", "guest")
$ sho sym a
A = "TRUE"
$ exit
```

This example retrieves each item from the global group "Domain Guests". Since domain and machine are omitted, the domain is used.

## 18.33 F\$INTEGER Lexical

This lexical returns the integer equivalent of the result of the specified expression.

**Format:**

**F\$INTEGER (expression)**

**Return Value:**

An integer value equivalent to the specified expression.

**Argument:**

**expression**

Specifies the expression to be evaluated. Specify either an integer or a character string expression.

If you specify an integer expression, the lexical evaluates the expression and returns the result. If you specify a string expression, the lexical evaluates the expression, converts the resulting string to an integer, and returns the result. If the resulting string contains characters that form a valid integer, then F\$INTEGER returns the integer value. If the string contains characters that do not form a valid integer, an integer result of 1 is returned if the string begins with T, t, Y or y. An integer result of 0 is returned if the string begins with any other character.

**Example:**

```
$ A = F$INTEGER("101")
$ SHOW SYMBOL A
$ A = 101
```

This example shows how the string "101" is converted to an integer value.

**See also:**

[F\\$STRING](#)

## 18.34 F\$LENGTH Lexical

The F\$LENGTH lexical returns the length of the specified character string.

**Format:**

**F\$LENGTH (string)**

**Return Value:**

An integer value for the length of the string.

**Argument:**

**string**

Specifies the character string whose length is being determined.

**Example:**

```
$ A="ABCDEFGHI"  
$ WRITE $STDOUT F$LENGTH(A)  
9
```

This code sequence shows how you can obtain the length of a string symbol.

## 18.35 F\$LOADLIBRARY Lexical

The F\$LOADLIBRARY lexical allows DLL's to be dynamically loaded for local or global access by a command procedure. This lexical is required for users who wish to invoke user-defined functions via the DECLARE FUNCTION statement.

### Format:

**F\$LOADLIBRARY (file-spec)**

### Return Value:

A handle that corresponds to the loaded DLL is returned as an integer value. The value returned may be checked for a true or false indication. Please note that if the symbol used to hold the handle is local, then the library will be implicitly freed on command procedure exit. If the symbol is global, then only an explicit F\$FREELIBRARY will free the loaded DLL (all DLLs are freed on main command procedure exit).

### Argument:

**file-spec**

The file-specification (including path) of the DLL to be checked.

### Example:

```
$ dll_handle=F$LOADLIBRARY ("c:\mydll.dll")
```

This example loads the MYDLL library. The handle symbol DLL\_HANDLE would be used in subsequent library calls.

### See also:

[F\\$CHECKLIBRARY](#) [F\\$FREELIBRARY](#)

## 18.36 F\$LOCALTIMEZONE Lexical

The F\$LOCALTIMEZONE lexical returns the requested time-period name of the local time zone currently set on the computer.

**Format:**

**F\$LOCALTIMEZONE (time-period)**

**Return Value:**

A character string containing the requested information.

**Arguments:**

**time-period**

A character string containing the requested time zone period. It can be one of the following: *CURRENT* to return the current time period name; *STANDARD* to return the standard time period name; *DAYLIGHT* to return the daylight time period name. If omitted, the current time period name will be returned.

**Example:**

```
$ WRITE $STDOUT F$LOCALTIMEZONE()
```

```
Eastern Standard Time
```

This example displays the current time zone period for the eastern United States.

## 18.37 F\$LOCATE Lexical

The F\$LOCATE lexical locates a specified portion of a character string and returns the offset of the first character. (The first character in a string is always offset position 0 from the beginning of the string.) If the substring is not found, F\$LOCATE returns the length (the offset of the last character in the string plus one) of the searched string.

### Format:

**F\$LOCATE (substring, string)**

### Return Value:

An integer value representing the offset of the substring argument.

### Arguments:

#### substring

Specifies the character string to be located within the string specified by the **string** argument.

#### string

Specifies the character string to be edited by F\$LOCATE.

### Example:

```
$ INQUIRE TIME "Enter time"  
$ IF F$LOCATE(":", TIME) .EQ. F$LENGTH(TIME) THEN -  
  GOTO NO_COLON
```

This portion of a command procedure compares the results of the F\$LOCATE and F\$LENGTH functions to see if they are equal. This technique is commonly used to determine whether a character or substring is contained in a string. In the example, the INQUIRE command prompts for a time value and assigns the user-supplied time to the symbol TIME. The IF command checks for the presence of a colon (:) in the string entered in response to the prompt. If the value returned by the F\$LOCATE function equals the value returned by the F\$LENGTH function, the colon is not present. You use the .EQ. operator (rather than .EQS.) because the F\$LOCATE and F\$LENGTH functions return integer values. Note that quotation marks are used around the substring argument, the colon, because it is a string literal. However, the symbol TIME does not require quotation marks because it is automatically evaluated as a string expression.

## 18.38 F\$LOOKUPREGISTRY Lexical

This lexical allows the lookup of a key (and optionally with value-name). If the value-name is specified, then the value and its datatype may be optionally returned.

### Format:

**F\$LOOKUPREGISTRY (hive, key, [reg-datatype, value])**

### Return Value:

This lexical returns a "TRUE" or "FALSE" string indicating whether the desired function was successful or failed.

### Arguments:

#### hive

A quoted string containing one Microsoft valid hive specification (for example, HK\_LOCAL\_MACHINE or HKLM). Please see the "Registry Lexicals" discussion for the table of hives and their abbreviations. To specify a remote machine, use UNC format and specify the \\machine\hive format.

#### key

A quoted string defining both the actual key and any associated value name using the syntax: key\sub-key\sub-sub-key...\value-name

If a NONAME value-name is to be associated with the key then two slashes are used.

**reg-datatype**

(Optional) If specified, the datatype of the associated value is returned as a character string. The string consisting of one of the following registry datatypes: REG\_DWORD, REG\_SZ, REG\_MULTI\_SZ, REG\_BINARY, REG\_EXPAND\_SZ

**value**

(Optional) The actual value associated with the key and value-name is returned if this parameter is passed. The symbol used must be of the appropriate datatype if the symbol is previously declared. If the symbol is implicit and has never been defined, then XLNT will define the symbol of appropriate datatype.

**Examples:**

```
$ A=F$LOOKUPREGISTRY ("HKEY_CURRENT_USER", -  
    "SOFTWARE\ASCIMYKEY\VALUE-NAME", -  
    returned_type, returned_value)
```

This example looks up the value of SOFTWARE\ASCIMYKEY\VALUE-NAME and returns both the value's datatype and the value itself in the corresponding arguments; returned\_type and returned\_value.

See also:

**F\$ADDREGISTRY F\$CHANGeregistry F\$DELETERegistry**

## 18.39 F\$MESSAGE Lexical

The F\$MESSAGE lexical returns a character string with the message text associated with a message code.

**Format:**

**F\$MESSAGE(message-value, [dll-handle])**

**Return Value:**

A character string which represents the textual equivalent of a message code.

**Arguments:**

**message-value**

An integer value representing a valid message code.

**dll-handle**

A handle passed back from a successful F\$LOADLIBRARY invocation. This optional argument is useful for accessing message DLLs to be used for message display.

**Example:**

```
$ WRITE $STDOUT F$MESSAGE(4)
```

The system cannot open the file.

## 18.40 F\$MODE Lexical

The F\$MODE lexical returns a character string showing the mode in which the process is executing.

**Format:**

**F\$MODE ( )**

**Return Value:**

The character string **INTERACTIVE** for interactive processes, **BATCH** if the process is running as an XLNT batch stream, **CGI** if the process was started via a Web Browser, **ACTIVEX** if the script was started through the ActiveX facility, **WSH** if the script was started through the Windows Scripting Host or **OTHER** for any other technique.

**Description:**

This very important lexical returns a character string that shows the mode that the command procedure is currently running in. This lexical is useful when a command procedure must execute different commands depending on the mode it has been started in. As the example below shows you might want to request confirmation on replacing a file using the COPY command whereas if the same command is executed non-interactively, there would be no one to ask! The same is true if the script originates from the ActiveX Scripting environment or more specifically WSH.

**Example:**

```
$ IF F$MODE() .EQS. "INTERACTIVE" THEN -  
    COPY == COPY/ASK/NOREPLACE
```

This simple example shows a possible line in a SYLOGIN or USRLOGIN script. The script could determine the mode it is being executed in (interactive, batch, etc) and take the appropriate action. In this example, the user is changing the COPY verb through the use of a symbol of the same name so that the COPY default of replacing an existing file is changed to ask for confirmation instead. In a Batch job, you wouldn't want that action since no one will be able to answer a COPY query. Please note that the use of this statement is critically important in CGI development since any non-standard Writes to STDOUT would abort Browser/Server processing.

## 18.41 F\$MSGBOX Lexical

The F\$MSGBOX creates and displays a message box. The message box contains a user-supplied message and title and any combination of predefined icons and push buttons.

### Format:

**F\$MSGBOX (message, type [,title] [,icon] [,defaultbutton] [,options] [,timeout])**

### Return Value:

A character string indicating which button was pressed. One of the following strings will be returned:

- **ABORT**            Abort button was pressed
- **CANCEL**          Cancel button was pressed
- **IGNORE**          Ignore button was pressed
- **NO**                No button was pressed
- **OK**                OK button was pressed
- **RETRY**            Retry button was pressed
- **YES**                Yes button was pressed

### Arguments:

#### message

Specifies a character string containing the message to be displayed.

#### type

Specifies a character string indicating the type of message box to be generated.. It must be specified as one of the following string expressions:

- **ABORTRETRYIGNORE** - Message box will contain three buttons: Abort, Retry, and Ignore
- **OK** - Message box will contain one button: OK
- **OKCANCEL** - Message box will contain two buttons: OK and Cancel
- **RETRYCANCEL** - Message box will contain two buttons: Retry and Cancel
- **YESNO** - Message box will contain two buttons: Yes and No
- **YESNOCANCEL** - Message box will contain three buttons: Yes, No, and Cancel

#### title

Specifies a character string containing the title of the Message Box. This argument is optional. By default, the title will be "Error".

#### icon

Specifies the icon to display in the message box. This argument is optional. If specified, it must be one of the following character strings:

- **EXCLAMATION** - an exclamation-point icon appears in the message box.
- **WARNING** - an exclamation-point icon appears in the message box.
- **INFORMATION** - an icon consisting of the lowercase letter *I* appears in the box.
- **ASTERISK** - an icon consisting of the lowercase letter *I* appears in the box.
- **QUESTION** - a question-mark icon appears in the message box.
- **STOP** - a stop-sign icon appears in the message box.
- **ERROR** - an error icon appears in the message box.
- **HAND** - a hand icon appears in the message box.

**defaultbutton**

Specifies the default button in the message box. This argument is optional. If specified, it must be one of the following character strings:

- **BUTTON1** - the first button is the default button.
- **BUTTON2** - the second button is the default button.
- **BUTTON3** - the third button is the default button.

**options**

Specifies message box options. This argument is optional. If specified, any of the following character strings may be supplied, separated by commas (i.e. "RIGHT,TOPMOST"):

- **RIGHT** - the text is right justified.
- **RTLREADING** - message and caption text are displayed using right-to-left reading order on Hebrew and Arabic systems.
- **TOPMOST** - the message box will remain on top.

## timeout

Specifies the period of time the script will wait before timing out the message box. This argument is optional. If specified, a non-zero positive integer value can be specified and will be used as the maximum period, in seconds, that XLNT will wait for message box input. If the wait period is expired, XLNT will continue script execution by causing the *defaultbutton* to be returned to the script. If *timeout* is specified, then the *defaultbutton* argument must also be specified.

### Example:

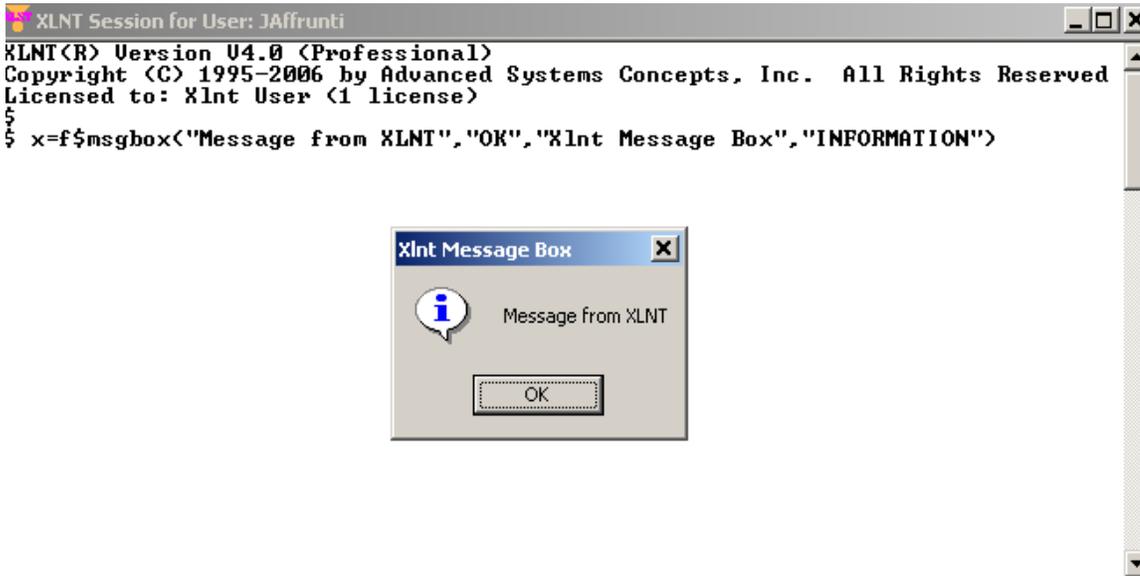


Figure 25. F\$MSGBOX Example

## 18.42 F\$PARSE Lexical

The F\$PARSE lexical returns a character string representing the information requested for the desired file specification.

### Format:

**F\$PARSE (file-spec, [default file-spec], [item], [option])**

### Arguments:

#### file-spec

Specifies the file specification to be parsed.

#### default file-spec

Specifies a default file specification to be combined with the file specification. Note: If a standalone directory specification is used, please ensure that the trailing “\” is also specified (i.e. c:\asci\ NOT c:\asci)

#### item

Specifies the desired parsing. By default, the entire file specification is returned as a character string. Valid item strings are:

**NODE** - Parse for NODE (machine) information

**DEVICE** - Parse for DEVICE information

**DIRECTORY** - Parse for DIRECTORY information

**NAME** - Parse for FILENAME information

**TYPE** - Parse for FILETYPE information

#### option

Specifies option information to the **F\$PARSE** command. By default, **F\$PARSE** performs a **SYNTAX\_ONLY** check of the file specification. The option **EXISTS**, causes **F\$PARSE** to check for the existence of the file before parsing its contents.

### Example:

```
$ FILE = F$PARSE ("XLNT.DAT",,"NAME")
$ SPEC = F$PARSE ("XLNT.HLP","C:\XLNT",,"EXISTS")
```

This example parses the file XLNT.DAT and returns the *filename* portion into the symbol FILE. The second example shows how the default file specification can be used. In this case, the primary file specification doesn't contain a device or directory spec, so those portions are merged from the default specification. The EXISTS keyword causes XLNT to also check and verify whether the file actually exists (as opposed to SYNTAX only usage).

### See also:

[F\\$SEARCH](#)

## 18.43 F\$PERMISSIONS Lexical

This lexical will retrieve owner, group and security permissions for File, Registry, Service and Printer objects. The information retrieved is in a string format perfect either for subsequent display or as keywords for the SET PERMISSIONS command.

### Format:

**F\$PERMISSIONS (object-type, object, item-code [,context])**

### Return-Value:

If successful, a string representing the desired item is returned. A null string indicates the end of a context search.

### Arguments:

#### object-type

This string argument must be one of the following: FILE, SHARE, REGKEY, SERVICE or PRINTER.

#### object

This string argument contains the actual object on which security information will be retrieved. The object must be compatible with the object-type specified. For example, a file must be specified for a FILE object.

#### item-code

This string argument must be one of the following:

**OWNER** - requests the owner of the object.

**PERMISSION** - requests security permissions of the object. Each successive call with the context argument set will pass back a string containing: Account:Raw

**ACCOUNT** - requests account security information for the object. Each successive call with the context argument set will pass back a string containing just the account name the permission is applicable towards.

**FORMATTED** - requested formatted security information for the object in a style suitable for display. Each successive call with the context argument set will pass back a string containing just the formatted permission.

**RAW** - requested raw security information for the object in a style suitable for the SET PERMISSIONS command or for advanced Power Users. Each successive call with the context argument set will pass back a string containing just the formatted permission.

**SIDACCOUNT** - similar to ACCOUNT above except that textual SIDs are returned instead of account names.

**SIDOWNER** - similar to OWNER above except that a textual SID is returned instead of the owner's account name.

**SIDPERMISSION** – similar to PERMISSION above except that textual SIDs are returned instead of account names.

#### context

This optional integer symbol argument is used to iteratively retrieve an object's permission when the item code PERMISSION is specified and this context argument is specified. You do not have to declare the symbol used for context information (although if the symbol is already typed it must be integer and must be zero to initiate a new search). The context symbol will be updated upon each successful call to the lexical. You must not alter the context symbol.

### Examples:

```
$ a=f$permissions ("file", "D:\WTKILFIL.EXE", "owner")
$ show sym a
```

```
A = "Administrators"
$ for (, )
$ a=f$permissions ("file", "D:\WTKILFIL.EXE", "permission", context)
$ if a .eqs. "" then leave
$ show sym a
A = "Everyone: All: All"
$ endfor
$ a=f$permissions ("file", "D:\WTKILFIL.EXE", "permission", context)
$ if a .eqs. "" then leave
$ exit
```

The example above uses a file object named "D:\WTKILFIL.EXE and retrieves several different types of security information.

## 18.44 F\$PID Lexical

This lexical will pass back either the caller's Process Identification (PID) or a PID matching the Image Name of the process. F\$PID can also sequentially return all PIDs on the local or remote system.

### Format:

**F\$PID ([image-name],[context],[machine-name])**

### Return-Value:

If successful, the PID (integer longword) is passed back.

### Arguments:

#### **image-name**

This optional string argument causes the lexical to selectively search for a process whose image-name matches the argument specified. If this argument is omitted, no special filtering for specific processes are performed.

#### **context**

This optional integer symbol argument is used to iteratively scan through one or more processes on a WinNT system. The symbol which may be undefined or, if defined, is an integer longword equal to zero, provides the search context for F\$PID to remember the next search point as it is invoked multiple times. If this argument is omitted and image-name is omitted then the current process (caller) PID is returned. If this argument is omitted and the image-name argument is specified, then the PID matching the image-name (if any) is returned.

#### **machine-name**

This optional string argument indicates the machine-name to search for retrieving PIDs based on the search criteria either specified or omitted through the image-name argument.

### Description:

F\$PID can be used to sequentially retrieve Process ID's on a local or remote machine or retrieve a specific PID based on image-name.

### Examples:

**\$ A = F\$PID()**

This simple example returns the PID of the caller (in other words the PID of the process invoking the lexical).

**\$ A = F\$PID("Winlogon")**

This example returns the PID of the "Winlogon" process.

**\$ ON ERROR GOTO DONE**

**\$ FOR (,,)**

**\$ A = F\$PID(,B)**

**\$ ENDFOR**

**\$DONE:**

**#! WE ARE DONE**

This example is meant to show how the lexical can be used to retrieve multiple PIDs in a sequential manner. Note the use of the ON statement. This allows the procedure to intercept and process the NOPROCESSINFO error indicating the end

of the sequential search.

## 18.45 F\$RANDOM Lexical

This lexical yields a random integer number.

**Format:**

**F\$RANDOM([range-max][,seed])**

**Return-Value:**

An integer value representing a randomly generated number.

**Arguments:**

**range-max**

An optional integer value which, if specified, limits the random number returned to *range-max* minus 1. By default, the largest unsigned integer value is used.

**Seed**

An optional integer value which, if specified, is used as the seed value when generating random numbers. The Seed argument is very useful when you want to generate the same set of random numbers for each possible experiment. If this argument is not specified, then a true set of random numbers are generated based on time of day.

**Example:**

```
$ A=F$RANDOM()  
$ WRITE $STDOUT A  
41
```

This simple example yields a random number whose value is assigned to integer symbol A.

## 18.46 F\$READEVENT Lexical

This lexical allows you to read the Windows NT Event Log.

### Format:

**F\$READEVENT (EventLog, [Source], [ReadDir], [Offset] , [Return-Symbol] , [Item])**

### Return-Value:

A boolean value indicating whether the lexical succeeded or failed.

### Arguments:

#### EventLog

A string indicating which event log is to be read (for example, SYSTEM, APPLICATION, SECURITY). For remote access, a UNC style specification may be added. For example, \\server\system indicates that the SYSTEM event log on machine SERVER be accessed.

#### Source

A string indicating the Event Source name. This name is registered with the event log facility. An example would be "XLNTNET". If specified, this argument allows read selection for the specified Source in the direction specified (or by default) indicated by *ReadDir*. If omitted, source selection is not performed and all events may be selected and read.

#### ReadDir

A string indicating the direction the event log is to be read. Valid directions are: FORWARDS and BACKWARDS. If ReadDir is omitted, BACKWARDS is assumed. BACKWARDS is actually sequential reading of the event log from the newest entry to the oldest entry.

#### Offset

This optional argument provides a method for selecting a specific event or to receive the offset of an event for later viewing. If specified, Offset must be an integer symbol name. If the symbol passed is undefined or a zero then the function will perform the desired operation and pass back a value that represents the location of the specific event in the Windows NT Event log. If the symbol passed is not equal to zero, then a SEEK type operation will be performed and the specific event at that location will be read. If you are not interested in re-reading specific events, you may want to omit this argument.

#### Return-Symbol

This optional argument receives the data requested from *Item*. The symbol specified may be either undefined or if defined, the datatype must match that of the specific item requested.

#### item

This argument represents one of the following string item codes:

<b>CATEGORY</b>	returns category type of message as integer
<b>COMPUTER</b>	returns computer name string
<b>DATETIME</b>	returns Date/Time of event as string
<b>EVENTID</b>	returns Event ID as integer
<b>SOURCE</b>	returns EventLog as string
<b>TYPE</b>	returns Message Status type as string
<b>USER</b>	returns Username of event as string

### Example:

```
$ READEV = -F$READEVENT("APPLI CATI ON", , , OFFSET, LASTEVENT, "DATETI ME")
$ OFFSET=0
$ READEV = F$READEVENT("APPLI CATI ON", , , OFFSET, SOURCE, "SOURCE")
$ SHOW SYMBOL LASTEVENT
LASTEVENT = "7/7/97 11:37:59 AM"
$ SHOW SYMBOL SOURCE
SOURCE = "Perfl i b"
```

This simple example reads the event log from the beginning and obtains two pieces of information concerning that entry. The information returned is the source of the event and the date and time the event was logged. Note the user of "OFFSET=0". This is necessary since each call to READEVENT will sequentially obtain the next event. This requires saving the current offset when you want to retrieve multiple items of information from a single event.

## 18.47 F\$REPLACE Lexical

This lexical searches a string for a substring and then replaces that substring.

### Format:

**F\$REPLACE (search-string,[replace-string],string[,item])**

### Return Value:

A character string is returned after all replacements (if any) have been applied.

### Arguments:

#### search-string

A character string representing the data to be searched for within the **string** argument.

#### replace-string

A character string representing the data to be used to replace the **search-string** found within the **string** argument. A null string or omitting this argument indicates that the search-string should be removed within the string.

#### string

A character string representing the data that search-string and replace-string will work on. The string argument itself is never modified. The resultant string is passed back as the return value for F\$REPLACE.

#### item

A character string that changes the type of search and replace performed. Item code(s) may be one or more as listed below (item codes are enclosed within a quoted string and multiple codes are separated by commas). This argument may be omitted, the default is "CASE,ALL".

<b>CASE</b>	Case Sensitive (default)
<b>NOCASE</b>	Case Insensitive
<b>FIRST</b>	Search/Replace (first occurrence)
<b>ALL</b>	Search/Replace (all occurrences) (default)

### Example:

```
$ A="XXXX IS GREAT"  
$ B=F$REPLACE("XXXX","XLNT",A)
```

This example shows a very simple replacement. The symbol A is a string symbol of value "XXXX IS GREAT". The F\$REPLACE lexical searches symbol A for XXXX, when the search string is found, the replacement string of XLNT is replaced in the resultant string symbol. This find and replace operation continues until the search-string is no longer present.

```
$ A="XXXX IS GREAT"  
$ B=F$REPLACE("xxxx","XLNT",A,"NOCASE")
```

This example shows a variation of the above example. By specifying the NOCASE item, the search performed is caseless.

## 18.48 F\$REPORTEVENT Lexical

This lexical is used to write or report an event to the Windows NT Event Log.

### Format:

**F\$REPORTEVENT (EventLog, [status-id], [message-type], [additional-text])**

### Return-Value:

A boolean value indicating whether the lexical succeeded or failed.

### Arguments:

#### EventLog

A string indicating which event log source is to be written. This is a sub-key of SYSTEM, APPLICATION, SECURITY. For remote access, a UNC style specification may be added. For example, \\server\system indicates that the SYSTEM event log on machine SERVER be accessed.

#### Status-Id

An integer value indicating the message code identification to report to the event log.

#### Message-Type

A string indicating the severity of the status-id (or event). The string value may be one of the following: INFORMATION, WARNING or ERROR. If the severity cannot be determined it will default to ERROR.

#### Additional-Text

An optional string argument that will be associated with the *status-id* being reported.

**Note:** To report an event to the *eventlog* successfully you must know, at a minimum, both the *source* and the *message ID* of the event that you would like to post.

**For Advanced Users:** If you have your own message DLL that you would like to post messages from using F\$REPORTEVENT, create your own source Registry key (i.e. XLNT) in:

**HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\EventLog\Application**

Create two values under that new key. The first value would be a REG\_EXPAND\_SZ type with a name of EventMessageFile and a value of the full path to your message DLL. The second value under the new key is a REG\_DWORD called TypesSupported and give this key a Hex value of 7. Now using the new source that was just created called XLNT, an event could be reported with:

**\$ x = F\$REPORTEVENT("XLNT",messageid#)**

where *messageid#* is a valid message id in the XLNT message DLL. Some examples of useful *sources* are for script, CGI, or DOS application debugging.

### Example:

**\$ x = F\$REPORTEVENT("XLNTNET", \$STATUS, "ERROR")**

Will post an event using the registered source of XLNTNET in the event log with an ID of \$STATUS and a source type of "ERROR".

## 18.49 F\$SEARCH Lexical

The F\$SEARCH lexical returns a character string representing the file specification to search.

**Format:**

**F\$SEARCH (file-spec, [context-symbol])**

**Arguments:**

**file-spec**

Specifies the file specification to be searched. You may specify any legal Windows NT specification including use of any wildcard characters as noted in the section "Files and Directories".

**context-symbol**

Specifies a symbol name which stores a context pointer used for wildcard searching. To reset the context for a given search delete the symbol prior to use (DELETE/SYMBOL). This symbol must not be explicitly declared. If context is omitted for a wildcard operation only the first file found (if any) will be returned.

**Example:**

```
$ FILE = F$SEARCH ("XLNT.DAT")
$ WILDFILE = F$SEARCH ("*.XCP",NEXT_FILE)
```

The first example simply searches for the file XLNT.DAT (in the current directory). If the file is found, a non-null string will be passed back as the value. The second example shows a major use of F\$SEARCH for searching a file specification that contains a wildcard symbol. In this case, the context argument NEXT\_FILE is used. For each successful search, a file specification (non-null string) will be passed back to WILDFILE. When the search is completed, a null string is returned.

**See also:**

[F\\$PARSE](#)

## 18.50 F\$SEQARRAY Lexical

This lexical is used to sequentially scan through the key index of an associative array.

**Format:**

**F\$SEQARRAY (array-symbol, context)**

**Return Value:**

A character string containing the next sequential key index or a null string indicating the end of the array.

**Arguments:**

**array-symbol**

A symbol that represents the entire associative array (not just an element of the array).

**context**

An integer symbol that is initially set to zero OR an undefined symbol that will become an integer symbol initialized to zero. On each successful call, this symbol will be updated by the context necessary to read the next ordered key index in the array. This symbol should not be updated or modified by the script in any manner after it has been initialized.

**Description:**

This lexical is used to sequentially obtain an ordered list of the key index for an associative array. Each call after the context argument has been initialized with result in either the next key index string or a null string indicating that the end of the array has been reached. This technique allows script writers to really exploit associative arrays since the elements may be randomly and sequentially accessed.

**Example:**

```
$ ADDRESS{"irv"}="1 Sycamore Lane"
$ ADDRESS{"abe"}="214 Main Street"
$ ADDRESS{"don"}="659 Townsend Ave"
$ FOR (,)
$     KEY=F$SEQARRAY(ADDRESS,context)
$     IF KEY .EQS. "" THEN LEAVE
$     WRITE $STDOUT "Address for 'key' is "ADDRESS{"key"} "
$ ENDFOR
$ EXIT
```

```
Address for abe is 214 Main Street
Address for don is 659 Townsend Ave
Address for irv is 1 Sycamore Lane
```

This example depicts an associative arrays with three key index values of "irv", "abe" and "don" stored in that order. The F\$SEQARRAY lexical allows direct sequential and ordered access to the key index values.

## 18.51 F\$SERVICE\_STATUS Lexical

This lexical is used to determine the current status of a service. This lexical may be used on Windows NT only.

### Format:

**F\$SERVICE\_STATUS (service-name, item [,machine-name])**

### Return Value:

A character string containing the requested status.

### Arguments:

#### **service-name**

Specifies a string containing the name under which the desired service is installed.

#### **item**

Specifies a string containing the desired status item. It must be specified as one of the following string expressions:

#### **INSTALLED**

indicates whether the specified service is installed. If it is, a value of "TRUE" will be returned. If the service is not installed, a value of "FALSE" will be returned.

#### **TYPE**

returns a string indicating the specified service's type. The string will be one of the following:

OWN\_PROCESS - a Win32 service that runs in its own process;

SHARED\_PROCESS - a Win32 service that shares its process with other services;

DEVICE\_DRIVER - a Windows NT device driver;

FILE\_SYSTEM\_DRIVER - a Windows NT file system driver;

INTERACTIVE - a Win32 service process that can interact with the desktop.

#### **STATE**

returns a string indicating the current state of the service:

STOPPED - the service is not running;

START\_PENDING - the service is starting;

STOP\_PENDING - the service is stopping;

ACTIVE - the service is running;

CONTINUE\_PENDING - the service continue is pending;

PAUSE\_PENDING - the service pause is pending;

PAUSED - the service is paused.

**machine-name**

Specifies a string containing the name of the computer from which the service status will be retrieved. This argument is optional. By default, the status will be retrieved from the local computer.

**Example:**

```
$ A=F$SERVICE_STATUS ("XLNTNET","INSTALLED")
$ SHOW SYMBOL A
A = "TRUE"
```

This example shows how you can determine whether a specified service is installed. The return status is TRUE. If the return status was false then the INSTALL SERVICE command could have been used to install the service.

## 18.52 F\$STRING Lexical

This lexical returns a string that is equivalent to the specified expression.

**Format:**

**F\$STRING (expression)**

**Return Value:**

A character string equivalent to the specified expression.

**Argument:**

**expression**

The integer or string expression to be evaluated.

If you specify an integer expression, the lexical evaluates the expression, converts the resulting integer to a string and returns the result. If you specify a string expression, the lexical evaluates the expression and returns the result.

When converting an integer to a string, the lexical uses decimal representation and omits leading zeros. When converting a negative integer, the lexical places a minus sign at the beginning of the string representation of the integer.

**Example:**

```
$ COUNT = 100
```

```
$ A = F$STRING (COUNT+2)
```

This example shows how an integer symbol COUNT as part of an expression (COUNT+2) can be converted to a string value ("102").

**See also:**

[F\\$INTEGER](#)

## 18.53 F\$TIME Lexical

The F\$TIME lexical returns the current date and time in absolute time format.

**Format:**

**F\$TIME ( )**

**Return Value:**

The current date and time formatted as

*dd-mmm-yyyy hh:mm:ss.mms.*

**Arguments:**

None.

**Example:**

```
$ WRITE $STDOUT F$TIME()
```

```
04-Sep-1996 10:44:50.04
```

The current date and time are displayed in absolute time format with an abbreviated Day.

**See also:**

[F\\$CVTIME](#) [F\\$FORMATDATE](#) [F\\$FORMATTIME](#)

## 18.54 F\$TYPE Lexical

This lexical returns the datatype of a symbol.

**Format:**

**F\$TYPE(symbol)**

**Return Value:**

A string representing the datatype of the specified symbol is returned. The returned string will be one of the following: SBYTE, SWORD, INTEGER, SLONG, UBYTE, UWORD, WLONG, STRING, HANDLE, OBJECT or structure-name.

**Argument:**

**symbol**

Specifies the name of the symbol to be evaluated. The symbol must be defined.

**Note:**

When an array-symbol is presented to F\$TYPE the datatype of the array is interpreted and presented. When a structure-symbol is presented to F\$TYPE the name of the structure is returned. A structure is a form of datatype to XLNT.

**Example:**

```
$ A = 1
$ write $stdout f$type(A)
INTEGER
```

This example displays the type of a symbol.

## 18.55 F\$USERINFO Lexical

This lexical returns specific information concerning the specified user account.

**Format:**

**F\$USERINFO (user, [domain], [machine], item-code, [item-code-arg])**

**Return Value:**

Depending on the item code specified, a string, boolean string or integer value will be returned.

**Argument:**

**user**

A character string that represents the user account whose information is being requested.

**domain**

(Optional) A character string that represents a domain name in which the specified user name exists.

**machine**

(Optional) A character string that represents a local machine name in which the specified user name exists.

**item-code**

One of the following item code keywords:

<b>ACCTEXPIRES</b>	(String) The date/time when the account expires.
<b>ACTIVE</b>	(Boolean) Account Active?
<b>COMMENT</b>	(String) Administrator's comments
<b>COUNTRY-CODE</b>	(String) Country code
<b>EXISTS</b>	(Boolean) Account Exists?
<b>FULL-NAME</b>	(String) User's full name.
<b>GLOBAL-GROUPS</b>	(String) Zero or more global group names delimited by commas.
<b>HOME-DIR</b>	(String) Home directory
<b>HOME-DRIVE</b>	(String) Home drive letter if home directory is a UNC path.
<b>IFMEMBER</b>	(Boolean) Determine if user is a member of the groups which appear in <i>item-code-arg</i> .
<b>LOCAL-GROUPS</b>	(String) Zero or more local group names delimited by commas.
<b>LOCKED</b>	(Boolean) Is Account Locked Out?
<b>LOGOFF-LAST</b>	(String) Last date/time that user logged off machine.
<b>LOGON-ERROR-CNT</b>	(Integer) Number of failed logins.
<b>LOGON-HOURS-ALLOWED</b>	(String) Valid times that user may logon.
<b>LOGON-LAST</b>	(String) Last date/time that user logged into machine.
<b>LOGON-SCRIPT</b>	(String) User's logon script file specification.
<b>LOGON-SERVER</b>	(String) User's Logon Server.
<b>LOGON-SUCCESS-CNT</b>	(Integer) Number of successful logins.
<b>PRI-GLOBAL-GRP</b>	(String) Primary global group.

<b>PROFILE</b>	(String) User Profile Path
<b>PSWD-CHANGED</b>	(String) Date/time password was changed.
<b>PSWD-EXPIRES</b>	(String) Date/time password expires.
<b>PSWD-LAST-SET</b>	(String) Date/time password was last set.
<b>RAS-DIALPRIV</b>	(Boolean) Does User have dialin priv?
<b>RAS-CALLTYPE</b>	(String) RAS Callback Type: None, Caller or Admin
<b>RAS-CALLPHONE</b>	(String) RAS Callback Phone number (if Admin callback type).
<b>SID</b>	(String) A textual SID of the user account is returned.
<b>USER-CHG-PSWD</b>	(String) Date/Time User changed password
<b>USERNAME</b>	(String) Username
<b>WKSTA-ALLOWED</b>	(String) One or more workstations that user may specifically log into. The value "ALL" indicates no workstation restrictions.

#### Item-code-arg

(Optional) A string argument that is used by the IFMEMBER item code to complete item code determination. When IFMEMBER is specified, then one or more groups can be specified. The item code will determine and return TRUE if the user is a member of any one of those groups.

#### Examples:

```
$ a = f$userinfo ("p1","p2","p3","EXISTS")
$ sho sym a
$ a = f$userinfo ("p1","p2","p3","FULL-NAME")
$ sho sym a
$ a = f$userinfo ("p1","p2","p3","home-dir")
$ sho sym a
$ a = f$userinfo ("p1","p2","p3","pri-global-grp")
$ sho sym a
$ a = f$userinfo ("p1","p2","p3","global-groups")
$ sho sym a
$ exit
A = "TRUE"
A = "Guest"
A = "d: \guest"
A = "DOMAIN ADMINISTRATORS"
A = "Domain Users, Domain Admins, Domain Guests"
```

This example displays a procedure and its output. The symbol P1 represents a username, P2 represents the domain-name, P3 represents a machine-name. P2 and P3 can be omitted and if they are the default login domain server is used.

## 18.56 F\$UTCTIME Lexical

The F\$UTCTIME lexical returns the current Universal Time Coordinates (UTC) date and time in absolute time format.

**Format:**

**F\$UTCTIME ()**

**Return Value:**

The UTC date and time formatted as:

*dd-mmm-yyyy hh:mm:ss.mms*

**Arguments:**

None.

**Example:**

```
$ WRITE $STDOUT F$UTCTIME ()
```

```
10-Jan-2006 15:10:07.714
```

The UTC date and time are displayed in absolute time format.

## 18.57 F\$VERIFY Lexical

This lexical returns the current procedure verification level and also allows a new verification level to be set.

**Format:**

**F\$VERIFY([procedure-verification])**

**Return Value:**

An integer value is returned representing the current procedure verification level. A zero means no verification set, a one means that verification is set.

**Argument:**

**procedure-verification**

An integer value of zero or one. Zero disables procedure verification and one enables verification. Verification corresponds with XLNT's ability to display the procedure statements.

**Example:**

```
$ VER=F$VERIFY(1)
$ WRITE $STDOUT "TEST: Veri fication was 'ver' on entry."          TEST: Veri fication was 0 on
entry.                    $ VER=F$VERIFY(VER)
$ EXIT
```

This example enables verification mode and remembers the current setting on entry (in symbol VER). The setting on entry is then restored when the procedure exits.

# 19 Sample XLNT Procedures

The procedures that follow demonstrate some of the XLNT language statements and facilities.

Please note that these procedure's wrap due to formatting constraints of the manual. Therefore, only statements that begin with "\$" indicate the start of a line and any other data is just a continuation of the previous line. The procedures listed here are available for use and can be found on the product CD or via download from the ASCII web site. All procedures listed here are unsupported and provided in "AS IS" condition. For the complete terms of the use of each procedure please examine the procedure itself for more details.

## 19.1 Administrator's Helper Procedure

```
$!+
$! ADMMENU.XCP - View Important Domain/Machine Information
$! Script Version 1.0
$!
$! USER TYPE: Administrator
$!
$! SPECIAL RIGHTS REQUIRED: None or list of specific rights
$!
$! DESCRIPTION:
$! This procedure is designed to view all important information about a computer
$! and the domain it is attached to for security, administrative and information
$! Purposes
$!
$!-
$ echo = "write $stdout"
$ dtest = "N"
$ on error then goto ERROR_FATAL
$! MAIN MENU
$ MAINMENU:
$ cls
$ echo ""
$ echo ""
$ echo " ***** "
$ echo " *          MAIN MENU          * "
$ echo " ***** "
$ echo " * * * * * "
$ echo " * 1. View All Domain Users * "
$ echo " * 2. View All User Groups * "
$ echo " * 3. List All Shares on the Current Machine * "
$ echo " * 4. List All Open Network Files on Current Machine * "
$ echo " * 5. Show All Domain Info for Current User * "
$ echo " * 6. Show All Local Info for Current User * "
$ echo " * 7. Show All Domain Rights for Current User * "
$ echo " * 8. Show All Local Rights for Current User * "
$ echo " * 9. View All Machines and Shares on Network * "
$ echo " * 10. Exit XLNT ADMIN MENU * "
$ echo " ***** "
$ echo ""
$ echo ""
$ tselect = 0
$ inquire inselect "Selection"
$ tselect = f$integer(inselect)
$ if tselect .eq. 10 then goto EXIT_MENU
$ if tselect .lt. 1 then goto MAINMENU
$ if tselect .gt. 10 then goto MAINMENU
$ cls
$ goto SUB'inselect
$!
$! ***** SUBROUTINES FOR MAIN MENU *****
$SUB1:
$ SECURITY SHOW USER * /DOMAIN
$ inquire inselect "Press Enter to Return to MAIN MENU"
$ goto MAINMENU
```



\$ abort "Script: admmenu.xcp exiting." 1

## 19.2 View Dormant Accounts Procedure

```
#!/+
#!/ DORMANT.XCP - Finds dormant accounts on an NT domain
#!/ Script Version 2.0
#!/
#!/ USER TYPE: Administrator
#!/
#!/ SPECIAL RIGHTS REQUIRED: None
#!/
#!/ DESCRIPTION:
#!/ This script will use XLNT V2.0's F$ENUMUSER and F$USERINFO to
#!/ figure out the last login date of each user. Then it will determine
#!/ is the user has never logged in, logged in today, or logged in X number
#!/ of days ago. This can be used to disable accounts based on date, or delete
#!/ unneeded accounts.
#!/
#!/ REQUIREMENTS:
#!/ - XLNT V2.0
#!/ - Windows NT Workstation or Server
#!/
#!/-
#!/
#!/ *****
#!/ VARIABLES
$ DOMAIN = $DOMAIN
#!/ *****
#!/
#!/ First check to see if this script if running on Version 2.0, if not exit
$ INREG = F$LOOKUPREGISTRY("HKLM","SOFTWARE\ASCII\XLNT\Current_Version\CurrentVersion",B,XLVER)
$ if XLVER .lts. "V2.0" then abort "XLNT V2.0 or greater is needed to run this script"
#!/
#!/ Open logfile
$ ON ERROR THEN GOTO ERROR_FATAL:
$ open outdat dormant.log /CREATE=CREATE_ALWAYS /WRITE
#!/
$GET_ANOTHER1:
$ USERNAME = F$ENUMUSER("DOMAIN",,CTX)
$ if F$EXTRACT(F$LENGTH(USERNAME)-1,1,USERNAME) .eqs. "$" then goto GET_ANOTHER1
#!/
$ WHILE USERNAME .nes. ""
#!/
$ llogin = F$USERINFO("USERNAME","DOMAIN",,"LOGON-LAST")
#!/
#!/ figure out the number of spaces to add between the name and the date
#!/ to look nice in the formatting
$ sph = ""
$ sp = 17 - f$length(USERNAME)
$ x = 0
$ for (x = 0, x .lt. sp, x = x + 1)
$     sph = sph + " "
$ endfor
#!/ The next line had to be added because of a formatting problem
#!/
$ if F$LOCATE("Never",llogin) .ne. F$LENGTH(llogin)
$ then
$     write $stdout ""USERNAME":'sph'Never"
$     write outdat ""USERNAME":'sph'Never"
$ else
$     lastlogon = llogin
#!/ Trying to convert date to real date:
$ months = "JAN|FEB|MAR|APR|MAY|JUN|JUL|AUG|SEP|OCT|NOV|DEC"
$ get_date = f$element(0,"",lastlogon)
$ date_month_temp = f$integer("f$element(0,/,",get_date)")
$ date_month_temp = date_month_temp - 1
$ date_month = f$element(date_month_temp,"|",months)
$ date_day = f$element(1,/,get_date)
$ date_year = f$element(2,/,get_date)
```

```

$ new_date = "date_day-'date_month'-19'date_year"
$ temp_date = f$integer(f$cvtime(new_date,"JULIAN"))
$ today_date = f$integer(f$cvtime("TODAY","JULIAN"))
$! now that we have them both, figure out the real difference
$! taking into account the year also
$ temp_date = ""temp_date"
$ today_date = ""today_date"
$ year1 = f$integer("f$extract(0,2,temp_date)") * 365
$ year2 = f$integer("f$extract(0,2,today_date)") * 365
$ days1 = f$integer("f$extract(2,3,temp_date)") + year1
$ days2 = f$integer("f$extract(2,3,today_date)") + year2
$ diff = days2 - days1
$ if diff .eq. 0
$ then
$     write $stdout ""USERNAME:"sphToday"
$     write outdat ""USERNAME:"sphToday"
$ else
$     write $stdout ""USERNAME:"sph"diff' days ago"
$     write outdat ""USERNAME:"sph"diff' days ago"
$ endif
$! *****
$! Here code can be added to do something based on the status of an account
$! I wouldn't do something if the account is never logged into. Maybe just
$! log it and double check it because something like the account IUSR_computer
$! never logs in with WWW, so if you disable the IUSR_computer account
$! then no one will be able to use the web.
$! The next piece of sample code displays a warning message if the user
$! has not logged in in 30 Days.
$
$ LAST_LIMIT = 30
$ if diff .gt. LAST_LIMIT
$ then
$     over = diff - 30
$     write $stdout "***** 'USERNAME' is 'over' days past the limit *****"
$     write outdat "***** 'USERNAME' is 'over' days past the limit *****"
$ endif
$!
$ endif
$
$!
$ GET_ANOTHER2:
$ USERNAME = F$ENUMUSER("DOMAIN",,CTX)
$ if F$EXTRACT(F$LENGTH(USERNAME)-1,1,USERNAME) .eqs. "$" then goto GET_ANOTHER2
$ endwhile
$!
$ ERROR_FATAL:
$ close outdat
$ exit

```

## 19.3 Simple F\$SEARCH Example Procedure

```

$!+
$! F$SEARCH - Sample script using the F$SEARCH Lexical
$! Script Version 1.0
$!
$! USER TYPE: Administrator or User
$!
$! SPECIAL RIGHTS REQUIRED: None
$!
$! DESCRIPTION:
$! Sample script using the F$SEARCH Lexical
$!
$!-
$ search:
$ afile = f$search("**.*", NEXT_FILE)
$ if afile .eqs. "" then goto done_file

```

```
$ write $stdout afile
$ goto search
$ done_file:
$exit
```

## 19.4 Sample LOGIN Script

```
$!+
$! LOGINV2.XCP - Login Script for Windows running XLNT V4.0
$! Script Version 2.5
$!
$! USER TYPE: Administrator or User
$!
$! SPECIAL RIGHTS REQUIRED: None
$!
$! DESCRIPTION:
$! This procedure uses both the new lexicals in Version 2.0 as well as
$! the RPC Services Found in XLNT V1.1 Service Pack 2 of XLNT to lookup
$! the account information of a person that is logged into an NT domain
$! and determine if they are in a certain group or not. Based on that
$! group the script can map a drive to the login name of the user or
$! any other drive that the administrator wishes. This script works on
$! Logging in from a Windows machine. Since this script utilizes
$! XLNT Version 4.0, all machines need to be at Version 4.0 to utilize
$! the power of this script. The client can have either XLNT the Language,
$! XLNT Batch, XLNT Professional or XLNT RunTime installed to execute this
$! script (It must be compiled in XLNT Professional to run in XLNT Runtime)
$!
$! INSTRUCTIONS:
$! In the variables section, change the ntmachine variable value
$! NT-MACH, to the name of the NT-MACHINE that is running XLNTSERV.
$! For ease this should be the Domain Controller. See notes for more
$! details. Right now the script checks if the user is in the "Domain
$! Admins" Group. If so, it will do the code that is in the if statement
$! Change this commented code to anything that you want to do once the
$! group is checked. Also see the notes on this section for what to
$! do for checking for multiple groups or also adding common code for
$! all groups.
$!
$! Advanced Features:
$! This script will first determine if the Machine is Windows NT
$! or Windows '95. It will run the appropriate commands
$! based on the operating system.
$!
$! Notes:
$! There is a problem when the machine that XLNTSERV is a workstation
$! that has a local user account of the same name that the user
$! is logging into the domain. This is a problem only with Windows '95
$! machines and the work around for it is to make just simply pick the
$! primary domain controller for the /on= parameter in the security
$! command
$!
$! Due to an bug in Windows '95 the script does not always exit
$! when it is completed. The script finishes to completion but then
$! leaves the NT script running message box on the screen. This
$! happens with any batch file, not only XLNT and does not happen
$! on an Windows NT Machine.
$!
$!-
$! *****
$! *****
$! VARIABLES
$!
$ ntmachine = "\\NT-MACH"
$!
$! *****
```

```

$! *****
$!
$ echo = "write $stdout"
$! Comment the following lines to debug login script
$ set noverify
$ set message /nomessage
$ on error then goto END_OF_SCRIPT
$ cls
$! Here we are going to check what type of operating system we have
$! and determine which commands to use if they are different in
$! Windows NT and Windows 95
$!
$ ISWINNT = 1
$ if f$getsyi("platform") .nes. "Windows NT" then ISWINNT = 0
$!
$! Now we are going to obtain the user name, local groups, and global groups
$! If you would like more information than this, just add the appropriate
$! lexical item code. This utilizes the new lexicals in Version 2.0 of XLNT
$!
$!
$ username = F$USERINFO("",$USERNAME",,"USERNAME")
$ localgroups = F$USERINFO("",$USERNAME",,"Local-Groups")
$ globalgroups = F$USERINFO("",$USERNAME",,"Global-Groups")
$!
$! Code to look for a certain local or global group
$! if you only want it to search for local or global groups just
$! comment out or delete one of the if statements. Also, if you would like
$! to do things for multiple groups, just copy and paste the block below
$! labeled GROUP SEARCH AND PROCESS and change the name of SGROUP to a group
$! of your choice
$!
$! *****
$! ***** GROUP SEARCH AND PROCESS *****
$! *****
$!
$!
$ SGROUP = "Domain Admins"
$ if F$GROUPINFO("",$SGROUP",,"IFMEMBER",,"username") .eqs. "TRUE"
$ then
$!   Add code here to do something if the person is in "Domain Admins" group
$ else
$!   Add code here to do something if the person is not in "Domain Admins" group
$ endif
$!
$! *****
$! ***** GROUP SEARCH AND PROCESS *****
$! *****
$!
$! COMMON TO ALL LOGON USERS
$!
$! Add code here to do something that is generic for all users like
$! mapping drives for showing the user specific computer or network
$! information
$!
$END_OF_SCRIPT:
$ exit
$ lo

```

## 19.5 List Windows NT Accounts Machine/Domain Procedure

```

$!+
$! NTACC.XCP - Enumerates NT User and Machine Accounts
$! Script Version 4.0
$!
$! USER TYPE: Administrator or User
$!

```

```

$! SPECIAL RIGHTS REQUIRED: None
$!
$! DESCRIPTION:
$! This script will enumerate all of the machines that have an account
$! on your NT domain. The difference between this and the Enumerate
$! Domains script is that it checks for all machines that are registered
$! not just the ones that are powered on.
$!
$! ADVANCED FEATURES:
$! This script will created a file called NTMACHINES.DAT that will store all
$! of the machine names in a list so that it can be used as a list to process
$! from any other script of your choice.
$!
$! ntacc.xcp Version 3 supports enumerating thousands of accounts.
$!
$! ntacc.xcp Version 3.11 supports printing all domain accounts to
$! a file called NTACCOUNTS.DAT.
$!
$! ntacc.xcp Version 4 corrected formatting differences between SP3 and SP4
$! Also added checking for SP Versions less than 2.
$!
$!-
$!
$! *****
$! *****
$!
$! Added Check here for Service Pack
$ extraread = 0
$ INREG = F$LOOKUPREGISTRY("HKEY_LOCAL_MACHINE","SOFTWARE\ASCII\XLNT\Current_Version\ServicePackVersion",B,SPVER)
$ SPVER = F$INTEGER(SPVER)
$ if SPVER .lt. 2 then abort "XLNT V1.1 Service Pack 2 or greater is needed to run this script"
$ if SPVER .gt. 3
$ then
$     extraread = 0
$ else
$     extraread = 1
$ endif
$
$!
$LCOUNT = 0
$!
$!
$write $stdout "Reading all accounts, please wait...."
$ SECURITY SHOW USER * /DOMAIN /OUT=NT.TMP /NOFORMAT
$! Find out the size of the temp file and read it in 500 byte chunks so that
$! it can be processed
$ OPEN INFILE "NT.TMP"
$ OPEN OUTFILE "NTMACHINES.DAT" /CREATE=CREATE_ALWAYS /WRITE
$ OPEN ACCFILE "NTACCOUNTS.DAT" /CREATE=CREATE_ALWAYS /WRITE
$ READ INFILE DATA1
$ READ INFILE DATA1
$ if extraread .eq. 1 then READ INFILE DATA1
$! We took care of all of the headers, now we have to take care of the machines.
$!
$ write $stdout DATA1
$ if extraread .eq. 1
$ then
$     READ/END_OF_FILE=JUMP_OUT1 INFILE DATA1 /DELIMITER=44
$     LCOUNT = LCOUNT + 1
$     MACH = F$ELEMENT(1,"=",DATA1)
$     mach2 = F$EDIT(MACH,"TRIM")
$     if F$ELEMENT(1,"$",mach2) .nes. "$"
$     then
$         write OUTFILE ""F$ELEMENT(0,"$",mach2)""
$     else
$         write ACCFILE ""F$ELEMENT(0,"$",mach2)""
$     endif
$ endif
$!
$endif
$FILE_LOOP1:

```

```

$ READ/END_OF_FILE=JUMP_OUT1 INFILE DATA1 /DELIMITER=44
$ LCOUNT = LCOUNT + 1
$ mach = F$EDIT(DATA1,"TRIM")
$ if F$ELEMENT(1,"$",mach) .nes. "$"
$   then
$     MACHTMP = F$ELEMENT(0,"$",MACH)
$     MACHNAME2 = F$EDIT(MACHTMP,"TRIM")
$     WRITE OUTFILE ""MACHNAME2"
$   else
$     MACHTMP = F$ELEMENT(0,"$",MACH)
$     MACHNAME2 = F$EDIT(MACHTMP,"TRIM")
$     WRITE ACCFILE ""MACHNAME2"
$   endif
$ GOTO FILE_LOOP1
$JUMP_OUT1:
$!
$ CLOSE INFILE
$ CLOSE OUTFILE
$ CLOSE ACCFILE
$! DEL NT.TMP
$END_OF_SCRIPT:
$ exit
$SCRIPT_ERROR:
$abort "SCRIPT ERROR: There was an error in NTACC.XCP, exiting" 0

```

## 19.6 Process INI Files Procedure

```

$!+
$! READINI.XCP - Sample Procedure to read INI files
$! Script Version 1.0
$!
$! USER TYPE: Administrator or User
$!
$! SPECIAL RIGHTS REQUIRED: None
$!
$! DESCRIPTION:
$! This procedure illustrates some of the use of Win32Api's to read INI
$! files.
$!
$!-
$!
$! Ok lets set the system base ie Windows drive and directory
$!
$ base = f$getvar("SystemRoot")
$!
$! Ok Lets check if the library is loaded
$!
$! = f$checklibrary("""base\system32\kernel32.dll")
$!if .not. !
$!then
$! Not loaded lets load it
$ kernel32=f$loadlibrary("""base\system32\kernel32.dll")
$ l = f$checklibrary("""base\system32\kernel32.dll")
$endif
$!
$! Now lets declare our functions
$!
$ DECLARE FUNCTION long "SetConsoleTitleA" kernel32 string STDCALL
$ DECLARE FUNCTION Ulong "GetProfileIntA" kernel32 string,string,integer STDCALL
$ DECLARE FUNCTION Ulong "GetPrivateProfileIntA" kernel32 string,string,integer,string STDCALL
$ DECLARE FUNCTION Ulong "GetPrivateProfileStringA" kernel32 string,string,string,&string,long,string STDCALL
$!
$! Lets set the Title
$!
$ Result = SetConsoleTitleA("Test Ini File functions")
$!
$! Assign the variables we will use

```

```

$!
$ s = "*****"
$ j = -1
$ j2 = -1
$!
$! Lets get to work
$!
$! Working with INI files
$!
$ j2 = GetPrivateProfileIntA("GX","Append_option2",3,"d:\asci\Int\new final\gx.ini")
$ j = GetPrivateProfileStringA("GX", "Append_option", "default", &s, F$length(s), "d:\asci\Int\new final\gx.ini")
$!
$! Lets see what we got
$!
$ write $stdout "Int: "j2"
$ write $stdout "Number of characters: "j"
$ write $stdout "String: "s"
$!
$! Free up some memory and release the DLL
$!
$ Result = F$FREELIBRARY(kernel32)
$ exit
$!
$! Contents of INI FILE BEIOW
$!
$! [GX]
$! Append_option=Chicken Hawk
$! Append_option2=15
$!

```

## 19.7 Machine Reboot Procedure

```

$!+
$! REBOOT.XCP - Reboot Remote WinNT system
$! Script Version 1.5
$!
$! USER TYPE: Administrator
$!
$! SPECIAL RIGHTS REQUIRED: None
$!
$! DESCRIPTION:
$! This procedure will remotely reboot a WinNT machine
$! (assuming you have the rights to do so).
$!
$! P1 = machine-name (to reboot). If not present then procedure will prompt for machine
$!
$!-
$!
$! Establish a NULL
$ NULL[0,8]=%x00
$! Set Base Directory
$ base = F$GETVAR("WINDIR")
$ if P1 .eqs. ""
$ then
$ INQUIRE COMP_NAME "Machine to reboot "
$ else
$ COMP_NAME = ""P1""
$ endif
$ if comp_name .eqs. ""
$ then write $stdout "Machine-name must be specified."
$ exit 0
$ endif
$ INQUIRE SHUT_TIME "Seconds to shutdown [5] "
$ if shut_time .eqs. "" then shut_time="5"
$ stop_time=f$integer(shut_time)
$ inquire reboot_restart "Restart after Reboot [Y] "
$ reboot=1

```

```

$ if reboot_restart .eqs. "" then reboot_restart="T"
$ if .not. reboot_restart then reboot=0
$ DECLARE FUNCTION dword "InitiateSystemShutdownA" advapi32 string,string,dword,dword,dword STDCALL
$ IF .NOT. F$CHECKLIBRARY("base\SYSTEM32\advapi32.dll")
$ THEN VER32=F$LOADLIBRARY("base\SYSTEM32\advapi32.dll")
$     IF .NOT. VER32
$     THEN
$     WRITE $STDOUT "$LOADLIBRARY failed for advapi32.dll"
$     END IF
$ ENDIF
$ x = COMP_NAME+NULL
$ xm = "SHUT DOWN by Remote Site '$local_machine' User:'$USERNAME'+NULL
$ REPLY = InitiateSystemShutdownA(x,xm,stop_time,1,reboot)
$ write $stdout "The Action resulted in: "REPLY"
$ RESULT = F$FREELIBRARY(VER32)
$ EXIT

```

## 19.8 Display System Information Procedure

```

$!+
$! SYSINFO.XCP - Windows System Information
$! Script Version 1.0
$!
$! USER TYPE: Administrator or User
$!
$! SPECIAL RIGHTS REQUIRED: None
$!
$! DESCRIPTION:
$! This procedure will give you a listing of all of the System Information about
$! the machine that it is run on.
$!
$!-
$!
$!
$!
$ WRITE $STDOUT " _____"
$ WRITE $STDOUT " "
$ WRITE $STDOUT " Computer Name: '$local_machine' "
$ WRITE $STDOUT " Computer UserName: '$username' "
$ WRITE $STDOUT " Operating System: "F$GETSYI("VERSION") Build: "F$GETSYI("BUILD_NUMBER")"
$ WRITE $STDOUT " Architecture: "F$GETSYI("ARCHITECTURE") Processor: "F$GETSYI("PROCESSOR_COUNT") -
"F$GETSYI("PROCESSOR_TYPE")"
$ WRITE $STDOUT " Local Time: "F$TIME() "
$ WRITE $STDOUT " Up-Time: "F$GETSYI("UP_TIME")"
$ WRITE $STDOUT ""
$ WRITE $STDOUT " _____"
$ WRITE $STDOUT ""
$ exit

```

## 19.9 Registry Procedure

This procedure demonstrates some simple Registry lexicals. In particular this procedure associates the extension .XCP with the XLNT image.

```

$!
$! REGISTER.XCP - Sample REGISTRY (lexicals) Procedure
$!
$! Description:
$!
$! This script will install a file association.
$!
$! You should change the path if it is not "c:\asci\xlnt"
$!
$! Lets look it up and see if it exists first.

```

```

$!
$ write $stdout "Checking Registry Information"
$ check = F$LOOKUPREGISTRY("HKEY_CLASSES_ROOT", ".xcp\\", var1, var2)
$!
$! Evaluate check value to see if Hive and Key exist.
$!
$ if check .eqs. 1
$ then
$     gosub Exist
$ Else
$!
$! Fail's to exist so lets create Hive, Key and Values.
$!
$     write $stdout "Creating Hive, Key and Values"
$     x = F$ADDREGISTRY("HKEY_CLASSES_ROOT", ".xcp\\", "REG_SZ", "XLNT_Script")
$     x = F$ADDREGISTRY("HKEY_CLASSES_ROOT", "XLNT_Script\shell\open\command\\", "REG_SZ", "c:\asci\xInt\xIntcli.exe @%1")
$     write $stdout "Job Completed"
$ endif
$ exit
$!
$! Hive and Key Exist so we will change values appropriately.
$!
$ Exist:
$     write $stdout "BOTH the HIVE and the KEY Exist."
$     write $stdout "Updating Values:"
$     x = F$CHANGeregistry("HKEY_CLASSES_ROOT", ".xcp\\", "REG_SZ", "XLNT_Script")
$     x = F$CHANGeregistry("HKEY_CLASSES_ROOT", ""var2\shell\open\command\\", "REG_SZ", "c:\asci\xInt\xIntcli.exe @%1")
$     write $stdout "Job Completed"
$     exit

```

## 19.10 Enumerate Domain/Machine Procedure

This procedure will enumerate all sharepoints, machine and domains in your network.

```

$!
$! ADMIN.XCP - Sample Domain Administrator Procedure
$!
$! Description:
$! This procedure illustrates some of the Enumerate lexicals
$! that are available in XLNT for determining domains, machines
$! and their sharepoints.
$!
$! NOTE: If you have a large domain, this procedure may run for
$! awhile.
$!
$! Example: @Admin "/all" to excute Combo
$ echo = "write $stdout"
$ if p1 .EQS. "/ALL" Then Call GetAllCombo
$ if p1 .EQS. "/SYS" Then Call GetAllSystems
$ if p1 .EQS. "/SRV" Then Call GetAllServices
$ if p1 .EQS. "/TRE" Then Call GetAllTree
$ if p1 .EQS. "" Then Call GetAllCombo
$ EXIT
$
$ GetAllSystems: SubRoutine
$     domain = "."
$     WHILE domain .nes. ""
$         domain = F$ENUMDOMAIN(dctx)
$         if domain .eqs. "" then leave
$         machine = "."
$         WHILE machine .nes. ""
$             machine = F$ENUMMACHINE(domain,,mctx)
$             if machine .eqs. "" then leave
$             ON ERROR THEN GOTO BYPASS
$                 show system /on='machine
$         BYPASS:
$             endwhile
$

```

```

$        endwhile
$ EndSubRoutine
$
$ GetAllServices: SubRoutine
$     domain = "."
$     WHILE domain .nes. ""
$     domain = F$ENUMDOMAIN(,dctx)
$     if domain .eqs. "" then leave
$     machine = "."
$     WHILE machine .nes. ""
$     machine = F$ENUMMACHINE(domain,,mctx)
$     if machine .eqs. "" then leave
$     ON ERROR THEN GOTO BYPASS
$     show service/on='machine
$     BYPASS:
$     endwhile
$     endwhile
$ EndSubRoutine
$
$ GetAllCombo: SubRoutine
$     domain = "."
$     WHILE domain .nes. ""
$     domain = F$ENUMDOMAIN(,dctx)
$     if domain .eqs. "" then leave
$     machine = "."
$     WHILE machine .nes. ""
$     machine = F$ENUMMACHINE(domain,,mctx)
$     if machine .eqs. "" then leave
$     ON ERROR THEN GOTO BYPASS
$     show system /on='machine
$     show service/on='machine
$     BYPASS:
$     share = "."
$     echo ""
$     echo "The available share points on "domain are:"
$     WHILE share .nes. ""
$     share = F$ENUMSHAREPOINT(domain,machine,,sctx)
$     if share .eqs. "" then leave
$     echo "|--"share"
$     endwhile
$     endwhile
$     endwhile
$ call GetAllTree
$ EndSubRoutine
$
$ GetAllTree: SubRoutine
$ echo ""
$ echo "Network tree created: "F$TIME()"
$ echo ""
$ domain = "."
$ WHILE domain .nes. ""
$     domain = F$ENUMDOMAIN(,dctx)
$     if domain .eqs. "" then leave
$     echo "--"domain"
$     machine = "."
$     WHILE machine .nes. ""
$     machine = F$ENUMMACHINE(domain,,mctx)
$     if machine .eqs. "" then leave
$     ON ERROR THEN GOTO BYPASS
$     echo " |--"machine"
$     BYPASS:
$     share = "."
$     WHILE share .nes. ""
$     share = F$ENUMSHAREPOINT(domain,machine,,sctx)
$     if share .eqs. "" then leave
$     echo " | |--"share"
$     ENDWHILE
$     ENDWHILE
$ ENDWHILE
$ ENDWHILE

```

```
$ EndSubRoutine
```

## 19.11 Enumerate Drives (WSH Script)

This XLNT WSH script enumerates network drive connections using the WSH object; Network.

```
$!+
$! EDRIVE.XCS - Script to Enumerate Network Drive using WSH
$! Script Version 1.0
$!
$! USER TYPE: Administrator or User
$!
$! SPECIAL RIGHTS REQUIRED: None
$!
$! DESCRIPTION:
$! Script to Enumerate Network Drive using WSH
$!
$!-
$ WSHNetwork = WScript.CreateObject("WScript.Network")
$!
$ NetPrint = WSHNetwork.EnumNetworkDrives
$ x = NetPrint.Count
$ numdrives = x / 2
$ write $stdout "There are 'numdrives' Network Drives"
$ write $stdout "-----"
$ write $stdout ""
$!
$ for (y = 0, y .lt. x, y = y + 2)
$ drivelet = NetPrint.Item('y')
$ sh = y + 1
$ sharename = NetPrint.Item('sh')
$ write $stdout "Drive 'drivelet' is connected to 'sharename'"
$ endfor
$ write $stdout ""
$ WScript.Quit(1)
$ exit
```

## 19.12 Display Environmental Variables (WSH)

This XLNT WSH Script displays all of the typical CMD environmental variables using the WSH Shell object.

```
$!+
$! ENVIRON.XCS - Script to display Environment Variables
$! Script Version 1.0
$!
$! USER TYPE: Administrator or User
$!
$! SPECIAL RIGHTS REQUIRED: None
$!
$! DESCRIPTION:
$! Script to display Environment Variables
$!
$!-
$ Shell = WScript.CreateObject("WScript.Shell")
$ SysEnv = Shell.Environment("PROCESS")
$ write $stdout "Number of Processors: "SysEnv.Item("NUMBER_OF_PROCESSORS")"
$ write $stdout "Processor Architecture: "SysEnv.Item("PROCESSOR_ARCHITECTURE")"
$ write $stdout "Processor Identifier: "SysEnv.Item("PROCESSOR_IDENTIFIER")"
$ write $stdout "Processor Level: "SysEnv.Item("PROCESSOR_LEVEL")"
$ write $stdout "Processor Revision: "SysEnv.Item("PROCESSOR_REVISION")"
$ write $stdout "Operating System: "SysEnv.Item("OS")"
$ write $stdout "Comspec: "SysEnv.Item("COMSPEC")"
$ write $stdout "Home Drive: "SysEnv.Item("HOMEDRIVE")"
$ write $stdout "Home Path: "SysEnv.Item("HOMEPATH")"
$ write $stdout "Path: "SysEnv.Item("PATH")"
```

```

$ write $stdout "Path Extensions:      "SysEnv.Item("PATHEXT")"
$ write $stdout "Command Prompt:      "SysEnv.Item("PROMPT")"
$ write $stdout "System Drive:        "SysEnv.Item("SYSTEMDRIVE")"
$ write $stdout "System Root:         "SysEnv.Item("SYSTEMROOT")"
$ write $stdout "Windows Directory:   "SysEnv.Item("WINDIR")"
$ write $stdout "Temp Directory:      "SysEnv.Item("TEMP")"
$ write $stdout "Tmp Directory:       "SysEnv.Item("TMP")"
$ exit

```

## 19.13 Optimized Display Environmental Variables (WSH)

```

$!
$ Shell = WScript.CreateObject("WScript.Shell")
$ SysEnv = Shell.Environment("PROCESS")
$ while (n .lt. 'SysEnv.Count)
$     write $stdout SysEnv('n)
$     n = n + 1
$ endwhile
$ exit

```

## 19.14 Shortcut Creation Script (WSH)

This XLNT WSH procedure creates a desktop shortcut.

```

$! A Script To Create ShortCut
$
$ Item = "welcome.exe"           ! Sample File for Shortcut
$ directory = ""f$getvar("windir")"
$ !*****
$ WSHShell = WScript.CreateObject("WScript.Shell")
$!
$ DesktopPath = WSHShell.SpecialFolders.Item("Desktop")
$ write $stdout " Desktop Path: "DesktopPath' "
$ write $stdout ""f$getvar("windir")\Item"
$!
$ !*****
$ MyShortcut = WshShell.CreateShortcut("DesktopPath\Shortcut to 'Item'.lnk")
$ MyShortcut.TargetPath = ""directory\Item"
$ MyShortcut.WorkingDirectory = ""directory"
$ MyShortcut.WindowStyle = 4
$ MyShortcut.FullName = ""DesktopPath\Shortcut to 'Item'.lnk"
$ MyShortcut.IconLocation = ""directory\Item', 0"
$ MyShortcut.Save
$ !*****
$ WScript.echo = "A shortcut to 'Item' now exists on your Desktop."

```

## 19.15 Sample Script Using Microsoft WORD Object

This procedure creates a small Word document using several Word methods.

```

$!+
$! WORD.XCS - Example of Using Word from XLNT using WSH
$! Script Version 1.0
$!
$! USER TYPE:      Administrator or User
$!
$! SPECIAL RIGHTS REQUIRED: None
$!
$! DESCRIPTION:
$! Example of Using Word from XLNT using WSH
$!
$!-
$ objWord = WScript.CreateObject("Word.Basic")

```

```
$ objWord.FileNew  
$ objWord.InsertDateTime  
$ objWord.FileSaveAs("xInt.doc")  
$ exit
```

## 20 Sales and Technical Support

For additional information on this product and others, contact Advanced Systems Concepts at:

**Voice:** (973) 539-2660

**Sales:** (800) 229-ASCI (U.S. and Canada)

**Fax:** (973) 539-3390

**Internet:**

**Sales:** [sales@advsyscon.com](mailto:sales@advsyscon.com)

This E-mail destination should be used for sales, pricing and other sales-related questions.

**Information:** [info@advsyscon.com](mailto:info@advsyscon.com)

This E-mail destination should be used for general product information requests which will result in an automatic reply of product information.

**Technical:** [support@advsyscon.com](mailto:support@advsyscon.com)

This E-mail destination should be used for technical product assistance.

**Web Site:** <http://www.advsyscon.com>

This destination represents Advanced Systems Concepts world-wide web site. Product information, downloads and other requests can be satisfied by visiting this site.

**FTP Site:** <ftp.advsyscon.com> (Web browser users may add "ftp://" to form a URL)

This destination represents the File Transfer site for Advanced Systems Concepts, product downloads.

See *About XLNT* menu item in XLNT WinHELP for software version release information.

## 21 Appendix A – Active Directory Country Codes

This appendix contains the ISO 3166 2-digit country codes for use with XLNT's Active Directory commands.

Country	Code
AFGHANISTAN	AF
ALBANIA	AL
ALGERIA	DZ
AMERICAN SAMOA	AS
ANDORRA	AD
ANGOLA	AO
ANGUILLA	AI
ANTARCTICA	AQ
ANTIGUA AND BARBUDA	AG
ARGENTINA	AR
ARMENIA	AM
ARUBA	AW
AUSTRALIA	AU
AUSTRIA	AT
AZERBAIJAN	AZ
BAHAMAS	BS
BAHRAIN	BH
BANGLADESH	BD
BARBADOS	BB
BELARUS	BY
BELGIUM	BE
BELIZE	BZ
BENIN	BJ

BERMUDA	BM
BHUTAN	BT
BOLIVIA	BO
BOZANIA AND HERZEGOWINA	BA
BOTSWANA	BW
BOUVET ISLAND	BV
BRAZIL	BR
BRITISH INDIAN OCEAN TERRITORY	IO
BRUNEI DARUSSALAM	BN
BULGARIA	BG
BURKINA FASO	BF
BURUNDI	BI
CAMBODIA	KH
CAMEROON	CM
CANADA	CA
CAPE VERDE	CV
CAYMAN ISLANDS	KY
CENTRAL AFRICAN REPUBLIC	CF
CHAD	TD
CHILE	CL
CHINA	CN
CHRISTMAS ISLAND	CX
COCOS (KEELING) ISLANDS	CC
COLUMBIA	CO
COMOROS	KM
CONGO, Democratic Republic of	CD
CONGO, People's Republic of	CG
COOK ISLANDS	CK
COSTA RICA	CR

COTE D'IVOIRE	CI
CROATIA	HR
CUBA	CU
CYPRUS	CY
CZECH REPUBLIC	CZ
DENMARK	DK
DJIBOUTI	DJ
DOMINICA	DM
DOMINICAN REPUBLIC	DO
EAST TIMOR	TL
ECUADOR	EC
EGYPT	EG
EL SALVADOR	SV
EQUATORIAL GUINEA	GQ
ERITREA	ER
ESTONIA	EE
ETHIOPIA	ET
FALKLAND ISLANDS (MALVINAS)	FK
FAROE ISLANDS	FO
FIJI	FJ
FINLAND	FI
FRANCE	FR
FRANCE, METROPOLITAN	FX
FRENCH GUIANA	GF
FRENCH POLYNESIA	PF
FRENCH SOUTHERN TERRITORIES	TF
GABON	GA
GAMBIA	GM
GEORGIA	GE
GERMANY	DE

GHANA	GH
GIBRALTAR	GI
GREECE	GR
GREENLAND	GL
GRENADA	GD
GUADELOUPE	GP
GUAM	GU
GUATEMALA	GT
GUINEA	GN
GUINEA-BISSAU	GW
GUYANA	GY
HAITI	HT
HEARD AND MC DONALD ISLANDS	HM
HONDURAS	HN
HONG KONG	HK
HUNGARY	HU
ICELAND	IS
INDIA	IN
INDONESIA	ID
IRAN (ISLAMIC REPUBLIC OF)	IR
IRAQ	IQ
IRELAND	IE
ISRAEL	IL
ITALY	IT
JAMAICA	JM
JAPAN	JP
JORDAN	JO
KAZAKHSTAN	KZ
KENYA	KE
KIRIBATI	KI

KOREA, DEMOCRATIC PEOPLE'S REPUBLIC OF	KP
KOREA, REPUBLIC OF	KR
KUWAIT	KW
KYRGYZSTAN	KG
LAO PEOPLE'S DEMOCRATIC REPUBLIC	LA
LATVIA	LV
LEBANON	LB
LESOTHO	LS
LIBERIA	LR
LIBYAN ARAB JAMANIRIYA	LY
LIECHTENSTEIN	LI
LITHUANIA	LT
LUXEMBOURG	LU
MACAU	MO
MACEDONIA	MK
MADAGASCAR	MG
MALAWI	MW
MALAYSIA	MY
MALDIVES	MV
MALI	ML
MALTA	MT
MARSHALL ISLANDS	MH
MARTINIQUE	MQ
MAURITANIA	MR
MAURITIUS	MU
MAYOTTE	YT
MEXICO	MX
MICRONESIA, FEDERATED STATES OF	FM

MOLDOVA, REPUBLIC OF	MD
MONACO	MC
MONGOLIA	MN
MONTSERRAT	MS
MOROCCO	MA
MOZAMBIQUE	MZ
MYANMAR	MN
NAMIBIA	NA
NAURU	NR
NEPAL	NP
NETHERLANDS	NL
NETHERLANDS ANTILLES	AN
NEW CALEDONIA	NC
NEW ZEALAND	NZ
NICARAGUA	NI
NIGER	NE
NIGERIA	NG
NIUE	NU
NORFOLK ISLAND	NF
NORTHERN MARIANA ISLANDS	MP
NORWAY	NO
OMAN	OM
PAKISTAN	PK
PALAU	PW
PALESTINIAN TERRITORY	PS
PANAMA	PA
PAPUA NEW GUINEA	PG
PARAGUAY	PY
PERU	PE
PHILIPPINES	PH

PITCAIRN	PN
POLAND	PL
PORTUGAL	PT
PUERTO RICO	PR
QUATAR	QA
REUNION	RE
ROMANIA	RO
RUSSIAN FEDERATION	RU
RWANDA	RW
SAINT KITTS AND NEVIS	KN
SAINT LUCIA	LC
SAINT VINCENT AND THE GRENADINES	VC
SAMOA	WS
SAN MARINO	SM
SAO TOME AND PRINCIPE	ST
SAUDI ARABIA	SA
SENEGAL	SN
SEYCHELLES	SC
SIERRA LEONE	SL
SINGAPORE	SG
SLOVAKIA (Slovak Republic)	SK
SLOVENIA	SI
SOLOMON ISLANDS	SB
SOMALIA	SO
SOUTH AFRICA	ZA
SOUTH GEORGIA AND THE SOUTH SANDWICH ISLANDS	GS
SPAIN	ES
SRI LANKA	LK
ST. HELENA	SH

ST. PIERRE AND MIQUELON	PM
SUDAN	SD
SURINAME	SR
SVALBARD AND JAN MAYEN ISLANDS	SJ
SWAZILAND	SZ
SWEDEN	SE
SWITZERLAND	CH
SYRIAN ARAB REPUBLIC	SY
TAIWAN	TW
TAJIKISTAN	TJ
TANZANIA, UNITED REPUBLIC OF	TZ
THAILAND	TH
TOGO	TG
TOKELAU	TK
TONGA	TO
TRINIDAD AND TOBAGO	TT
TUNISIA	TN
TURKEY	TR
TURKMENISTAN	TM
TURKS AND CAICOS ISLANDS	TC
TUVULU	TV
UGANDA	UG
UKRAINE	UA
UNITED ARAB EMIRATES	AE
UNITED KINGDOM	GB
UNITED STATES	US
UNITED STATES MINOR OUTLYING ISLANDS	UM
URUGUAY	UY
UZBEKISTAN	UZ

VANATU	VU
VATICAN CITY STATE (HOLY SEE)	VA
VENEZUELA	VE
VIET NAM	VN
VIRGIN ISLANDS (BRITISH)	VG
VIRGIN ISLANDS (U.S.)	VI
WALLIS AND FUTUNA ISLANDS	WF
WESTERN SAHARA	EH
YEMEM	YE
YUGOSLAVIA	YU
ZAMBIA	ZM
ZIMBABWE	ZW

## 22Index

- ! (Comment), 179
- .XCP, 66
- .XCS, 66
- := (String Assignment), 177
- @ (Execute Procedure) Command, 178
- = (Assignment Statement), 175
- ABORT Command, 180
- APPEND Command, 71
- Arrays, 26
- Associative Indexed Arrays, 26
- CALL Command, 181
- CLOSE Command, 73
- CLS Command, 183
- Collection Object, 68
- Command Interpreter Processing, 13
- Command Procedure Input, 45
- Command Procedure Output, 47
- CONFIGURE SERVICE Command, 150
- Controlling Execution Flow, 48
- COPY Command, 74
- CREATE Command, 77
- CREATE/DIRECTORY Command, 78
- Date and Time Format, 38
- DECK Command, 184
- DECLARE FUNCTION Command, 185
- DECLARE SYMBOL Command, 187
- Declared Symbols, 22
- Declaring Structures, 27
- Defining Foreign Commands, 19
- DELETE Command, 79
- DELETE/SYMBOL Command, 81
- DFS ADD, 115
- DFS REMOVE, 116
- DFS SET, 117
- DFS SHOW, 118
- DIFFERENCES Command, 82
- DIRECTORY Command, 85
- DISCONNECT FILE Command, 91
- DISCONNECT SESSION Command, 92
- Display and Deleting Symbols, 21
- DOS Command, 189
- DUMP Command, 93
- EDIT Command, 95
- Editing the XLNT Command Line, 16
- ENDFOR Statement, 190
- ENDSTRUCTURE Statement, 192
- ENDSUBROUTINE Command, 193
- ENDUNTIL Statement, 194
- ENDWHILE Statement, 195
- Entering an XLNT Command, 15
- EOD Command, 196
- EXIT Command, 197
- F\$ADDREGISTRY Lexical, 264
- F\$CHANGEREGISTRY Lexical, 265
- F\$CHECKLIBRARY Lexical, 266
- F\$CVSI Lexical, 267
- F\$CVTIME Lexical, 268
- F\$CVUI Lexical, 270
- F\$DELETEREGISTRY Lexical, 271
- F\$DIRECTORY Lexical, 272
- F\$EDIT Lexical, 273
- F\$ELEMENT Lexical, 274
- F\$ENUMDOMAIN Lexical, 275
- F\$ENUMGROUP Lexical, 276
- F\$ENUMMACHINE Lexical, 277
- F\$ENUMSHAREPOINT Lexical, 279
- F\$ENUMUSER Lexical, 280
- F\$ENVIRONMENT Lexical, 281
- F\$EXTRACT Lexical, 282
- F\$FILE\_ATTRIBUTES Lexical, 283
- F\$FORMAT Lexical, 285
- F\$FORMATDATE Lexical, 286
- F\$FORMATTIME Lexical, 288
- F\$FREELIBRARY Lexical, 289
- F\$GETBQI Lexical, 290
- F\$GETDVI Lexical, 294, 296
- F\$GETHOSTBYADDR Lexical, 298
- F\$GETHOSTBYNAME Lexical, 299
- F\$GETJPI Lexical, 300
- F\$GETLASTERROR Lexical, 302
- F\$GETSYI Lexical, 303
- F\$GETVARIABLE Lexical, 305
- F\$GROUPINFO Lexical, 306
- F\$INTEGER Lexical, 308
- F\$LENGTH Lexical, 309
- F\$LOADLIBRARY Lexical, 310
- F\$LOCATE Lexical, 312
- F\$LOOKUPREGISTRY Lexical, 312
- F\$MESSAGE Lexical, 314
- F\$MODE Lexical, 315
- F\$MSGBOX Lexical, 316
- F\$PARSE Lexical, 319
- F\$PERMISSIONS Lexical, 320
- F\$PID Lexical, 322
- F\$RANDOM Lexical, 324
- F\$READEVENT Lexical, 325
- F\$REPLACE Lexical, 327
- F\$REPORTEVENT Lexical, 328
- F\$SEARCH Lexical, 329
- F\$SEQARRAY Lexical, 330
- F\$SERVICE\_STATUS Lexical, 331
- F\$STRING Lexical, 333
- F\$TIME Lexical, 334
- F\$TYPE Lexical, 335
- F\$USERINFO Lexical, 336
- F\$VERIFY Lexical, 339

**File-Processing Command Summary**, 71  
 Files and Directories, 36  
 FOR Command, 198  
 FTP URL Specifications, 37  
 Functions and Lexicals, 34  
**GOSUB Command**, 201  
**GOTO Command**, 202  
**Handling Control/C Interruptions**, 44  
**Handling Error Conditions**, 42  
**Handling Error Conditions from DOS/CMD Programs**, 44  
**HELP Command**, 203  
 Hive Abbreviations, 263  
**IDE**, 51  
**IF Command**, 204  
**Implicit Symbols**, 24  
**INQUIRE Command**, 206  
**In-Script Data Processing**, 46  
**INSTALL SERVICE Command**, 153  
**Interacting with XLNT:**, 13  
**Interactive Development Environment**, 51  
 Introduction, 4  
 ITERATE Command, 207  
 KILL Command, 208  
 Label, 24  
 Language Statements, 175  
 LEAVE Command, 210  
**Lexical Commands**, 263  
 License Agreement, 1  
**Logical Operators**, 34  
**Logical Values and Expressions**, 33  
**LOGOUT Command**, 210  
**MAIL Command**, 97  
**MAIL SEND Command**, 97  
**MAIL SEND/SMTP Command**, 98  
 MANAGE ADD DRIVER, 120  
 MANAGE ADD PRINTER, 127  
 MANAGE CONNECT PRINTER, 130  
 MANAGE DELETE DRIVER, 122  
 MANAGE DELETE PRINTER, 131  
 MANAGE DISCONNECT PRINTER, 132  
 MANAGE SET JOB, 133  
 MANAGE SET PRINTER, 135  
 MANAGE SHOW DRIVER, 123  
 MANAGE SHOW MONITOR, 124  
 MANAGE SHOW PORT, 125  
 MANAGE SHOW PRINTER, 138  
 Numeric Indexed Arrays, 26  
 Numeric Operators, 34  
**Object Reference**, 30  
 object symbol, 30  
**ON Command**, 211  
**OPEN Command**, 100  
 Operators, 31  
 PAUSE SERVICE Command, 156  
**PRINT Command**, 102  
**Product Editions**, 5  
**READ Command**, 103  
 Reading and Writing Files, 47  
**RECALL Command**, 212  
**Registration and Support**, 12  
**Registry Entries**, 8  
 Registry Lexicals, 263  
 REMOVE SERVICE Command, 157  
**RENAME Command**, 105  
 RESUME SERVICE Command, 159  
**RETURN Command**, 213  
**RUN Command**, 214  
 SCHEDULE Command, 107, 108  
 SEARCH Command, 109  
 Security Commands, 140  
 SECURITY Create, 140  
 SECURITY Delete, 143  
 SECURITY Modify, 145  
 SECURITY SHOW, 148  
**Service Commands**, 150  
**Set / Show Commands**, 224  
 SET DEFAULT, 224  
 SET DOSERROR Command, 225  
**SET FILE Command**, 226  
 SET HOST Command, 228  
**SET HOST Installation**, 9  
**SET HOST Security and Registry**, 10  
 SET MESSAGE Command, 230  
 SET ON Command, 231  
 SET PERMISSIONS, 232  
 SET PREFERENCE, 236  
 SET PROCESS/PRIORITY, 238  
 SET PROMPT, 239  
 SET TIME, 240  
 SET VARIABLE Command, 241  
 SET VERIFY, 242  
 SHARE ADD, 163  
 SHARE Commands, 163  
 SHARE DELETE, 164  
 SHARE SET, 165  
 SHARE SHOW, 166  
 SHOW DEFAULT Command, 243  
 SHOW DEVICE, 244  
 SHOW DLLS, 246  
 SHOW HOST, 247  
 SHOW MACHINE, 249  
 SHOW MEMORY, 250  
 SHOW PERMISSIONS, 251  
 SHOW PREFERENCES, 253  
 SHOW PROCESS, 254  
 SHOW SERVER/FILES, 256  
 SHOW SERVICE Command, 160  
 SHOW SESSION, 257  
 SHOW SYMBOL, 258  
 SHOW SYSTEM, 259  
 SHOW TIME, 260  
 SHOW USER, 261

SHOW VARIABLE Command, 262  
**SPAWN Command**, 216  
START SERVICE Command, 161  
**STOP SERVICE Command**, 162  
**String Operators**, 31  
**STRUCTURE Command**, 217  
**SUBROUTINE Command**, 219  
Supported Datatypes, 25  
**Symbol Scope**, 28  
**Symbol Substitution**, 28, 29  
Symbols, 18  
System Requirements, 4  
**TYPE Command**, 113  
UNC Specifications, 36  
  
UNTIL Command, 220  
Using XLNT, 13  
**Using XLNT as a CGI**, 61  
**Using XLNT for Logon Script Processing**, 70  
**WAIT Command**, 221  
WHILE Command, 222  
**WRITE Command**, 114  
**Writing Procedures**, 41  
**XC Command**, 223  
**XLNT and DOS/WIN CMD**, 60  
**XLNT Command Syntax**, 15  
**XLNT Console Window**, 8  
**XLNT Debugger**, 51  
XLNT Mouse Handling, 16